Analyzing Symbols in Architectural Floor Plans via Traditional Computer Vision and Deep Learning Approaches

by

Alireza Rezvanifar

A Dissertation Submitted in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Electrical and Computer Engineering

University of Victoria

Analyzing Symbols in Architectural Floor Plans via Traditional Computer Vision and Deep Learning Approaches

by

Alireza Rezvanifar

M.Sc., Isfahan University of Technology, 2010

B.Sc., Islamic Azad University (Najafabad Branch), 2006

Supervisory Committee

Dr. Alexandra Branzan Albu, Supervisor

(Department of Electrical and Computer Engineering)

Dr. David Capson, Departmental Member

(Department of Electrical and Computer Engineering)

Dr. George Tzanetakis, Outside Member

(Department of Computer Science)

ABSTRACT

Architectural floor plans are scale-accurate 2D drawings of one level of a building, seen from above, which convey structural and semantic information related to rooms, walls, symbols, textual data, etc. They consist of lines, curves, symbols, and textual markings, showing the relationships between rooms and all physical features, required for the proper construction or renovation of the building.

First, this thesis provides a thorough study of state-of-the-art on symbol spotting methods for architectural drawings, an application domain providing the document image analysis and graphic recognition communities with an interesting set of challenges linked to the sheer complexity and density of embedded information, that have yet to be resolved.

Second, we propose a hybrid method that capitalizes on strengths of both vector-based and pixel-based symbol spotting techniques. In the *description* phase, the salient geometric constituents of a symbol are extracted by a variety of vectorization techniques, including a proposed voting-based algorithm for finding partial ellipses. This enables us to better handle local shape irregularities and boundary discontinuities, as well as partial occlusion and overlap. In the *matching* phase, the spatial relationship between the geometric primitives is encoded via a primitive-aware proximity graph. A statistical approach is then used to rapidly yield a *coarse* localization of symbols within the plan. Localization is further refined with a pixel-based step implementing a modified cross-correlation function. Experimental results on the public SESYD synthetic dataset and real-world images demonstrate that our approach clearly outperforms other popular symbol spotting approaches.

Traditional on-the-fly symbol spotting methods are unable to address the semantic challenge of graphical notation variability, i.e. low intra-class symbol similarity, an issue that is particularly important in architectural floor plan analysis. The presence of occlusion and clutter, characteristic of real-world plans, along with a varying graphical symbol complexity from almost trivial to highly complex, also pose challenges to existing spotting methods.

Third, we address all the above issues by leveraging recent advances in deep learning-based neural networks and adapting an object detection framework based on the YOLO (You Only Look Once) architecture. We propose a training strategy based on tiles, avoiding many issues particular to deep learning-based object detection networks related to the relatively small size of symbols compared to entire floor plans, aspect ratios, and data augmentation. Experimental results demonstrate that our method successfully detects architectural symbols with low intra-class similarity and of variable graphical complexity, even in the presence of heavy occlusion and clutter.

Contents

List of Tables

List of Figures

Acronyms

ARG Attributed Relational Graph. 14, 19–21, 26

BLP Binary Linear Program. 27

BoGP Bag-of-GraphPaths. 22

BoR Bag-of-Relations. 23

BSM Blurred Shape Model. x, 14, 16, 17

CBSM Circular Blurred Shape Model. x, 14, 16, 17, 24

CNN Convolutional Neural Network. 30, 69–72

CSS Curvature Scale-Space. 41

DIA Document Image Analysis. 1, 10

DL Deep Learning. vi, 5, 6, 8, 13, 29, 65, 68–70, 75, 80, 82, 83, 86

FGE Fuzzy Graph Embedding. 28

FSFV Fuzzy Structural Feature Vector. 28

FVG Full Visibility Graph. 14, 19

HPG Hierarchical Plausibility Graph. 14, 21

ILP Integer Linear Program. 27

IoU Intersection over Union. 58, 61, 75, 84

IP Integer Programming. 31

MPNN Message Passing Neural Network. 29

NCRAG Near Convex Region Adjacency Graph. 14, 22

QBE Query By Example. 3

RAG Region Adjacency Graph. 14, 21, 22, 27

RAST Recognition by Adaptive Subdivision of Transformation space. 25

RoI Regions of Interest. 10, 24, 26, 32

SCIP Shape Context descriptor computed on Interest Points. 14, 17, 18

SESYD Systems Evaluation SYnthetic Documents. iii, vii–xi, xiii, xv, 8, 11, 20, 29, 31–34, 42, 45, 55–59, 61–63, 65, 73, 74, 77–81, 83

SHT Standard Hough Transform. 44, 45

SSD Single Shot Multibox Detector. 69

SVM Support Vector Machine. 26

YOLO You Only Look Once. iv, vii, xiv, 8, 35, 68, 69, 71, 72, 80, 81, 83, 84

ACKNOWLEDGEMENTS

During my Ph.D., I had the pleasure to work with some wonderful people who have helped me to finish this journey. First and foremost, I am extremely grateful to my supervisor, Dr. Alexandra Branzan Albu, for all her kind supports, patience, guidance and understanding. Her helps were not limited to just my academic progress, but also to the other challenges of my life.

I would also like to thank my committee members, Dr. David Capson and Dr. George Tzanetakis, for their valuable comments and feedback that definitely improved this work.

A special thanks to Dr. Mélissa Côté, the research associate in our lab for her advice, mentorship and guidance towards this research. This work would simply have not been possible without her help.

I extend my thanks to all my fellow lab members and friends at the computer vision lab of the University of Victoria, Tunai Porto Marques, Amanda Dash and Maryam Alizadeh, who made this journey special and unforgettable.

My heartfelt and deepest gratitude to my family for their unconditional love and support, especially to my parents and my wife, Nasim.

Finally, this research was supported by Natural Sciences and Engineering Research Council of Canada (NSERC) and Triumph Electrical Consulting Engineering Ltd. through the CRD Grants Program. I am thankful of their support and appreciate Steven Cooke at Triumph for providing the real-world dataset and for his assistance in interpreting architectural drawings.

To the loves of my life:

my parents, my wife, Nasim, and my little Ryan ♥

Chapter 1

Introduction

1.1 Overview

Symbol recognition is a particular application of the general problem of pattern recognition, in which unknown input patterns are classified as belonging to one of many classes (i.e. predefined symbol types) in the application domain. According to [4], symbols can be defined as graphical entities which hold a semantic meaning in a specific domain, while being the minimum constituents that convey meaningful information. Logos, silhouettes, musical notes, and simple line segment groups with an engineering, electronics, or architectural flair constitute some examples of symbols that have been investigated by the Document Image Analysis (DIA) community.

The research on isolated symbol recognition is quite mature. Several comprehensive papers summarizing the state-of-the-art in symbol recognition can be found in the literature (see Table 1.1). However, the research on symbol recognition in context, i.e. the retrieval of symbols that are embedded in larger images as part of complex drawings, is still an open problem. Recognizing a symbol in context has many more practical applications than recognizing an isolated symbol, but also entails significant challenges. In early

Table 1.1: Key review papers on symbol recognition and spotting methods.

Survey	Recognition Methods	Spotting Methods	Categorization Approach
Chhabra [5], 1997	✓	✗	Application domains (circuit diagrams, engineering drawings, architectural drawings, etc.)
Kasturi and Luo [6], 1997	✓	✗	Application domains
Cordella and Vento [7], 2000	✓	Only segmentation approaches	Based on different stages of a general recognition method
Lladós *et al*. [8], 2001	✓	Only segmentation approaches	Application domains and pattern recognition methods (structural vs. statistical)
Tombre *et al*. [9], 2005	✓	✓	Challenges in the field and research directions
Rusiñol and Lladós [10], 2010	✓	✓	Based on different stages of a general spotting method
Tabbone and Terrades [11], 2014	✓	✓	Based on different stages of a general spotting method
Santosh [12], 2016	✓	✓	Statistical, structural, and syntactic approaches

attempts of symbol recognition in context, the retrieval process is mostly done by first using a segmentation step to extract regions of interest and then using a recognition step to cluster these regions. However, the main challenge here is the segmentation/recognition dilemma (known as Sayre's paradox [13], originally formulated in the context of an automated handwriting recognition system): a system should segment the symbols before recognizing them but, at the same time, some kind of recognition might be necessary to obtain a correct segmentation [14]. This dilemma may explain why the older survey papers in Table 1.1 did not cover retrieval methods or only talked about possible segmentation approaches. The second challenge is that due to the ever-growing size of document image repositories, fast(er) methods are necessary to handle large databases. To overcome these

bottlenecks, the research trend has gradually shifted from the traditional concept of symbol recognition toward the newer concept of symbol spotting, a way of locating a given query symbol within a graphical document image without using full recognition methods, while limiting the computational complexity [9]. As a consequence, symbol spotting has experienced a growing interest in the Graphics Recognition community in such a way that the evaluation of symbol spotting approaches was added for the first time in 2011 to the series of IAPR Workshops on Graphics Recognition (GREC) symbol recognition contests [15].

Symbol spotting methods are usually based on a Query By Example (QBE) approach [16]: an input symbol model is used as a query to locate similar symbols on architectural drawings. The output of a spotting method is a ranked list of similar symbols along with their localization data. This process can be more complex when no library of models is available at first and the input query is selected interactively by user. In this case, which is known as on-the-fly symbol recognition [17], learning-based methods cannot be used, and no initial assumption can be made about the shape of the input model. Finding and locating architectural symbols in context, i.e. as embedded elements of a realistic and complex complete architectural drawing, brings substantial difficulties to the analysis process due to the presence of clutter, connecting lines with other parts of the design, and overlaps with text and/or other symbols. The on-the-fly property of symbol spotting enables the user to select any desired image patch as query. The system then retrieves all the similar patches across the input floor plan. However, traditional on-the-fly symbol spotting methods are unable to address the semantic challenge of graphical notation variability, i.e. low intra-class symbol similarity, an issue that is particularly important in architectural floor plan analysis. The presence of occlusion and clutter, characteristic of real-world plans (i.e. architectural floor plans that are actually used for building construction projects), along with a varying graphical symbol complexity from almost trivial to highly complex, also pose challenges to existing spotting methods (some examples can be found in Figure 4.1).

This thesis provides several contributions to the filed of symbol spotting and detection in architectural floor plans, as outlined in the following section.

1.2 Thesis Objectives and Contributions

As mentioned, the vocabulary of symbols is application specific, covering a wide range of contexts from engineering documents to musical sheets. Thus, depending on the graphical structure of various vocabularies, different spotting/detection methods are required to reach the desired performance. One of the important applications of symbol spotting is the analysis of architectural floor plans. The obtained information from symbols of a floor plan convey crucial information that can be employed for some higher level semantic analysis like designing the different infrastructure layers of a building, vectorization of images, 3D reconstruction of designs, etc. The main objective of this thesis is to localize different architectural symbols (such as furniture, kitchen appliances, sinks, tubs, etc.) on 2D top-view architectural floor plans by detecting the minimal bounding box encompassing each symbol. Figure 1.1 shows the result of applying an ideal symbol spotting system to a synthetic floor plan.

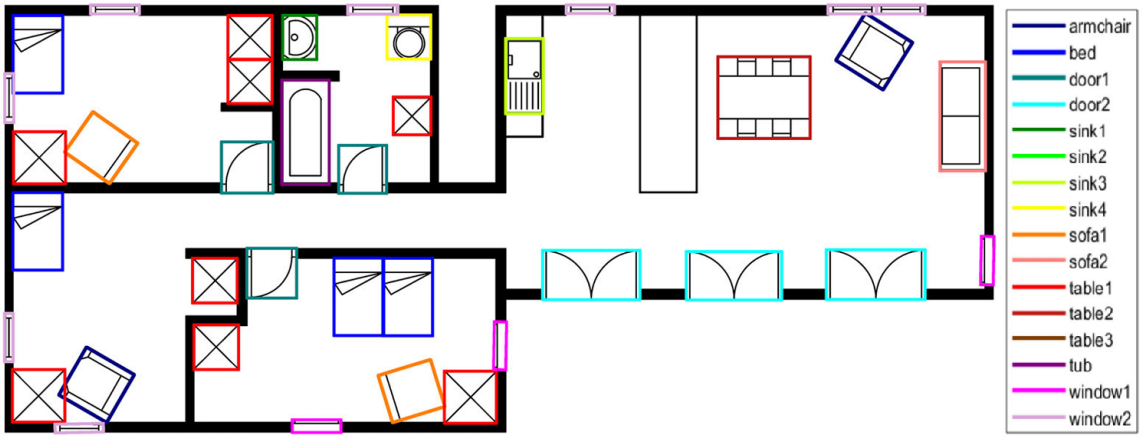


Figure 1.1: The expected output of a symbol spotting system. The bounding boxes around each symbol class are drawn by the same color.

This study is part of a comprehensive team project that aims to parse all the elements of a floor plan, such as rooms boundaries, room tags, location of walls and symbols and analyze the input floor plan based on this information to extract an infrastructure layer of a building. The rest of the system are/were developed by other researchers/students in our lab. This entire system is eventually going to be professionally developed for commercial purposes by our industrial partner.

The contributions of this thesis are as follows:

- We offer a thorough review of the literature on symbol spotting/detection advances with a specific focus on architectural drawings, a particularly challenging application domain [18].
- We propose a symbol spotting method that reaps the benefits of both pixel-based and vector-based approaches. A coarse localization of candidates using vector-based techniques is introduced first, then a pixel-based based method refines the detected instances [19].
- By relaxing the on-the fly property of symbol spotting, a Deep Learning (DL)-based framework is proposed to address the practical challenges of detecting symbols in real-world architectural floor plans [20].

These contributions will be articulated more in chapters 2, 3, and 4, respectively.

1.3 Publications

The contributions of this study led to multiple publications in journals and conference proceedings as follows:

1. **Symbol spotting for architectural drawings: state-of-the-art and new industry-driven developments** (published in [18]),

I was the first author of this work and my main responsibility was to collect and categorize publications and write the survey part. The proposed symbol spotting method in this paper was introduced by co-authors and will not be discussed in this thesis.

2. **Geometry-based Symbol Spotting in Born-Digital Architectural Floor Plans** (published in [19]),

   I was the first author of this work. The co-authors of this paper helped me with the structural design of the manuscript, writing and revising the paper in the submission process. They also assisted me with addressing the reviewers comments in the peer-review process, which majorly improved the initial submission.

3. **Symbol spotting on digital architectural floor plans using a Deep Learning-based framework** (published in [20]),

   I was the first author of this work. The co-authors of this paper helped me with the structural design of the manuscript, literature review, writing and revising the paper in the submission process.

4. **Automatic Generation of Electrical Plan Documents from Architectural Data** (published in [21]),

   This paper explores a novel application of document generation: the automatic creation of residential electrical plans from architectural data. Electrical plan documents, crucial to all residential construction and renovation projects, are currently generated manually by electrical designers who must adhere to the local electrical code and follow industry best practices. The designers decide the type and location of all household electrical devices and outlets based on the architectural floor plans. This is a tedious, highly repetitive, and time-consuming task. We propose a procedural approach to automate the generation of residential electrical plans via a stack-based finite state machine model

that mimics the electrical designer's thought process. The system receives 2D architectural data (e.g. wall location) as input and yields a customized electrical plan as output. Experimental results on a variety of architectural layouts of bathrooms, bedrooms, and kitchens are very promising and demonstrate the approach's functionality and usefulness. This paper paves the way for new algorithmic tools facilitating the design cycle of building projects.

As a co-author, I wrote the literature review of this manuscript and helped with the revision of paper. Since this work is beyond the scope of this thesis, we do not talk about it here.

The body of thesis is mainly structured based on the above publications (except paper #4). Each publication is discussed in an individual chapter.

1.4 Outline of Thesis

The body of this thesis is composed of three chapters which correspond to three papers, followed by a final chapter for conclusion. Please note that although each chapter is allocated to a stand-alone publication, the introduction and literature review sections of the papers are mainly moved to Chapter 2, in order to avoid redundancy along the thesis. Still, there are some overlaps between the chapters (publications) which is necessary to preserve coherency of the text.

The rest of this thesis is organized as follows:

Chapter 2 (Ref. [18]) offers an up-to-date literature survey on symbol spotting in architectural drawing images. The study categorizes the papers based on their contributions to different phases of a typical symbol spotting process, i.e., description phase and matching/spotting stage. Vector- and pixel-based descriptors are covered, by

highlighting their employed image features and the advantages and disadvantages of each. For the spotting phase of vector-based descriptors, graph matching techniques and their alternatives are investigated. Aside from symbol spotting, new DL-based symbol detection approaches are also reviewed. Finally, we discuss the performance of all methods which are evaluated on the SESYD dataset by their authors.

Chapter 3 (Ref. [19]) proposes a new symbol spotting method that performs well in real-world images, which are significantly more challenging than synthetic datasets in terms of overlap, occlusion and presence of non-symbolic graphical data. The method offers a new descriptor which finds the constituent geometric shapes of symbols and represents them as ellipse parameters. For this, we also introduce a new voting-based algorithm for finding partial ellipses. Then, a primitive-aware proximity graph followed by a statistical approach retrieve the query symbols from the input images. We performed some experimental results on both SESYD and real-world datasets and showcased the efficiency of the proposed pipeline in the presence of overlap, occlusion and non-symbolic graphical data.

Chapter 4 (Ref. [20]) focuses on symbol detection on real-world digital architectural floor plans with a DL-based framework. By relaxing the on-the-fly property of spotting approaches, this method can detect symbols with different notations. In this chapter, using our own real-world dataset, we trained YOLOv2 with a tiling strategy in order to solve problems particular to DL-based object detection networks related to the relatively small size of symbols compared to entire floor plans, aspect ratios, and data augmentation. The proposed method can offer automatic detection of objects without any user interaction and a better performance when dealing with real-world scenarios. The proposed method outperforms symbol spotting methods on the public SESYD dataset and performs well in detecting symbols with low intra-class similarity and of variable graphical complexity in real-world dataset.

Chapter 5 discusses the main takeaways of our research and offers recommendations for future work.

Chapter 2

Literature Review

In this chapter, we review the recent progress on symbol spotting with a specific focus on digital architectural floor plans as an application. An architectural floor plan (or architectural drawing) is a scaled two-dimensional diagram of one level of a building, consisting of lines, curves, symbols, and textual markings. Created by an architect, an architectural floor plan shows the relationships between rooms and physical features from above, and typically includes walls, doors, windows, staircases, fixtures, dimensions, labels of various kinds, and sometimes furniture. Figure 2.1 shows an example of the floor plan of a residential dwelling unit's bathroom and kitchen areas.

The analysis of digital graphics-rich documents such as architectural floor plans focuses on the automatic extraction of visual information initially intended for human comprehension. In particular, symbol spotting approaches aim to detect a ranked list of regions of interest (RoIs) which are more likely to contain a query symbol [22]. While a symbol always encodes domain-specific semantics, it may be generically defined as a minimum graphical entity that conveys meaningful information within a specific document [10]. Symbol spotting differs from symbol recognition, which intends to classify isolated symbols. Digital architectural drawings were noticed as being of particular interest in DIA communities, as

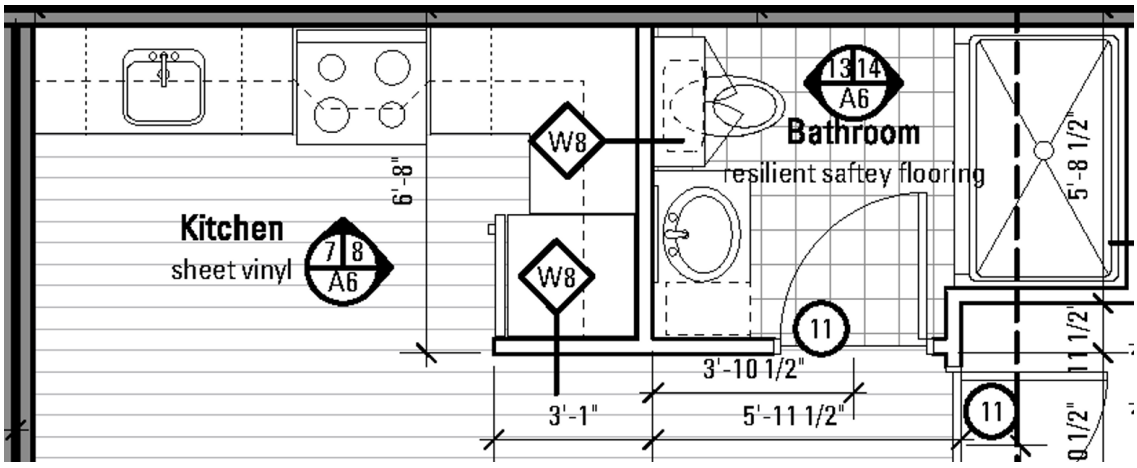


Figure 2.1: Part of an architectural floor plan (kitchen and bathroom areas of a residential dwelling unit).

can be seen in the Graphics Recognition Workshops and related conferences. For example, [23, 24, 25, 15] report on the four editions of the International Symbol Recognition Contest, which were held at GREC'03, GREC'05, GREC'07 and GREC'11, respectively. The main purpose of these contests was to provide some standard evaluation tools (datasets, ground truth data, evaluation metrics and evaluation protocols) in order to compare the performance of different symbol recognition methods (and symbol spotting methods in the latest edition). As a result, one of the most popular databases for symbol recognition and spotting, called Systems Evaluation SYnthetic Documents (SESYD)[1], was proposed in [14], focusing on synthetic architectural drawings and electrical diagrams, part of which was used to create the dataset for the latest contest edition [15].

In the literature, symbol recognition and spotting methods have been categorized differently based on the authors' viewpoint. Categorizations follow, for instance, the application domains, the various stages involved in the overall process, and the challenges in the field (see Table 1.1). In this chapter, we review papers according to their contributions with respect to the different steps involved in symbol spotting. The general framework of a symbol spotting method can be considered as consisting of three main levels [10]. First, the

[1] http://mathieu.delalandre.free.fr/projects/sesyd/

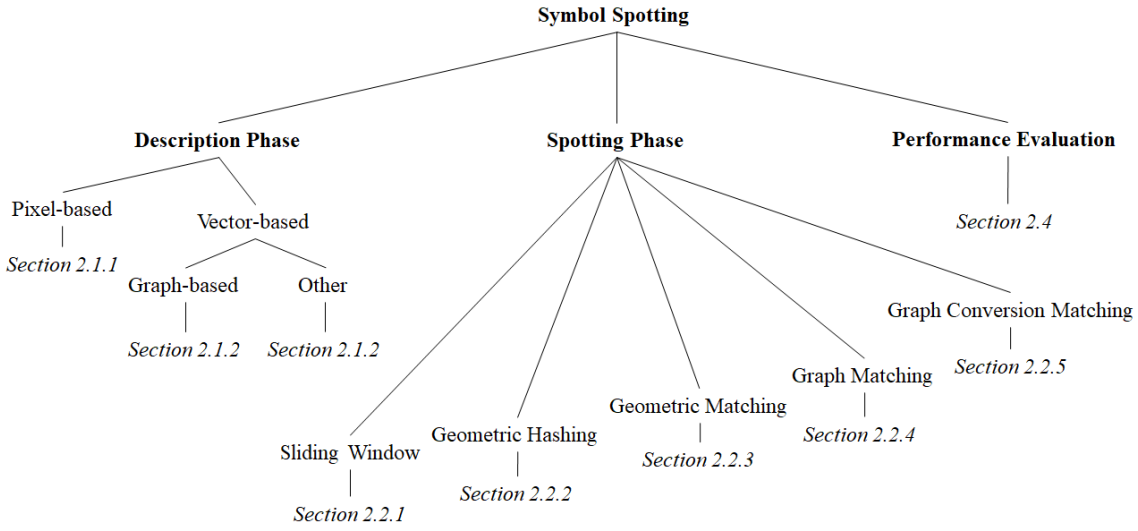


Figure 2.2: Categorization of methods. Hierarchical overview of processes and concepts in the reviewed methods and corresponding sections in this chapter.

query symbol and input document images are described via feature descriptors. The descriptors can either extract features based on image pixels or on some vectorial primitives such as arcs, lines, segments, etc. At the second level, descriptors are organized in such a way that they can be efficiently employed for the purpose of matching. In some methods, this organization requires the extraction of regions of interest which are more likely to contain symbols. Finally, at the third level, hypotheses arising from the matching between models and document images are validated and a ranked list of localized symbols is obtained. Recent symbol spotting methods mostly bring novelty to the first two steps since the validation step is usually subjective and application-dependent. So, for the purpose of reviewing and categorizing the literature, we adopt a framework consisting of two main steps: 1) description phase, and 2) matching/locating phase. Figure 2.2 gives an overview of the categorization of the reviewed symbol spotting methods and corresponding sections in the chapter.

The rest of this chapter is organized as follows: sections 2.1 and 2.2 review the literature on symbol spotting methods applicable to architectural drawings according to the

proposed 2-steps categorization. In particular, Section 2.1 targets papers contributing to the description level, and Section 2.2 to the matching/locating (spotting) level. Section 2.3 briefly reviews the recent advances in symbol detection systems based on the progresses in DL-based neural networks. Section 2.4 focuses on the performance of symbol spotting methods on architectural drawings and Section 2.5 discusses key findings from the literature review.

2.1 Symbol Spotting: Description Phase

As a first step of any symbol spotting method, much like for any other vision algorithm, the most significant and relevant information should be extracted from the document images for further processing. In this section, we review symbol information representations and descriptors which are used for architectural floor plan analysis. In [7], a distinction is made between the representation and the description of symbols as follows:

1. Representation phase: aims to reduce noise and extract only the most significant information from the images. Some examples are image skeletonization, polygonal approximation and Hough or other transforms.

2. Description phase: tries to use preliminary information provided by the representation phase to build a new descriptor.

Here, we opted to review methods tackling the description phase, since representation phase methods are typically low-level, have been well-studied before and are not specific to architectural floor plan images nor symbol spotting.

In architectural floor plan analysis, images are typically binary (bi-level) or grayscale and thus *shape* [49], as opposed to color for instance, is the most important visual cue for describing them. The main challenges present in this step are scale and rotation changes,

Table 2.1: Reviewed representations and descriptors for symbol recognition and spotting.

Paradigm*	Descriptor	Robustness and invariance	Features used
P	$\mathcal{F}$-signature [26]	Robust to geometric transformations and noise	Image pixels
P	Pixel-Level Constraint [27]	Scale and rotation invariant and robust to degradation	Image skeleton
P	Blurred Shape Model (BSM) [28, 29]	Robust to soft, rigid, and elastic deformations	Skeleton points
P	Circular Blurred Shape Model (CBSM) [30, 31],	BSM properties + rotation invariant	Contour map (flexible for other representations)
P	Shape Context descriptor computed on Interest Points (SCIP) and extensions [32, 33]	Scale and rotation invariant	Interest points
V	Perceptual grouping, Full Visibility Graph (FVG) [34]	Rotation and scale invariant	Vectorial primitives
V	Structural representation and Attributed Relational Graph (ARG) [35, 36, 37, 38]	Scale and rotation invariant, robust to small variations	Vectors and quadrilateral primitives
V	Hierarchical Plausibility Graph (HPG) [39, 40]	Robust to various distortions	Critical points and lines
V	Shape, topology, and Region Adjacency Graph (RAG) [41, 42]	Rotation and scale invariant	Image regions
V	Boundary and RAG [43]	Rotation and scale invariant	Image regions
V	Convexity and Near Convex Region Adjacency Graph (NCRAG) [44]	Rotation and scale invariant	Oriented line segments
V	Bag-of-GraphPaths (BoGP) [45]	Rotation invariant	Critical points
V	Jacobs' statistical grouping [46]	Scale and rotation invariant	Contour map
V	Bag-of-Relations (BoR) [47]	Scale and rotation invariant and robust to irregularities	Thick (solid) components, circles, corners and extremities
V	Cassinian ovals [48]	Not invariant to scaling and rotation	Polylines of closed region contours

* P = pixel-based, V = vector-based

occlusions, elastic deformations, as well as intra- and inter-class variations for symbols. Ideally, descriptors should be coarse to decrease the computation load of the matching/locating phase but at the same time accurate enough to yield a reasonable recognition rate. Several *ad-hoc* descriptors have been introduced in the literature to address those challenges. They can be divided into two main categories based on the type of primitives that are used for representing a query symbol: pixel-based vs. vector-based. Pixel-based descriptors work directly on the raster image format. They are usually associated with statistical approaches, and are typically (more) robust to noise. On the other hand, vector-based descriptors require the raster image format to be converted to vectorial primitives such as segments, polylines, arcs, etc. They typically allow for a more compact description and so faster matching phase, are usually associated with structural approaches (e.g. graphs), but sometimes lead to more false alarms and are more sensitive to noise. Table 2.1 summarizes descriptors according to this pixel-vector paradigm, which we review in the following subsections. Descriptors for both symbol recognition and symbol spotting are included as some relevant descriptors were proposed prior to the advent of symbol spotting. The table also mentions the low-level representation methods (corresponding to the representation phase in [7]) that are used in conjunction with each descriptor.

2.1.1 Pixel-based Descriptors

One of the earliest pixel-based signatures (compact representations) of line drawing images was proposed in [26] based on the idea of the $\mathcal{F}$-signature, which is a particular histogram of forces. This signature calculates exerted attraction forces, based on an attraction function, between different segments of a symbol in each direction θ. This description is invariant to fundamental geometric transformations such as scaling, translation, symmetry, and rotation, and is also able to handle symbol degradation to some extent. Although this descriptor is fast to implement with a computational complexity equal to $\mathcal{O}(p.n)$, where

n is the number of image pixels and p the number of digitalization steps of angles in the histogram, the descriptor does not lend itself easily to recognizing symbols in context, embedded in a floor plan.

In [27], geometric constraints such as length ratios or angles between pair of pixels considering a third pixel as the reference point, are summarized via histograms. These histograms are used to determine the similarity between symbols. Considering the constraints makes the descriptor rotation- and scale-invariant and also robust to degradation. One issue is that the complexity of the resulting algorithm is roughly equal to $O(n^3)$, and like $\mathcal{F}$-signature, it is not easily applicable to recognition in context.

The Blurred Shape Model (BSM) descriptor was first introduced in [28] for recognizing handwritten symbols, and further developed in [29]. To define BSM using skeleton points, the images are partitioned into a grid of $n \times n$ equal-sized sub-regions (where $n \times n$ identifies the blurring level allowed for the shapes). Each bin of the grid receives votes from the shape points it contains, and also from the shape points in the neighboring bins. The most interesting property of this descriptor is its robustness to elastic and non-uniform distortions. However, it cannot cope with geometric transformations like rotation and scale changes. According to the experiments in [28], BSM outperformed several other descriptors (e.g. Zernike descriptors) in terms of accuracy and scalability. In terms of computational complexity, for a region of $n \times n$ pixels, $k \leq n \times n$ skeleton points are considered to obtain the BSM with a cost of $O(k)$ simple operations, which is faster than the compared descriptors. To make BSM rotational invariant, an extension named Circular Blurred Shape Model (CBSM) was proposed in [30, 31]. CBSM is also able to handle irregular deformations and is fast to compute ($O(k)$ for k contour points). Using correlograms (radial distribution of sub-regions of the image) and shape features derived from the Canny edge detector, CBSM is defined based on a spatial arrangement of pixels (contour map). Figure 2.3 shows the grid structures being used for BSM and CBSM. The correlogram grid of CBSM makes it

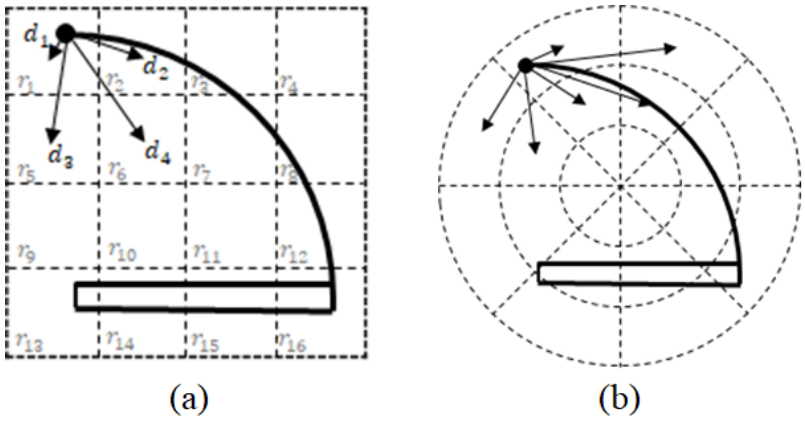


Figure 2.3: Pixel-based descriptors. Grid structures of BSM (a) and CBSM (correlogram) (b) for describing a door symbol. Each boundary pixel votes based on its distance to the adjacent points.

rotation invariant when the final result is rotated and aligned around the main diagonal of the correlogram with the highest density. Results show that CBSM can outperform many other descriptors, including the original BSM, for multi-class object categorization problems. Although CBSM was successful in making BSM invariant to rotation, occlusion and noise can highly affect the location of the main diagonal of the correlogram. Moreover, the calculation of the scale change remains a challenge.

Some symbol descriptors are inspired from the concept of visual words in text indexing/retrieval approaches. This concept has been studied in many works related to video/image retrieval (see for instance [50, 51, 52]). In [32], a new descriptor, known as Shape Context descriptor computed on Interest Points (SCIP), is proposed in order to describe a graphic symbol using visual words. The shape context of a contour point is defined based on the bivariate histogram (distance and angle) of the relative coordinates of neighboring contour points. Since the number of contour points may be huge in real documents, shape context is only computed on key points (for instance obtained via the difference of Gaussian operator, or DoG). SCIP can adaptively reflect the local geometry of symbols based on their complexity and details, and also guarantees invariance to scaling and rotation for isolated symbols. Unfortunately, the rotation and scale invariance is

accomplished via a normalization step at the symbol level, and it is therefore difficult to apply the descriptor at the document level, i.e. for recognition in context. For this reason, an extension of SCIP (sometimes called ESCIP in some papers, such as in [53]) for the document level was introduced in [33]. In that paper, ESCIP is used to describe a document with a library of visual words. This allows for floor plans to be processed as text documents, thus text retrieval/indexing techniques can be applied. Although the experimental results of this method are promising, false detections are a major issue, due to spatial relations between visual words being ignored, and to an instability of the key points detection in the case of symbols composed of curves.

2.1.2 Vectorial Descriptors

Considering the fact that architectural floor plans are human-made and digitally born, symbols are usually composed of regular primitives such as lines, arcs, polylines, etc. For this reason, a vectorial representation of symbols may seem appropriate to describe their shape. Typically, a vectorial representation of the meaningful parts of document images and symbols is first obtained via primitives, which are then further grouped via a suitable vector-based descriptor. We further categorize vectorial descriptors into graph-based and other, due to the prominent role of graph structures.

Graph-based

Graph structures constitute one popular way of utilizing vector-based descriptors. Although they cannot be considered themselves as descriptors, they constitute a flexible and powerful tool for describing the relational information found in architectural symbols. For that reason, we introduce here some related background information on graph structures typically used in the description phase for symbol spotting in architectural floor plan images. The main advantage of using graphs is their capability to represent individual symbols with

variable size and complexity. This is because a graph's size (number of nodes and edges) does not have to be set *a priori* and can be adjusted depending on the data. Moreover, graphs can simultaneously encode both the numeric properties and the structural data of a symbol. This latter property is what makes graphs popular in structural pattern recognition. To define nodes and edge attributes, scale and rotation invariant descriptors, in addition to relational information, are typically used in a way that makes the final description scale and rotation invariant, to avoid putting an extra computational load on the graph matching step. A drawback of using graph structures for describing objects (or symbols) is that pattern recognition (or symbol spotting) is converted to a subgraph matching (or isomorphism) problem which is NP-complete. Another disadvantage of graphs is their sensitivity to noise, as noise can affect the adjacency and Laplacian matrices of the graphs, and thus may negatively affect the final performance [54]. Nonetheless, graph structures remain highly popular for vector-based symbol description and spotting. Figure 2.4 shows several graph-based representations of a bed symbol.

In [34], vectorial primitives are considered as the nodes of a graph called Full Visibility Graph (FVG), and edges represent the visibility between every pair of primitives. Figure 2.4c shows an example of FVG for the bed symbol of Figure 2.4a. Visibility here refers to the existence of at least one straight line between two points on the primitives such that the line does not touch any other primitive in the drawing. Perceptual grouping of the primitives can then be done by applying clique detection to FVG. In [35], a structural representation of line drawing images is proposed based on polygonal and quadrilateral approximations of the symbol's contours. After this vectorization step, a structural graph is constructed based on different types of interactions between vectors in order to organize the extracted vectors more appropriately. More specifically, the graph is an Attributed Relational Graph (ARG) as it extracts topological and geometric features. Figure 2.4d shows an example of ARG for the bed symbol considering the attributes defined in Figure 2.4e. In this figure, L, T

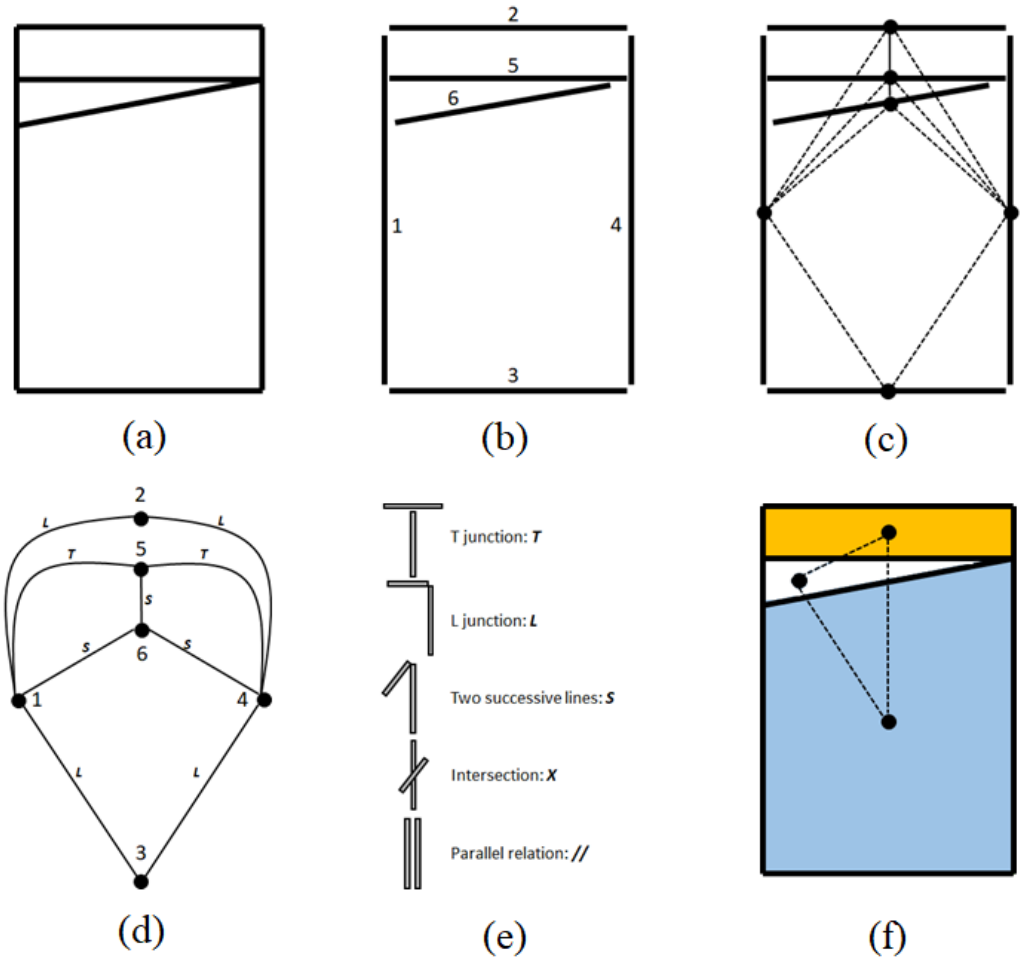


Figure 2.4: Graph-based descriptions of a bed symbol. Bed symbol from SESYD (a), vectorized image (b), Full Visibility Graph (FVG) (c), Attributed Relational Graph (ARG) (d), intersection types used for ARG (e), and Region Adjacency Graph (RAG) (f).

and S show an L junction, a T junction, and two successive lines, respectively. The relative geometric features considered in ARG allow for invariance to rotation and scale changes, and also make the graph robust to small variations in the symbols.

ARGs have also been utilized in hybrid symbol recognition approaches in order to integrate structural and statistical methods. In [36], a symbol is described by a set of spatio-structural descriptors (vocabulary) associated with visual primitives such as corners, circles, thick primitives, and extremities. This description is mainly inspired from [55, 56], where both complete symbols and symbols with missing parts are recognized using their

vocabulary and grammar. The ARG-based spatio-structural description in [36] combines both topological and directional information specific to the symbol. The label assigned to each ARG vertex is the class of the corresponding extracted vocabulary, while the edge label is defined based on the spatial relation between two endpoint primitives. Two extensions of this method are proposed in [37] and [38] to include global shape signatures of each vocabulary class in the constructed ARG. By using a set of fixed and completely labelled attributes, NP-hardness of the matching problem can be avoided [36]. However, the underlying graph matching problem requires the symbols to have at least two different types of visual primitives in order to compute the spatial relations. Thus, the method cannot handle symbols containing only one primitive type [47].

One of the difficulties when working with graphs is their sensitivity to the vectorization step. Structural errors in this step can easily result in spurious nodes, edges, and some disconnections between different parts of the graph, yielding to poor results. To cope with this problem, a hierarchical graph representation known as Hierarchical Plausibility Graph (HPG) is introduced in [39, 40] to cover different possible vectorizations. HPG is able to solve three kinds of distortions: split nodes, dispensable nodes, and gaps. Although HPG can address the distortion issue under these circumstances, heavy distortion is still a major concern. In addition, building an HPG is time consuming and the utilized hierarchical matching algorithm sometimes fails to find local optima, yielding erroneous solutions.

Another type of graph representation called Region Adjacency Graph (RAG) can be created using regions of the floor plan images (e.g. based on connected components) as graph nodes, and connecting the adjacent regions via graph edges (Figure 2.4f). Typically, the characteristics of the region boundaries and the relational information between regions are used to label graph nodes and edges. Examples include boundary strings [57], Zernike moments for nodes and relative scale and distance for edges [41, 42], and histogram of distances between border points and centroids [43]. The main drawbacks of RAG is that it

only considers closed regions as components of symbols and can fail to represent a symbol well in case of occlusion and discontinuities in the boundaries. This problem is especially important when working with real-world architectural floor plan documents. Another serious problem with RAG is that it cannot properly represent symbols with no salient closed region in their structure.

To tackle the problem of RAG with closed regions, the Near Convex Region Adjacency Graph (NCRAG) is proposed in [44], in which regions do not need to be clearly and continuously bounded. NCRAG rather focuses on the convexity of the different parts of a symbol, as convexity is one key symbol property. Even though NCRAG improves upon RAG, it still has limitations with respect to small variations of symbol instances in the architectural drawing, which can cause erroneous splits or merges between neighboring regions.

Since graph-matching, which seems to be required for spotting phase of graph-based representations, is NP-complete, [45] proposed a Bag-of-GraphPaths (BoGP) descriptor that finds all acyclic paths between any two connected nodes of a graph representation of symbols, and then uses Zernike moments to describe these paths. The BoGP descriptor inherits the rotation invariance properties of graph paths. It is also unique in the sense that the descriptor converts the graph matching step to a statistical problem, which can be solved via hashing methods.

Other

While graph-based representations are popular for vectorial descriptors, other representations have also been proposed in the literature. One of the earliest methods for representing models in architectural plans is to consider a set of constraints between model segments obtained from a vectorization step (geometrical features) [58, 59, 60]. Taking inspiration from the work in [61, 62], these constraints are passed to a network of constraints and symbols

are detected by testing constraints on different nodes of the network. In [46], the authors apply Jacobs' statistical grouping algorithm [63] to find meaningful (or salient) convex groups of geometric primitives, as convexity is considered as one of the non-accidental properties of line segments. Likewise, the approach in [47] avoids the use of graphs by extracting a meaningful vocabulary of visual primitives such as thick (solid) components, circles, corners and extremities. The vocabulary is then used to build a Bag-of-Relations (BoR) based on pairwise topological and directional relations between visual primitives. Since the description is built from information on the spatial relations of symbols, it is robust to scaling and rotation changes, and also to irregularities. In addition, the use of BoR indexing reduces the computation time during the recognition process, which is the main problem of graph-based representations. The authors in [48] extract polylines of closed region contours of each symbol as primitives, which are then described with Cassinian ovals. Cassinian ovals were originally introduced in [64] to model the orbit of the Sun around the Earth. The use of this descriptor is an attempt to encode the eccentricity and non-circularity of an object. Using this descriptor, each polyline is encoded in terms of a tuple (a,b). This few-digit descriptor is then employed to build a hash table for indexing the input document. Unfortunately, Cassinian ovals are not invariant to scaling and rotation changes, and are only suitable for a subset of architectural symbols, i.e. those comprised of only closed regions.

2.2 Symbol Spotting: Matching/Locating Phase

Once symbols are described by an appropriate descriptor, the most difficult challenge of a symbol spotting method is finding the embedded symbols in the documents in a way that avoids a segmentation step. To accomplish this matching step, one has a range of options, from a trivial brute-force method, such as a sliding window, to more complicated

mechanisms like graph matching. The remainder of this section reviews matching methods for locating symbols in context, which we have categorized as follows: sliding windows and correlation matching, geometric hashing, geometric matching, graph matching, and graph conversion matching.

2.2.1 Sliding Windows and Correlation Matching

Searching the entire floor plan image via a sliding window (with or without overlap) performs an exhaustive search but is very time-consuming. In addition, depending on the robustness of the symbol descriptor, it may be necessary to search the image using windows with varying scales and rotations. Correlation-based methods like [65] which proposed a variation of the Hit and Miss transform for template matching, and some descriptors like CBSM [30] (see section 2.1.1) and vectorial signatures [66] use this method to locate Regions of Interest (RoI) in input images.

One way of avoiding the unreasonable computational complexity of exhaustive searches is to limit the search area to the most relevant parts of image. For instance, [26] applies a text string separation technique and implements the matching step on connected components. Like other matching methods relying on some form of pre-segmentation, such as connected components, or trying to detect RoIs first, there is an assumption that the symbols can be found as one entity within the components or RoIs, which is not necessarily true.

2.2.2 Geometric Hashing

Indexing methods constitute another popular approach for spotting symbols within architectural drawings; indexation usually involves encoding the symbols' and input documents' primitives via hashing functions in the form of lookup tables. The existing literature on primitive indexing methods usually follows the idea of geometric hashing introduced by

[67]. In [68], the contours of the closed regions composing a symbol are described by a chain of adjacent polylines, then transformed into attributed cyclic strings. Two polylines are thus compared and matched by taking into account the different string edit operations needed to transform a string into another. To build an indexing lookup table for locating symbols, representatives of similar strings are used as indexing keys. Another example of hashing approaches consists in choosing a digit descriptor obtained from Cassinian ovals to describe symbols and building an indexing hash table [48].

In [69], the authors propose a structural approach for indexing vectorial drawings. In their method, a proximity graph, which is built based on extracted primitives from the image, is used to save not only the spatial relationship between primitives but also their numerical description in a 2D hash table. This approach, called relational indexation, allows for the consideration of spatial relationships between primitives in the retrieval process. Considering spatial relationships makes this approach invariant to scale and rotation transforms. One downside is that spatial information is only limited to the proximity (adjacency) between primitives, which might yield the same representations for different symbols.

2.2.3 Geometric Matching

Nayef and Breuel in [70] use a geometric matching technique [71, 72] known as Recognition by Adaptive Subdivision of Transformation space (RAST) to search the space of the transformations for the most likely transformation parameters and thus the location of symbols in the document. In order to find matches of the features of a symbol model within a floor plan image, RAST performs a branch-and-bound search which yields a globally optimal solution. The authors employ either sample points along all lines in the drawings and symbol images, or segments and their orientation as feature points. They improved their geometric matching framework in [73] so that it is able to deal with vectorial primitives such as lines and arcs in addition to pixels. As an additional improvement, non-matched

features are also penalized in order to decrease the number of incorrect matches. This approach has a relatively low precision when dealing with noisy or older documents, so a solution is presented in [74], which adds a post-spotting module in order to refine results. In this module, candidate regions obtained from the geometric matching algorithm are classified as true and false matches using a Support Vector Machine (SVM) classifier.

2.2.4 Graph Matching

As mentioned in section 2.1.2, graph-based representations convert the problem of symbol spotting to a subgraph-matching problem; however, finding the optimal solution is not feasible since it is an NP-complete problem. Accordingly, several solutions have been proposed in the literature to find a reasonable solution. In [75], a new scoring function is proposed to evaluate the similarity of two different graphs. This scoring function employs a greedy graph matching algorithm to avoid the implementation of an exhaustive search.

In [76], an ARG representation is integrated with the graph matching technique proposed in [77] to build a complete symbol spotting framework. To decrease the computational complexity of the spotting step, a set of hypotheses are considered by the authors for detecting RoIs in the input architectural drawing images (or equivalently, detecting the parts of the corresponding graph that may include a symbol). This way, the graph matching only compares the model with RoIs, instead of the entire document. Although the extraction of RoIs can considerably decrease the computational complexity, the final performance of the algorithm is limited by the quality of this localization step. Indeed, the more comprehensive the hypotheses are, the better precision and recall can be expected, so the hypotheses should be properly defined in a way that no symbol will be missed, while being able to cope with occlusion and noise.

In [42], the authors propose a substitution-tolerant subgraph isomorphism to solve symbol spotting in technical drawings. Here, substitution tolerance means that the matching can

cope with attribute differences such as vertex and edge labels, but a one-to-one mapping has to exist between each vertex and each edge of the template graph (symbol) and the target graph (floor plan), thus structural distortions are not handled. In their approach, architectural floor plan images are described with a RAG, and subgraph isomorphism is modeled via an Integer Linear Program (ILP) optimization problem. To generalize this approach for handling structural distortions, [78] proposes a Binary Linear Program (BLP) which allows for the deletion of vertices and/or edges in the template graph. Unfortunately, there is no polynomial-time algorithm for solving ILP and BLP. The authors utilize Mathematical Programming (MP), which provides tools for solving optimization problems. In order to reduce the search space, these programs use a branch-and-bound algorithm along with some heuristics. Experimental results show better matching performance for BLP as well as faster solving time. The main drawback of these subgraph isomorphism approaches is that they remain unsuitable for large databases, since subgraph matching requires the investigation of the entire graph corresponding to the floor plan image for each symbol. Indexation techniques have thus the advantage over subgraph matching to be able to describe the input drawing just once in an offline process, allowing for a faster online querying. Another disadvantage of subgraph isomorphism approaches is that solving the optimization problem only yields the optimal solution in each implementation, which means that only the best match can be found in each iteration. The entire procedure must therefore be repeated several times for each symbol, ignoring the previous optimal solutions.

2.2.5 Graph Conversion Matching

To reduce the computational cost of graph matching, several works focus on translating graph characteristics into a statistical space and then use a statistical method for locating symbols, as opposed to structural approaches. We refer to this group of techniques as graph conversion matching.

Fuzzy Graph Embedding (FGE) is one of the conversion methods used for symbol spotting in [79, 80]. Graph embedding is a dimensionality reduction method which aims to exploit significant structural and statistical details of an attributed graph for embedding it into a feature vector (point) in a vector space. Feature vectors should reflect the similarity of corresponding graphs, i.e. the more similar two graphs are, the smaller the distance between their corresponding feature vectors should be. Graph embedding has the interesting advantage of enabling graph-based representations to access the whole range of statistical classifiers and machine learning tools. The resulting feature vector in the FGE approach is termed as Fuzzy Structural Feature Vector (FSFV), containing graph, node, and edge level features. Graph level features include graph order and size; node level features include fuzzy histograms of node degrees and of values taken by node attributes; edge level features include a fuzzy histogram of values taken by edge attributes. FGE employs fuzzy overlapping intervals for minimizing the information loss while mapping from continuous graph space to discrete vector space. The main drawbacks of the method in [80] are twofold: first, the presence of some imperfections in the representation of crossing lines or lines with angular points and second, a potential unwanted decomposition of a line into several quadrilaterals. The interested reader can refer to [54] for more information about graph embedding approaches.

Another alternative to graph matching is converting the geometric information carried by a graph to one-dimensional structures (serialization), which are less computationally expensive [81, 82, 83]. In these works, graph nodes are the critical points detected in the vectorized graphical documents and the lines joining them are considered as the edges. Serialization is accomplished by factorizing the graph into a set of all the acyclic paths between each pair of connected nodes. After this factorization, the paths are described by a proper descriptor and symbol spotting can be done through hashing the shape descriptors of the graph paths. The factorization step introduces a lot of extra paths which can help

the algorithm in handling distortion and noise, but on the other hand, it creates a lot of redundant information. In addition, since critical points are highly sensitive to noise, the serialization process can affect the final performance.

2.3 DL-Based Symbol Detection

Coping with notation variability of symbols remains a significant semantic challenge for traditional symbol spotting methods. Indeed, although some methods are relatively successful in dealing with noise, occlusion and clutter in the image [40, 84, 38], they are not capable of detecting symbols with low intra-class similarity. Non-traditional, DL-based methods have recently started to permeate the literature. For instance, in [85], the authors propose a shallow CNN for recognizing hand-drawn symbols in the context of multi-writer scenarios. In [86], the authors utilize a Message Passing Neural Network (MPNN), which is a graph neural network, to globally describe the structural graph representation of symbols and then use the output for graph matching. Testing in both [85] and [86] is limited to symbol recognition, as localization in context is problematic. Also, MPNNs are typically useful for dense graphs and do not yield the same performance on sparse graphs, which are more common for our application domain. More recently, Ghosh *et al.* [87] proposed GSD-Net, a compact network for pixel-level graphical symbol detection. They use a semantic segmentation network, which labels all pixels individually as opposed to extracting bounding boxes around objects of interest. Such a method requires expensive pixel-level annotations. The authors also trained their network on the public SESYD dataset [15], which is a set of synthetic images with less challenges of the real-world architectural floor plans (more details about this dataset in Sec. 3.2.1). In particular, SESYD does not include occlusion, clutter, nor any symbol intra-class graphical variability. Ziran and Marinai [88] and Goyal *et al.* [89] both utilized object detection networks for symbol spotting. Their

experiments, focused on simple floor plans, did not allow for a performance assessment under heavy occlusion and clutter.

Figure 2.5: Upper and middle rows: some examples of the employed datasets in [2] and [3], respectively. Lower row: some real-world occluded images with higher complexity.

More recently, the traditional approaches for detecting different elements of floor plans started to be replaced and integrated in one neural network, enabling the system to parse all the components of the image, simultaneously. One of the first Convolutional Neural Network (CNN)-based methods relating to architectural drawing analysis was introduced in [2]. The authors first extracted low-level geometric and semantic information from floor

plans (junctions, rooms and symbols) via a neural network architecture, and then used Integer Programming (IP) to aggregate junctions into primitives for extracting walls. They provided an annotated dataset of 870 real-world images based on the LIFULL HOME dataset[2]. This is a good start in terms of the availability of a realistic training dataset, but insufficient to address most real-world challenges, as images in the LIFULL HOME dataset only include limited architectural drawings of low complexity (with easy-to-segment notations, no overlapping graphics, and no occlusion), from small Japanese residential units. The authors of [3] improved upon the proposed network of [2] and introduced a larger dataset of 5000 images, CubiCasa5K. This is a step in the right direction for realistic training datasets, as their images are more complex. Their end goal was quite different though: reconstruct the vectorized representation of the input images. The first two rows of Figure 2.5 showcase some of sample images from LIFULL HOME and CubiCasa5K. For comparison, some problematic images are shown in the lower row that we usually face in real-world scenarios. Moreover, there are still some heuristics and constraints involved in the aggregation phase of these methods, when extracting structural representations from the junction points. This will limit the scalability of the proposed systems when dealing with new images.

2.4 Symbol Spotting: Performance Evaluation

As mentioned, the symbol recognition and spotting contests of the GREC workshops have provided the research community with standard evaluation tools including ground truth data, evaluation metrics, and evaluation protocols for comparing the performance of different symbol recognition and spotting methods in architectural floor plan applications. The general framework of these three key tools was published in [90]. Only two datasets of architectural floor plans, developed specifically for the purpose of symbol spotting, are publicly available: SESYD [14] and FPLAN-POLY [69]. In FPLAN-POLY, the floor plans are

[2]https://www.nii.ac.jp/dsc/idr/en/lifull

provided as vectorized graphic documents, which differs from our focus on document images. The SESYD dataset, related to the GREC symbol recognition and spotting contests, remains the most popular image dataset for evaluating symbol recognition and spotting algorithms.

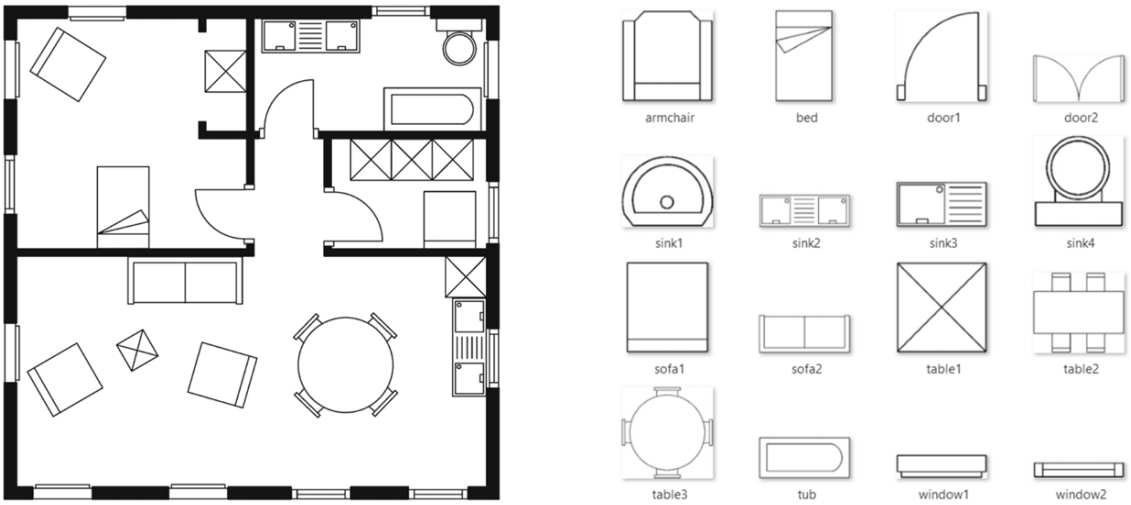


Figure 2.6: Sample synthetic architectural floor plan (a) and all architectural symbol models (b) from the SESYD dataset.

Figure 2.6 shows a typical floor plan, along with all 16 architectural symbol models, from SESYD. An overview of the use of SESYD can be found in [91]. As SESYD is a synthetic dataset, its creators proposed a generation tool to build synthetic documents that sets several constraints to determine various positions and locations of a given vocabulary in different ways over the same set of architectural backgrounds (building structure). Reference [92] proposed different evaluation metrics that should also be considered, introduced in terms of recognition abilities, location accuracy, and scalability, as well as a tool for manually annotating the location of symbols and their labels. In [93], a semi-automatic framework was proposed for ground-truthing real-world floor plan images. A top-down matching algorithm was used to minimize user interaction for defining the boundary of RoIs. The output of the tool includes graphics primitives composing the symbols as well as

the location and class of symbols. Finally, [94, 92] introduced characterization metrics that take into account not only the retrieval ability of symbol spotting methods in real images, but also their localization strength.

Table 2.2 summarizes the results of symbol spotting algorithms that have been evaluated using architectural floor plan images from SESYD. In this table, the evaluation metrics are precision (P), recall (R), F-score (F), the average precision (AveP), and the retrieval time of each symbol in each document (T). A detailed definition of all the parameters can be found in [92].

Table 2.2: Performance evaluation of symbol spotting approaches on SESYD.

Method	P (%)	R (%)	F	AveP (%)	T (sec.)	Subset
Nguyen *et al*. [33]	70.00	88.00	79.50	-	-	6 queries in 15 images from SESYD
Broelemann *et al*. [40]	75.17	93.17	83.21	-	-	6 queries in SESYD (floorplan16-01, floorplan16-05, floorplan16-06)
Bodic *et al*. [42]	90.00	81.00	85.30	-	-	16 queries in 200 architectural plans from SESYD
Nayef and Breuel [73]	98.90	98.10	98.50	-	-	12 queries in 20 architectural plan from SESYD
Dutta *et al*. [44]	62.33	95.67	75.50	70.66	0.57	SESYD (floorplans16-01)
Dutta *et al*. [81]	56.92	83.96	67.85	60.87	0.07	SESYD
Dutta *et al*. [83]	50.32	83.06	62.67	60.87	0.07	SESYD

It can be seen from Table 2.2 that, while SESYD is the most standard performance evaluation dataset for symbol spotting, only a handful of authors in the literature used it for evaluation purposes. Moreover, different subsets are typically used (see last column), which makes it difficult to provide fair comparisons between the spotting methods. Also, one can notice that the precision is typically fairly low. False positives thus remain an open

issue. In the next section, we expand on the remaining problems in the field.

2.5 Takeaways from Literature Review

Our literature review attempts to include the most important trends in the field of symbol spotting as they pertain to the application of architectural floor plan analysis. We mentioned some of the most important challenges, such as the lack of *a priori* information about the shape of symbol models, the presence of noise, clutter, occlusion, and the variability of notations for a given class of models. Other problems also arise when working with real-world images found in industry. The most popular dataset of document images for symbol spotting, SESYD, contains a set of synthetic images that are far from reflecting all the difficulties of real-world images. Thus, issues like occlusion, boundary discontinuities, distortions, and symbol irregularities, which are linked to real-world images, cannot be fully tested with SESYD. Since these problems are the main open challenges in real-world images used in industry, a considerable number of papers in the literature have tried to address these problems. For example, graphs are the main way to address irregularities and distortions of the images because of their flexibility in reflecting the structural relation between geometric primitives (although they are sensitive to noise that can throw off the vectorization process), despite a high computational complexity for the graph matching step. Although some authors have tested their approaches on their own datasets of real-world images, there is not a unique framework in the literature that would allow for fair comparisons of the ability of different methods to address the real-world architectural floor plan problems.

In the definition of a symbol spotting problem, it is generally assumed that the model image is a piece of input document which is either cropped by the user or is already available in a library. Symbol instances, embedded in a document, do look like the query symbol

from a topological viewpoint. However, when considering real-world floor plans actually used in industry, we are confronted with a large variability in the graphical notation used for a given architectural symbol class, that can affect even the most basic features of a symbol, such as its topology. This difficulty of clustering different designs of architectural symbols was noted in [92] in the context of ground-truthing graphic documents. As an example, a kitchen sink can have as many different representations as there are architectural firms, and even more (Figure 2.7). The variability in graphical notation is an open problem which is almost impossible to solve with a symbol spotting method in its regular sense, i.e. without using query symbols for all potential graphical notations. Methods that do tolerate some inexact matching between a symbol model and a symbol instance in a floor plan cannot reliably solve the problem due to potential changes in topology and the unbound variability. This issue seems more suitable for training-based methods which, however, bring their own issues such as a lack of sufficient and reliable training sets of symbols.

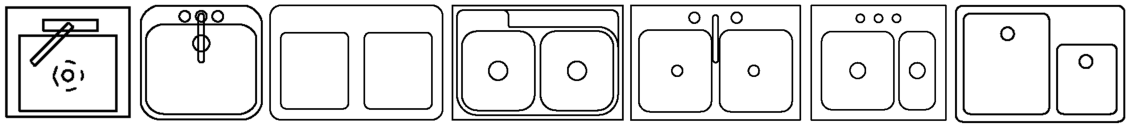


Figure 2.7: Graphical notation variability. Examples of graphical notation variability for kitchen sink symbols.

Deep learning techniques have recently enjoyed a spectacular success in detection, recognition, and classification tasks related to natural patterns [95, 96]. Such techniques (e.g. YOLO [97], R-CNNs [98]) are data hungry; thanks to large training datasets of natural images made publicly available by the computer vision community (e.g. ImageNet [99], MS-COCO [100]), these methods perform well for natural scenes. However, there is no equivalent of training datasets obtained from real-life architectural floor plans (see Section 2.3). Moreover, symbolic images such as floor plans have a high level of abstraction, conveying information via combinations of linear and curved segments, and lack the rich textural and color appearance of natural images.

Finally, the existence of very similar geometric substructures in architectural symbols constitutes another important issue which can mislead spotting algorithms and decrease their scalability. This issue makes the different classes of symbols less distinguishable and causes low precision when working on large datasets (see Table 2.2). In the next chapter, we propose a new system to solve some of the problems which are associated with symbol spotting methods.

Chapter 3

Geometry-based Symbol Spotting in Born-Digital Architectural Floor Plans

This chapter focuses on symbol spotting in born-digital architectural plans with a variety of graphical annotations for furniture, fixtures, appliances, etc. The main requirement for symbol spotting relates to performing "on-the-fly" queries [9], which precludes us from using learning-based methods, including deep learning. While some recent neural network-based methods relax the on-the-fly requirement [20, 87, 101] and reframe symbol spotting as an object detection problem, this chapter focuses on traditional symbol spotting, described in detail in Chapter 2, where the main objective is to retrieve all the image patches which contain the "user-selected" symbol.

We propose a symbol spotting method here that reaps the benefits of both pixel-based and vector-based approaches. A coarse localization of candidates using vector-based techniques is obtained first, then refined in a pixel-based manner. Our contributions are two-fold (theoretical and practical):

- From a theoretical viewpoint:

  (i) We propose a novel vector-based geometric descriptor that encodes the salient

geometric primitives of a symbol (i.e. linear, circular, and elliptical). We show that for digital-born architectural floor plans, these geometric shapes are the main constituents of the symbols and thus, the proposed descriptor can characterize the symbols very well, even in real-world scenarios where occlusion and noise pose challenges.

(ii) We propose a new voting-based algorithm to assist the descriptor in finding partial ellipses. This new ellipse detection approach is able to detect partial ellipses for which at least one endpoint from each axis (major and minor) is part of the image area.

(iii) We also introduce a novel primitive-aware proximity graph that encodes the spatial relationship between the extracted primitives and allows for a simpler matching paradigm based on statistical clustering (instead of structural pattern recognition).

- From a practical viewpoint:

  (i) We embed the descriptor and the matching approach into an end-to-end system addressing the challenges of symbol spotting in architectural floor plans.

  (ii) We demonstrate that our end-to-end system works particularly well on real-world images that contain occlusions and overlapping graphical notations, making the system a good candidate in these scenarios.

The remainder of the chapter is structured as follows: Section 3.1 details our approach to symbol spotting, Section 3.2 discusses the experimental results, and Section 3.3 highlights some computational considerations. Finally, Section 3.4 presents the concluding remarks.

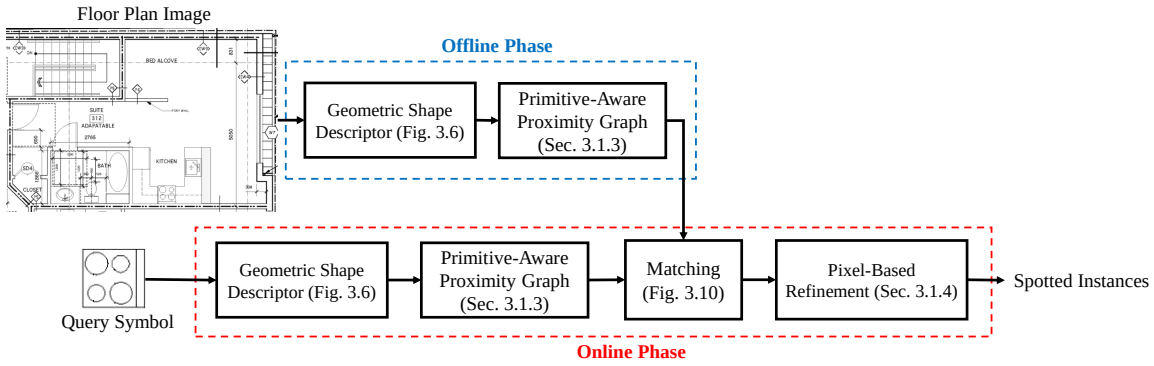


Figure 3.1: General block diagram of the proposed symbol spotting system.

3.1 Proposed Approach

We propose a new symbol spotting method for architectural floor plans that combines vector- and pixel-based techniques. The general block diagram of the entire system is shown in Figure 3.1. We first encode the salient geometric properties of symbols via a novel vector-based descriptor. Figure 3.6 shows the general block diagram of the geometric shape description process, which is further explained in sections 3.1.1 and 3.1.2. Using the vector-based description, we then construct a primitive-aware proximity graph that captures the spatial relationships between the constituents of a symbol, which is then matched, via clustering, to regions within a floor plan that have a similar graph. This step yields a coarse symbol localization and is detailed in Section 3.1.3. Finally, a pixel-based refinement phase fine-tunes the localization via template matching using a modified cross-correlation function, explained in Section 3.1.4.

3.1.1 Preprocessing

The presence of lines with different thicknesses might result in multiple detections of the same primitive due to the voting nature of the algorithm. This issue is solved with a preprocessing step, which consists in binarization and thinning. Binarization is carried out via a simple thresholding (Otsu's method [102]), and thinning is done via the method pro-

posed in [103]. Then, one-pixel thick linear structures of binary architectural floor plans are computed. To correctly detect primitives of symbols adjacent to walls (Figure 3.2a), thick walls are replaced by their boundaries (Figure 3.2b) before thinning, using morphological operations (the original image is subtracted from a dilated version obtained with a 10×10 structuring element). Although the line thickness may provide information about certain primitives such as interior/exterior walls, this is typically not the case for symbols, as the geometry of the symbols' constituent primitives is of the most importance. A similar reasoning applies to color information. Thus, the binarization and thinning processes will not negatively affect the retrieval performance. An added bonus of the preprocessing is that it makes the system more robust to differences in resolution which result in differences in line thickness.

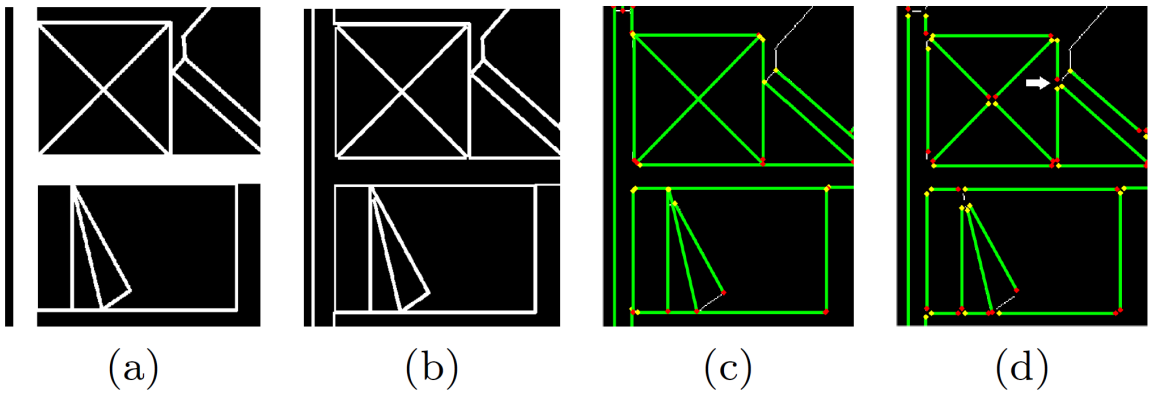


Figure 3.2: Preprocessing: (a) fragment of input image containing thick walls, a desk (top left), a bed (bottom) and an incomplete piece of furniture (top right); (b) replacing thick walls by their boundaries; (c) detected lines (green, with endpoints in yellow and red) before removing corners; (d) results after corner removal (white arrow refers to an undesired boundary fragmentation). Images are inverted for better visualization.

When the symbols are adjacent to walls or other symbols, their primitives may be merged with the other primitives in their neighborhood. This is solved in a second preprocessing step, which consists in corner removal; its rationale is best explained using the example in Figure 3.2c. Both the desk (top left) and bed (bottom) symbols share boundaries with walls, which complicates the extraction of their constituent primitives. This problem

is addressed by removing all intersection points with a corner removal algorithm. We opted to use Curvature Scale-Space (CSS) methods since they are well-suited for binary contour images. We chose to work with the method proposed in [104] due to its robustness to local variation and noise. This method estimates the discrete curvature via chord-to-point distance accumulation.

Figures 3.2c and 3.2d show the detected lines (see Section 3.1.2) before and after removing all corners from Figure 3.2b, respectively. Figure 3.2d shows that removing corners leads to a correct retrieval of all boundaries of symbols which are adjacent to walls. It is important to keep a copy of the original image as well for the geometric shape extraction, as sometimes the corner removal of overlapping symbols may split a symbol's boundary. For instance, the touching point of the two upper symbols in Figure 3.2d (see white arrow) is removed by CSS, which fragments the right vertical boundary of the desk symbol in two pieces, which in turn yields two incorrect constituent primitives. After extracting all the primitives from both the original image and the one without corners, the final description is obtained from the intersection of both sets.

3.1.2 Geometric Shape Description

Our motivation for describing the images by their salient geometric shapes comes from the study of architectural symbols in residential plans, which are all generated using combinations of simple geometric shapes (see Figure 3.3). More specifically, we are interested in finding primitives such as line segments as well as complete and partial circles and ellipses composing the symbols. These are the most frequent shapes that can be found in regular digital-born architectural images. To find these main geometric shapes, we employ voting-based approaches. The rationale is that they make the descriptor robust to noise and occlusion, as long as the main primitives of the symbols are dominant within their neighborhood.

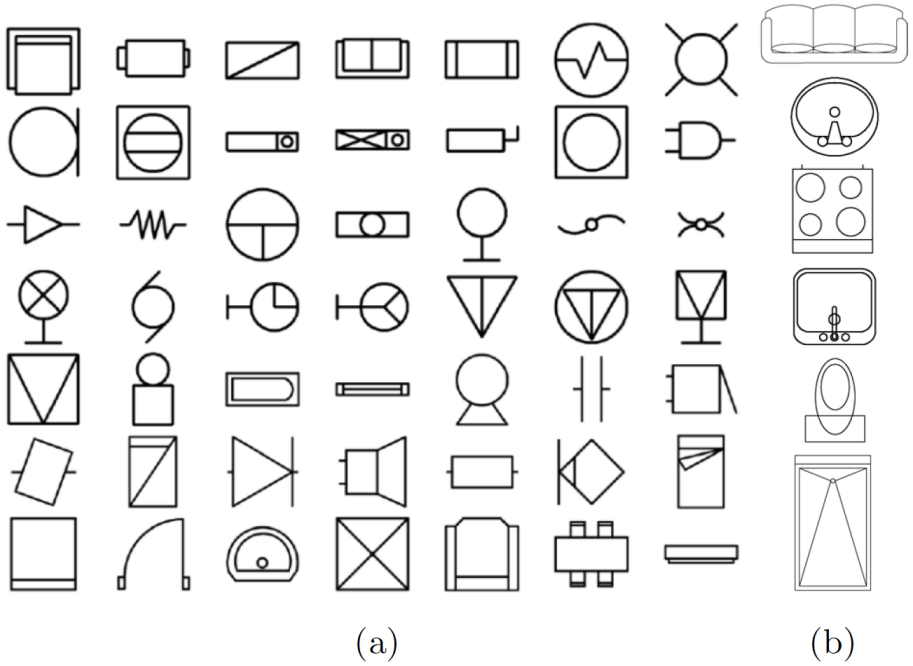
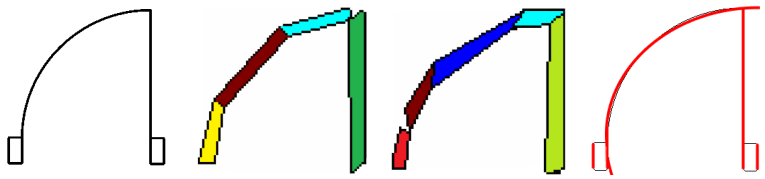


Figure 3.3: Examples of synthetic symbols from the public dataset SESYD (a), and of real-world architectural symbols (b).

The proposed descriptor does not have any inherent limitation in terms of what symbols it can describe, as some other descriptors do. For instance, quadrilateral-based representations [35, 76] are imprecise in representing curves (Figure 3.4). Figure 3.4a shows a door symbol with a quarter circle as its main primitive, Figures 3.4b-c (adapted from Ref. [76]) show the quadrilateral-based representation of two door instances within the same floor plan that are not affected by any important occlusion or noise, and Figure 3.4d shows the representation by the proposed descriptor for reference purposes. As it can be seen from Figure 3.4b-c, the quadrilateral-based descriptor yields two different descriptions of the same symbol, which shows its instability in dealing with curves. Since curves are frequent constituent primitives of architectural symbols (see Figure 3.12 for several examples), this limitation can highly degrade the detection performance of any spotting algorithm utilizing such a descriptor. A second example of inherent descriptive limitation can be found in the attributed relational graph of [38], which can only describe symbols that contain at least two



(a) (b) (c) (d)

Figure 3.4: Inherent limitations of some descriptors: door symbol (a), two different quadrilateral-based descriptions of door instances in the same document (b-c) which can yield to degraded detection performance, and singular description with the proposed geometric descriptor (d).

different types of visual primitives (thick components, circles, corners and extremities) in their structure. This is not necessarily the case for architectural symbols, as many of them include only one type of these four primitive, e.g. corners only (see armchair, window1, window2, table2, bed, sofa2, sofa1 and table1 in Figure 3.12). A third example of inherent descriptive limitation comes from closed region-based descriptors [101, 41, 43, 78, 69, 42], which have a hard time with symbols that are not entirely composed of closed regions such as the door symbol of Figure 3.4a.

In addition to being generic, the proposed geometrical descriptor is more stable than many other descriptors in the presence of occlusion and noise. Noise can easily mislead critical-point based descriptors such as those based on equidistant points [1], polygonal approximation [83], or edge-based primitives; similarly, occlusions and overlaps can drastically alter the output of quadrilateral primitives or region-based descriptors such as closed or convex regions [44]. For example, Figure 3.5 illustrates the failure of closed region-based descriptors in the presence of occlusion. Figure 3.5a shows an ideal bathtub symbol composed of two closed regions, as described in Figure 3.5b by a closed region-based descriptor with two different colors. Figure 3.5d is an instance of the same bathtub symbol as in Figure 3.5a but as part of a real-world scenario; the presence of occlusions yields a completely different closed region-based description (Figure 3.5e), compared to the non-occluded case (Figure 3.5b). The proposed descriptor was successful in extracting the

salient primitives of the bathtub symbol (ellipse and lines from the rectangle) in both the ideal (Figure 3.5c) and the occluded (Figure 3.5f) cases. Occlusions are very common in real-world images, due to other graphical information in the images such as measurements, annotations, revisions, etc., (see figures 3.14e and 3.14f in Section 3.2 for some visual examples). Closed region-based descriptors can significantly decrease the detection performance of a system in real-world scenarios due to incorrectly detected constituent primitives of interest, whereas the proposed method is able to retrieve those primitives even in the presence of various levels of occlusion.

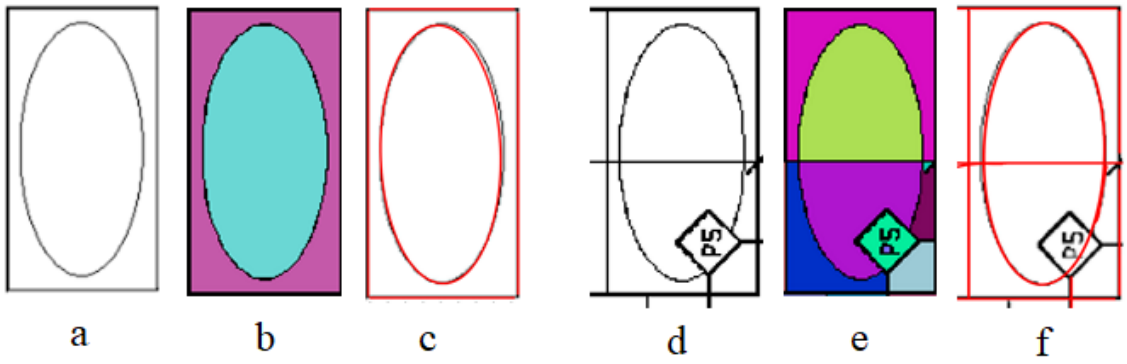


Figure 3.5: Occlusion-related limitations of some descriptors. A bathtub symbol (a) and its occluded version in a real-world floor plan (d), corresponding descriptions by closed region-based descriptors (b,e) and by the proposed method (c,f).

For a quantitative insight into how restrictive those inherent and occlusion-related limitations are, we invite the reader to consult Figure 3.14e in Section 3.2. Considering closed region-based descriptors, out of a total of nine symbols of interest, three doors would be missed as they do not include any closed region, and the main constituents of five other symbols (bathtub, toilet, fridge, range and kitchen sink) would be significantly altered as a result of light occlusion, all correctly detected by the proposed method.

For describing the input image, primitives are retrieved through a sequential strategy shown in Figure 3.6 to limit the computational load. In this pipeline, after applying the aforementioned preprocessing steps, first, a simple Standard Hough Transform (SHT)

[105], which groups edge points into shape candidates by performing an explicit voting procedure, detects and removes line segments from the preprocessed image. Then, ellipses (and circles) whose major/minor axis endpoints are in the image area are found via our own variation of the Hough Transform proposed by Xie and Ji [106]. Lastly, we retrieve partial ellipses (and circles) via a novel method, described in Section 3.1.2. All the geometric shapes are saved in the format of ellipse parameters, i.e. $\{x_c, y_c, a, b, \alpha\}$ where (x_c, y_c) is the center point, a and b are the length of semi-major and semi-minor axes, respectively, and α is the shape's orientation. For circles, $a = b$ and α is not defined, and for line segments, $b = 0$. The "Corner Detection", "Line Segment Detection" and "Ellipse Detection" blocks in Figure 3.6 have two outputs: the parametric description of the detected primitives and the corresponding image output, obtained by removing the detected primitives from the block's input image. The image outputs are necessary for progressing through the computational pipeline, while the primitive parametric descriptions compose the proposed descriptor's output.

Line Segment Detection

The SHT detects all straight lines in both the input image and the image without corners, by using the information in the Hough transform space of the image [105]. To find the finite line segments from the peaks in the Hough space, first, all the pixels that are associated with a given peak are found, then, the connected components that are 2 pixels apart form the line segments. Finally, each line segment is valid only if its endpoints are adjacent to corner points (C_r in Figure 3.6); this ensures that false line detections are pruned. The parameters of valid line segments (L_1 and L_2 in Figure 3.6) are added to the feature descriptor. Pixels belonging to L_2 are removed from the input image ($I_{in} \backslash L_2$ in Figure 3.6) before the ellipse detection step. Figs. 3.7a and 3.7b show part of a floor plan from the public SESYD dataset and the corresponding detected lines in green, respectively.

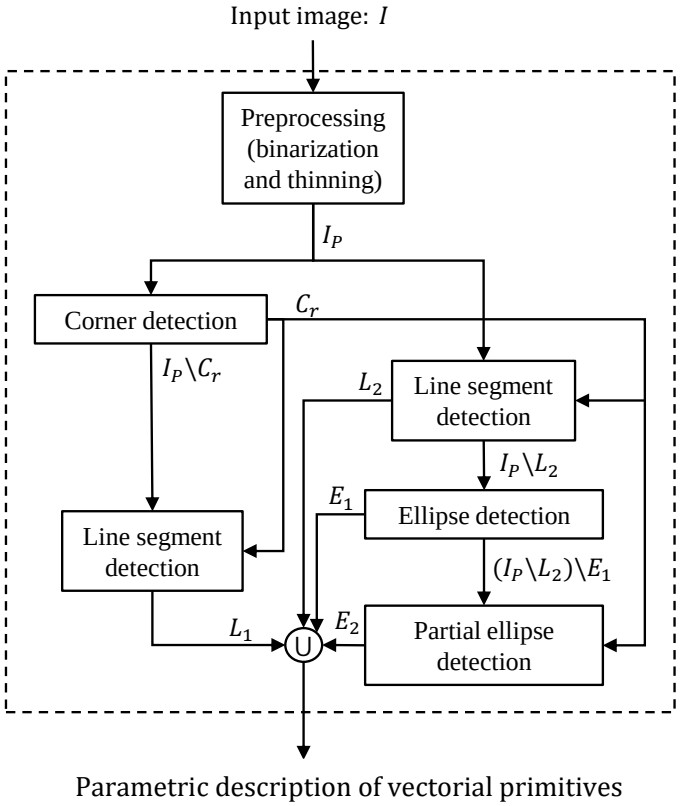


Figure 3.6: General block diagram of the proposed shape description. I and I_P are the original image and the image after the first preprocessing step, respectively. L_1 and L_2 show the sets of detected line segments in the output of the corresponding blocks (after and before corner removal) and E_1 and E_2 are sets of ellipses and partial ellipses, respectively. The preprocessed image I_P is passed through the "Corner Detection" block which outputs the corner points (C_r) and the preprocessed image without corner points ($I_P \backslash C_r$). C_r is used in Section 3.1.2 to decrease the search space when finding E_2. $I_P \backslash L_2$ is the preprocessed image after removing lines (L_2) and $(I_P \backslash L_2) \backslash E_1$ shows the image without lines (L_2) and ellipses (E_1).

Ellipse Detection

To detect ellipses in the image (and circles and remaining line segments at the output of the "Line Segment Detection" block), we use two approaches. The first one, inspired by the method proposed by Xie and Ji [106], detects all ellipses whose major/minor axis endpoints are available in the image area. The second method, described in Section 3.1.2, detects all remaining ellipses that have at least one endpoint of the minor axis and one endpoint of the

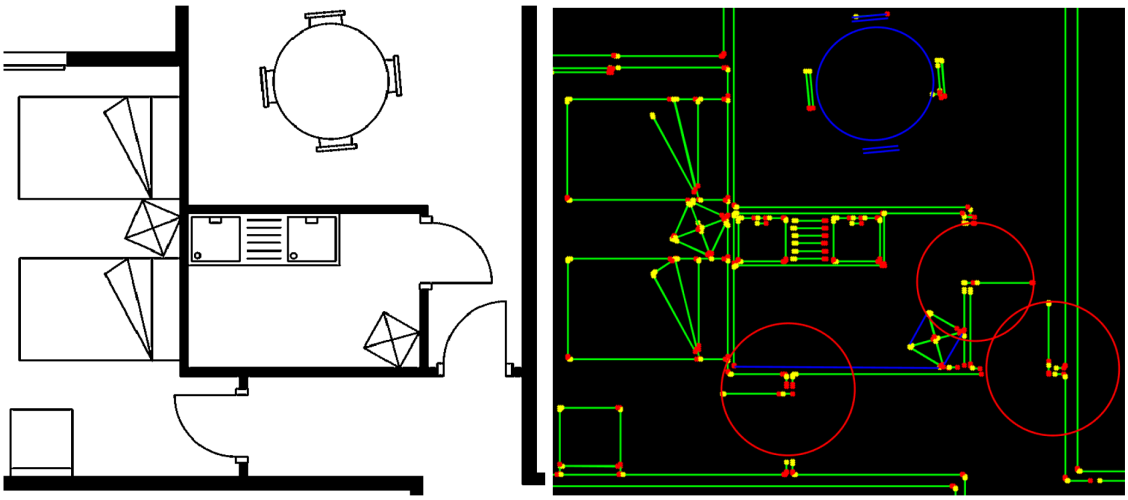


Figure 3.7: Part of a floor plan from the public dataset SESYD (a) and corresponding vectorial primitives (b), where $L_1 \cup L_2$ is shown in green (the endpoints of lines are represented by red and yellow dots), E_1 in blue and E_2 in red.

major axis available.

The original approach of Xie and Ji [106] estimates the five parameters of an ellipse by assuming the endpoints of the *major* axis as known. Figure 3.8a displays the geometric parameters of the ellipse and all notations that we will use hereafter. Each pair of points, (x_1, y_1) and (x_2, y_2), on the boundary of the object under evaluation are assumed as the potential endpoints of the major axis of an imaginary ellipse; this assumption yields four out of the five required parameters of an ellipse directly. Given (x_1, y_1) and (x_2, y_2), a third point (x, y) on the ellipse is enough to uniquely determine the 5^{th} parameter, i.e. the length of the semi-minor axis b. To find an ellipse, every pair of pixels on the boundary of the shape is tested as endpoints of the major axis and the rest of the points (which satisfy $d \leq a$) vote for the length of the minor axis. This way, the 5-dimensional voting space is reduced to a one-dimensional accumulator array for finding b. Similarly, one can consider every pair of pixels as endpoints of the minor axis and have the third point (x, y) vote for the length of the semi-major axis a. Since in some cases, one of the endpoints of one of the

axes might not be covered by the boundary, we first consider each pair of pixels as being potential endpoints of the major axis, then potential endpoints of the minor axis, and finally we choose the ellipse which passes through the most boundary points.

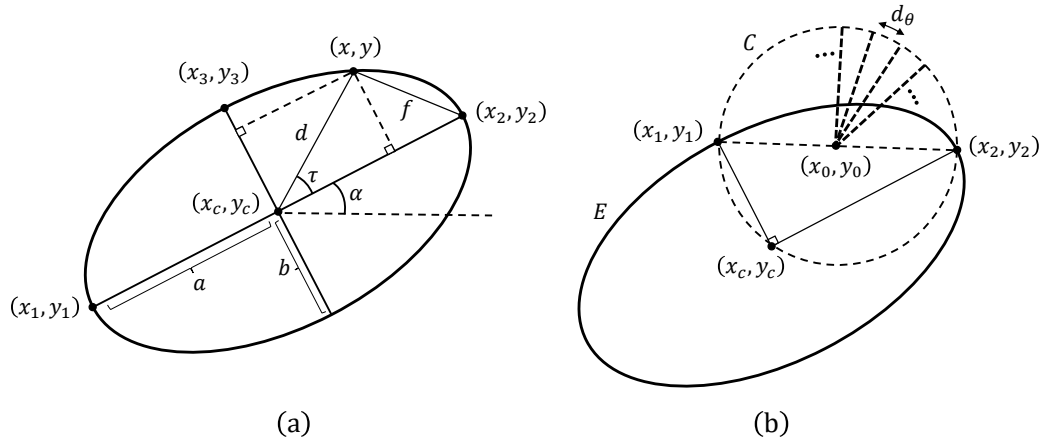


Figure 3.8: (a) Ellipse geometry; (b) a plausible ellipse, E, passing through (x_1, y_1) and (x_2, y_2) with center (x_c, y_c); C shows the foci of ellipse centers.

Since the number of ellipses (and circles) in an input image is typically way smaller than line segments, and since most of the line segments are already detected in the previous step, the computational load of the ellipse detector is of no concern here. Still, to further decrease the computational time, we do not run the ellipse detector directly on all the boundary points. If we were to do so, $\binom{k}{2}$ pairs of pixels, where k is the number of remaining points after line segment detection, would have to be considered as the potential endpoints of axes, while most of these pairs do not belong to an individual shape. Instead, all connected components that are less than 5 pixels away are merged into one segment. Then, for each segment, the ellipse detector is applied on the polygonal approximation of the boundary points of that segment to find the best geometric shapes, if there is any. The 5-pixel value was determined empirically; if two segments are more than 5 pixels away, they are assumed to belong to two different elements of the image.

Figure 3.7b shows detected ellipses in blue from the sample floor plan of Figure 3.7a. This may also include line segments that are missed by the "Line Segment Detection block"

(Section 3.1.2), since line segments are ellipses with $b = 0$.

Partial Ellipse Detection

The method described in Sec. 3.1.2 is only able to detect partial ellipses (and circles) when their major/minor axes endpoints are part of the image. This is a critical limitation for architectural plan analysis, as quarter ellipses compose many popular symbols (such as doors, see Figure 3.3).

We propose a new voting-based ellipse detector that is able to retrieve any partial ellipse (and circle) composed of at least one quarter of a full ellipse (i.e. containing at least one endpoint of each of the major and minor axes). For the sake of simplicity, we refer to these kinds of curves as partial ellipses here. We consider each pair of boundary pixels as one of the endpoints of the major and minor axes, such as (x_1, y_1) and (x_2, y_2) in Figure 3.8b. Since the major and minor axes are perpendicular, it can be easily shown that the foci of the ellipse center lies on a circle described by:

$$(x_c - x_0)^2 + (y_c - y_0)^2 = r^2, \tag{3.1}$$

where (x_0, y_0) and r are the midpoint and half of the Euclidean distance between (x_1, y_1) and (x_2, y_2), respectively.

Depending on the desired accuracy, a limited set of center points in Eq. (3.1) is selected with angular resolution d_θ (we use $d_\theta = 2$ degrees which gives 180 possible centers). Then, all boundary pixels vote for each selected ellipse center. To reduce the algorithmic complexity, polygonal approximations of the boundaries with resolution P_{res} can be used when testing all possible pairs of (x_1, y_1) and (x_2, y_2). P_{res} shows the minimum distance between successive points on the boundary. Since door symbols are essential and arguably one of the most important symbols in architectural floor plan analysis, we focus on the corner points (C_r in Figure 3.6), which are enough to extract partial ellipses in that case (in the

general case, all potential pairs of points should be tested, although at the expense of an increased computational time).

Algorithm 1 gives the detailed steps of the proposed partial ellipse detector. The computational complexity of this approach is higher than Xie and Ji's approach [106] and depends on the angular resolution of the center points on C (Figure 3.8b). Thus, it is applied only to the image from which line segments and ellipses have been removed (see $(I_p \backslash L_2) \backslash E_1$ in Figure 3.6). In Figure 3.7b, detected partial ellipses from the sample floor plan of Figure 3.7a are plotted in red. One may note that the entire description process is performed offline, as shown in Figure 3.1, and needs to be done only once for the entire input dataset. The computational complexity is therefore less important than the description accuracy.

3.1.3 Matching Via Primitive-Aware Proximity Graph

The geometric shape extraction methods described in the previous section provide a parametric representation of all primitives detected in the query symbol and in the image. In order to capture the intrinsic structure of a given query symbol, we also need to consider spatial relationships between its constituent primitives. Therefore, we propose a novel primitive-aware proximity graph (V, E) for representing both the query symbol and the image. The set of nodes V consists of the center points of all the geometric primitives (i.e. their parametric representation). The set of edges E encodes pairwise spatial relationships between the nodes that they link. Each edge $e \in E$ has three attributes:

- the Euclidean distance between the centers of the primitives that are linked;
- the edge orientation (angle θ_c in Figure 3.9) with respect to the horizontal axis;
- the connection type (six possible values encoding line-line, line-circle etc., shown in Figure 3.9).

Algorithm 1: Partial Ellipse Detector

Input: Binary image bw
Parameters: Angle resolution d_θ
Minimum curve length ratio r_l
Polygonal approximation resolution P_{res}
Output: $Primitive_{list}$: List of detected ellipses and circles
Begin:
$S \longleftarrow$ Segments of bw
while $S \neq \emptyset$ **do**
  for $Cmp \in S$ **do**
    $Cmp_{apr} \longleftarrow$ Polygonal approximation of Cmp with resolution P_{res}
    $CurveLength = 0$
    $Curve_{opt} = \emptyset$
    for *all* (p_1, p_2) *in* Cmp_{apr} **do**
      $P_0 \longleftarrow$ all center points from Eq. (3.1) with resolution d_θ
      for $p_0 \in P_0$ **do**
        $Curve \longleftarrow$ ellipse passing through p_1 and p_2 with center p_0
        $PixList \longleftarrow$ Pixels in bw belonging to $Curve$
        if $perimeter\ (PixList)/perimeter\ (Curve) > r_l$ **and**
         $perimeter\ (PixList) > CurveLength$ **then**
          $Curve_{opt} \longleftarrow Curve$
          $PixList_{opt} \longleftarrow PixList$
          $CurveLength \longleftarrow perimeter\ (PixList)$
    if $Curve_{opt} \neq \emptyset$ **then**
      remove $PixList_{opt}$ from bw
      $Primitive_{list} \longleftarrow Primitive_{list} \cup Curve_{opt}$
      $S \longleftarrow$ Segments of bw

One should note that the third attribute, encoding the connection type, is what confers primitive awareness to the set of edges E and thus to the proposed proximity graph.

Our proposed matching approach is suitable for detecting all instances of the query symbol in the image, based on the assumption that such instances might have undergone similarity transformations (a combination of rotation, translation, and scale change). We do not use rotation and scale-invariant primitive descriptors such as [69], as these information are crucial when comparing different pairs of adjacent primitives. Our primitive-aware proximity graph is also richer than the one proposed in [69], in terms of relational informa-

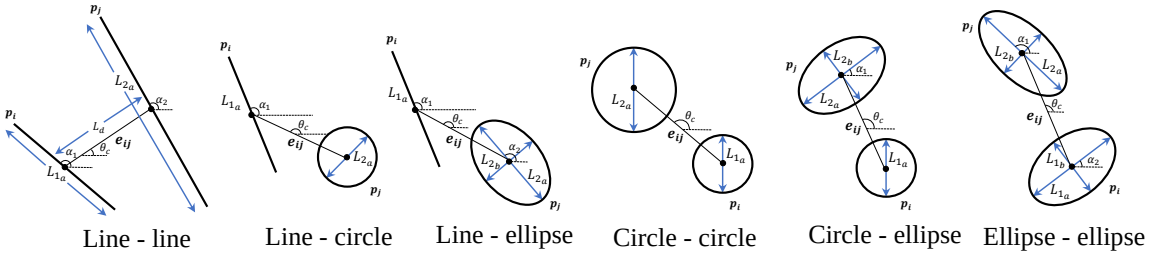


Figure 3.9: All six connection types for three different geometric shapes. p_i and p_j are two arbitrary primitives connected by edge e_{ij} in the primitive-aware proximity graph. θ_k ($k = 1, 2$) is the orientation of the k-th primitive connected to the edge w.r.t. the horizontal axis and θ_c is the orientation of e_{ij}. L_{k_a} and L_{k_b} are the lengths of the semi-major and semi-minor axes of the k-th primitive, respectively. For circles, θ_k is unknown and $L_{k_a} = L_{k_b}$. For lines, $L_{k_b} = 0$.

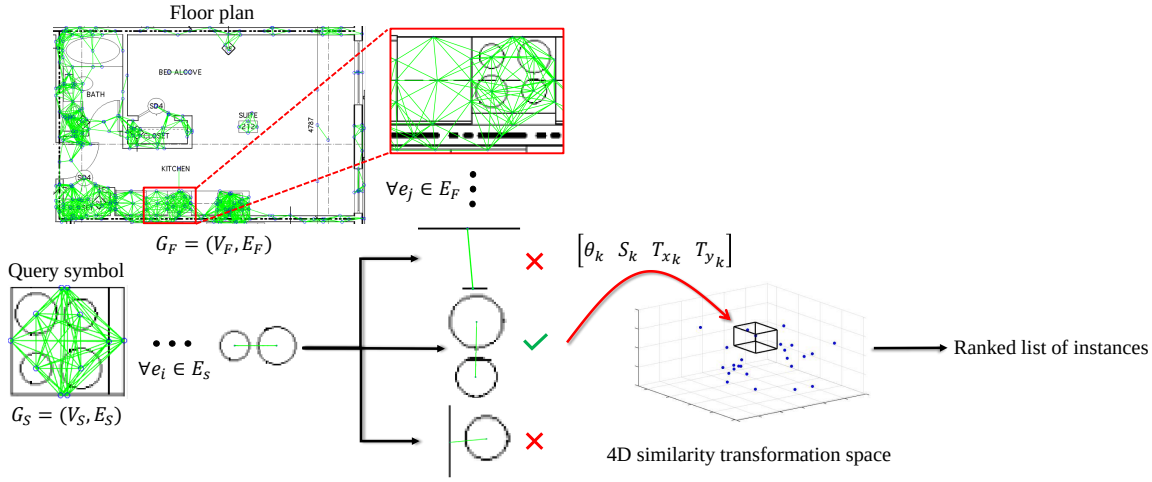


Figure 3.10: A graphical rendition of the pipeline of the spotting phase described in Section 3.1.3. G_F and G_S are primitive-aware proximity graphs of a floor plan and the query symbol, respectively, and $[\theta_k, S_k, T_{x_k}, T_{y_k}]$ is the $k-$th feasible spatial transformation vector. Each edge of G_S is compared with the edges in G_F and the ones that can be matched with a feasible transformation add a new point to the 4D similarity transformation space.

tion. Some variations of proximity graphs, e.g. [41, 42], attempt to encode the relational information of adjacent primitives such as the ratio between lengths, orientations, sizes, etc. Such works use computationally complex structural pattern recognition methods in the matching phase while here, we perform matching via a simple voting-based method described below.

The matching phase, illustrated in Figure 3.10, compares each edge in the primitive-

aware proximity graph corresponding to the query symbol to all the edges in the image's primitive-aware proximity graph. This comparison allows for computing the spatial transformation that maps the pair of primitives linked by the edge to a corresponding pair in the image. All allowed spatial transformations are defined as a combination of rotation, scale change, and translation, and are therefore encoded by four parameters $[\theta, S, T_x, T_y]$, where θ describes the rotation, S is the ratio between the size of the primitives in the image and the ones in the symbol, and T_x and T_y define horizontal and vertical translations. We retain only feasible transformations where the angles and lengths of primitives and their relational positions in the query symbols are correctly transformed. For instance, in Figure 3.11, let (p_i, p_j) and $(p_i^{'}, p_j^{'})$ show the primitive pairs in the query symbol and the image, respectively. Based on the spatial arrangement of primitives, Figure 3.11d is a feasible transformation of Figure 3.11a while Figs. 3.11b and 3.11c are not. This is because in Figure 3.11b, $\alpha_1^{'} - \alpha_1 = \theta_c^{'} - \theta_c \neq \alpha_2^{'} - \alpha_2$ and in Figure 3.11c, $L_d^{'}/L_d = L_{2_a}^{'}/L_{2_a} = L_{2_b}^{'}/L_{2_b} \neq L_{1_a}^{'}/L_{1_a}$ which makes it impossible to calculate the *angle* and *scale* parameters, respectively.

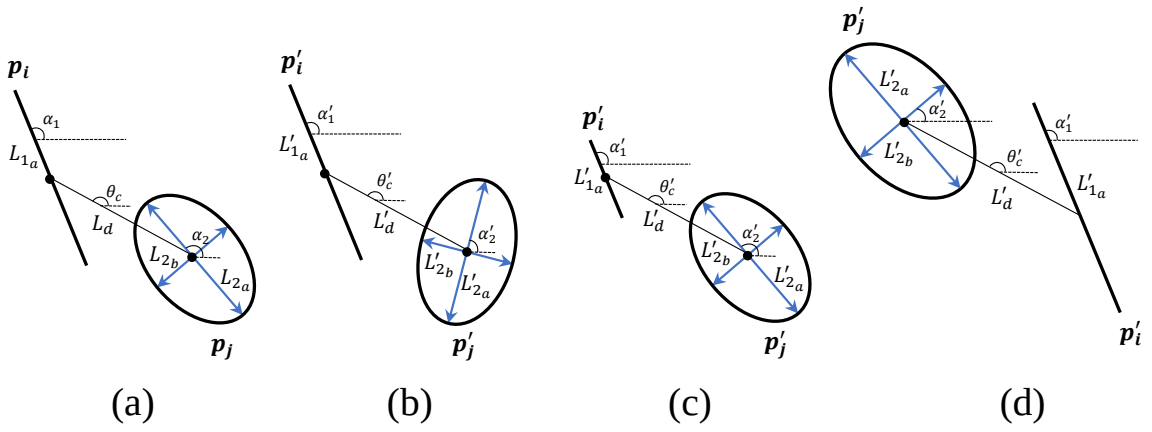


Figure 3.11: Some examples of feasible vs. non-feasible similarity transformations. A pair of primitives in the query symbol (a), two non-feasible transformations (b-c), and a feasible one (d) are shown.

All feasible similarity transformations that map the query's edges to the image edges can be considered as points in a 4D transformation space. The key idea is that all pairs

of primitives from a given query symbol are mapped by the same transformation to the image space. To find clusters of points in the transformation, we divide the 4D space into a grid with a resolution of 15 degrees for *angle*, 0.2 for *scale* and 50 pixels for both T_x and T_y. We then count the number of points in the feature space which fall into the same division of the grid. The divisions with the highest number of points are more likely to represent an actual instance of a symbol. The average of all the points in these divisions represent the similarity transformation that maps the query to its occurrence in the image. The acceptable number of points in each division, to consider the instance as legitimate, varies with the information content in the query symbol. Its default value is set as 10% of the number of edges in the query symbol's primitive-aware proximity graph. This choice maximizes recall at the expense of precision, which is improved with the step described in the next section.

3.1.4 Pixel-Based Matching Refinement

The output of Section 3.1.3 yields a set of similarity transformation vectors, each of them corresponding to the center of a detected grid division in the 4D transformation space. The image patches that are matched to the query symbol can then be easily found by applying the transformation parameters to the query. However, errors may occur due to the one-pixel thick nature of all primitives composing both the symbol and the image patch, and to irregular/noisy boundaries. To mitigate such errors, we propose a pixel-based template matching method based on a modified cross-correlation function for binary images, as follows:

$$\gamma(u,v) = \frac{\sum\limits_{x,y} I_p^d(x,y)\, q(x-u,y-v)}{\|q\|} + \frac{\sum\limits_{x,y} I_p(x,y)\, q^d(x-u,y-v)}{\|I_p\|}, \tag{3.2}$$

where $\gamma(u,v)$ is the cross-correlation coefficient at location (u,v), q and I_p are the query symbol and the image patch with one-pixel thick boundaries, q^d and I_p^d are the dilated images of q and I_p with a $n \times n$ structuring element (we use a 5×5 all-ones matrix), the $\|.\|$

operator computes the total number of 'ON' pixels of its operand, and the summation $\sum_{x,y}$ takes place over the entire image patch domain. The first component of the right term of Eq. (3.2) counts the number of foreground pixels in the query image that match the foreground pixels of the dilated image patch; the dilation is used to address slight variations in the primitives of the query. Likewise, the second component counts the number of foreground pixels in the image patch that match the foreground pixels of the dilated query symbol. Both terms are normalized to [0,1], thus $\gamma(u, v)$ results in a score in the range $[0, 2]$ with a score of 2 reflecting a perfect match.

Using Eq. (3.2), $\gamma' = \arg\max_{(u,v)} \gamma$ gives the best match and (u', v'), where $\gamma' = \gamma|_{(u',v')}$, is used to fine-tune the translation parameters T_x and T_y of the transformations found in Section 3.1.3.

3.2 Experimental Results and Discussion

To evaluate the performance of the proposed method, we selected the popular synthetic dataset SESYD. This allows us to compare our system with other related works. We also gathered a set of real-world images to show the efficiency of the method in real-world scenarios. The new dataset presents difficulties that do not exist in SESYD, such as overlapping elements, textual and graphical information, etc.

In the evaluation process, a first comparative evaluation is performed for the retrieval of individual symbols (i.e. instancewise), followed by a study of the accuracy of the detected bounding boxes (i.e. pixelwise). For the instancewise evaluation, we report the results using precision, recall, and F-score (P, R and F in Table 3.1, respectively) for individual symbols. For the pixelwise evaluation, we measure the precision, recall, and F-score (P_p, R_p and F_p in Table 3.2, respectively) in terms of retrieved and relevant pixels in a manner consistent with [92]. The pixelwise evaluation provides us with information on the localization ability

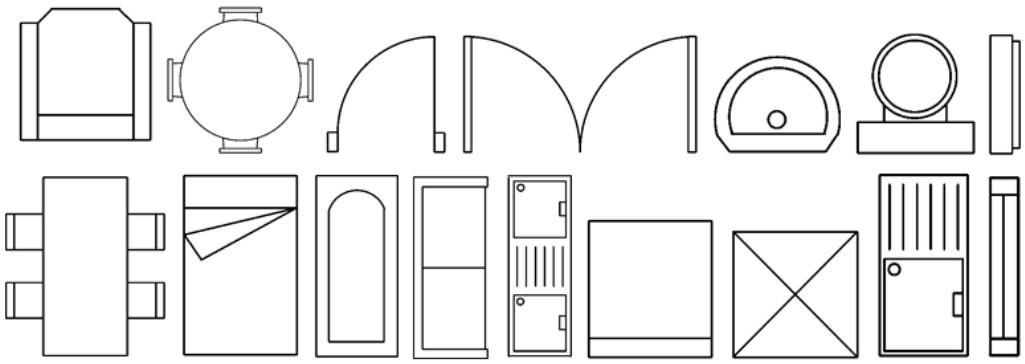


Figure 3.12: Query architectural symbols of the SESYD dataset. From left to right, first row: armchair, table3, door1, door2, sink1, sink4, and window1. From left to right, second row: table2, bed, tub, sofa2, sink2, sofa1, table1, sink3, and window2.

of the symbol spotting methods.

3.2.1 SESYD Dataset

Our experiments were performed on the entire architectural section of the SESYD dataset, proposed for the GREC Symbol Recognition and Spotting Contest [107]. The dataset consists of four subsets: a subset of ideal images and three noisy subsets with different degradations, named levels #1 to #3. Each of these four subsets possesses 20 floor plans, for a total of 633, 597, 561, and 593 symbol instances in each subset, respectively. The 16 query symbols of SESYD are shown in Figure 3.12. For evaluation purposes, we use all three degraded subsets of the contest, which were created by the contest's authors utilizing the binary degradation method proposed in [108]. The method applies random pixel inversion and blurring to model "light intensity fluctuations and point-spread function of the scanner optical system". The noise levels #1 to #3 in [107] simulate thinner and thicker lines than the original image lines, and add global noise to the image, respectively. Examples of noisy architectural floor plans from SESYD can be seen in Figure 3.14a-c.

Our proposed vector-based matching (as described in Section 3.1.1 - Section 3.1.3) needs to yield a high recall value, as missed symbol instances cannot be recovered in the

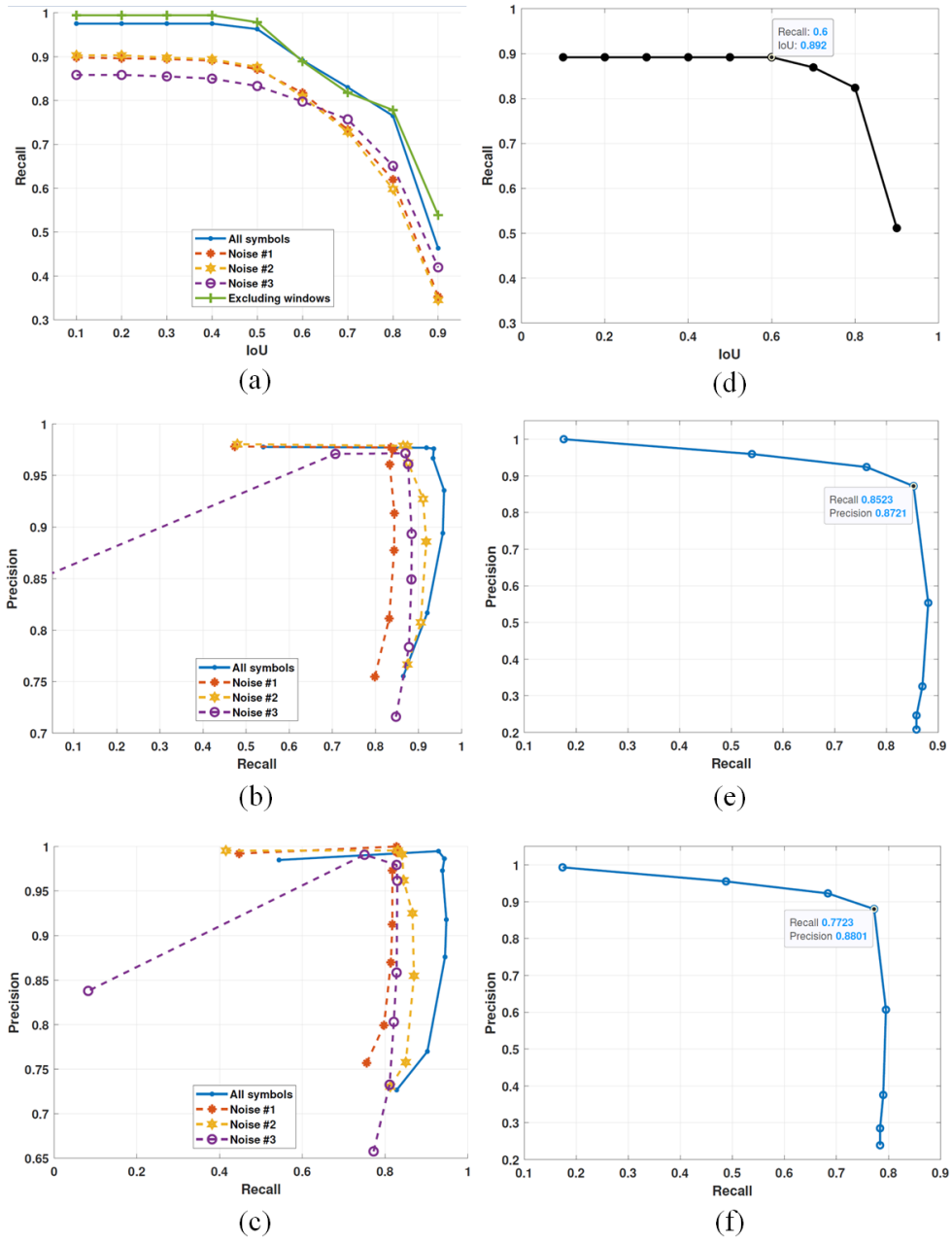


Figure 3.13: Performance evaluation of the proposed method on the SESYD original and noisy subsets (left column) and on the real-world images (right column). In (a,d), recall value as a function of IoU for the vector-based phase (without pixel-based refinement). In (b,e), precision value as a function of recall computed instancewise for the complete method (with pixel-based refinement). In (c,f), precision value as a function of recall computed pixelwise for the complete method (with pixel-based refinement). For (b,c,e,f), correlation threshold values are uniformly sampled from 1.3 to 2 with a 0.1 step. Datatips in (e) and (f) point to a threshold equal to 1.7 which gives the best results in terms of F-score on the real-world dataset (a threshold of 1.8 yields to the best results on SESYD).

pixel-based refinement phase (Section 3.1.4). Figure 3.13 (left column) illustrates the performance on the original images of SESYD and on the three noisy subsets. Figure 3.13a shows the recall value in the vector-based phase of the proposed method based on the Intersection over Union (IoU) between the correctly retrieved symbols and the ground truth. Clearly, higher IoUs in the graphs indicate larger overlaps between the detected symbols and the ground truth, and thus a better localization accuracy. One may note that IoU = 0.4 is the best trade-off in terms of detection overlap and recall value, yielding a recall of 97.52% which is almost as high as that corresponding to IoU = 0.1. It should be noted that a large number of the missed detections in this stage correspond to two query *window* symbols (Figure 3.12), out of a total of 16 query symbols. The *window* symbols are composed of a very limited number of line primitives, and thus have a very sparse description, which makes this kind of symbol hard to detect due to a lack of information. For this reason, several other works in the literature report their results on a limited number of query symbols of SESYD (see Table 3.1). Similarly, if we exclude the two *window* symbols, the recall value jumps up to 99.4% (see Figure 3.13a). The results in Figure 3.13a also show that the proposed vector-based matching works well under different noisy conditions, although there is some degradation in its performance.

The pixel-based refinement phase (Section 3.1.4) removes false positives and improves precision. Figure 3.13b shows the instancewise precision-recall curve for the original images and the three noisy subsets when different threshold values are used for the cross-correlation. Thresholds range between $[1.3, 2]$, with 2 corresponding to a perfect detection. The best performance in terms of F-score was obtained when the threshold was set to 1.8. For this threshold, the average IoUs between the detected symbols and the ground truth for the original images and the three noisy subsets (noise levels #1 to #3) are 93.00%, 92.94%, 94.46%, and 92.97%, respectively, which shows a great localization accuracy at the end of the symbol spotting pipeline.

Table 3.1: Instancewise evaluation of symbol spotting approaches on SESYD.

Method	P (%)	R (%)	F (%)	# Queries
Nguyen et al. [33]	70.00	88.00	79.50	6
Broelemann et al. [40]	75.17	93.17	83.21	6
Dutta et al. [44]	62.33	95.67	75.50	16
Le Bodic et al. [42]	90.00	81.00	85.30	16
J. Lerouge et al. [78]	91.00	82.00	85.00	11
Nayef and Breuel [1]	98.90	98.10	98.50	12
Proposed method (original dataset)	97.60	93.53	**95.52**	**16**
Proposed method (Noise #1)	97.51	84.07	90.29	16
Proposed method (Noise #2)	97.85	87.42	92.34	16
Proposed method (Noise #3)	97.16	86.93	91.76	16

Table 3.2: Pixelwise evaluation of symbol spotting approaches on SESYD.

Method	P_p (%)	R_p (%)	F_p (%)
Nayef and Breuel [1], as reported in Ref. [107]	62.00	99.00	76.00
Proposed method (original dataset)	98.63	94.31	**96.42**
Proposed method (Noise #1)	99.37	82.73	90.29
Proposed method (Noise #2)	99.13	84.11	91.00
Proposed method (Noise #3)	97.93	82.76	89.71

Table 3.1 compares our results, for a correlation threshold of 1.8, with other methods in the literature. It can be seen that the proposed method improves the current results on the SESYD dataset in terms of F-score when all the 16 symbols of the dataset are present. The results are also very close to [1], where the authors tested the instancewise performance of their spotting algorithm on only 12 query symbols. Moreover, the presence of noise does not affect much the results of our method, showing its robustness to distortions.

In the GREC Symbol Recognition and Spotting Contest [107], the performance of the winning algorithm [1] is reported in terms of P_p and R_p, to better quantify the localization power of the algorithms at the pixel level. Figure 3.13c shows the pixelwise precision-recall

Figure 3.14: Typical examples of results obtained when applying the proposed system on SESYD (top row), and on real-world (bottom row) images. (a-c) show sample images from noise levels 1-3, respectively. Red dotted bounding boxes show false detections.

curves of our system and the best results, in terms of F-score (correlation threshold of 1.8), are compared with [1] in Table 3.2. Based on Table 3.2, although the proposed method has a lower recall value, it significantly outperforms that of [1] in terms of F-score.

The first row in Figure 3.14 depicts the detected symbols on the noisy subsets (noise levels #1 to #3 from left to right).

3.2.2 Real-World Dataset

In this section we evaluate the efficacy of our system on real-world images. We used a set of architectural plans designed for multi-level buildings in the city of Victoria, BC, Canada, composed of 20 units with the same vocabulary of 7 symbols, shown in Figure 3.15. Table 3.3 provides detailed information about the number of instances for each symbol in this dataset.

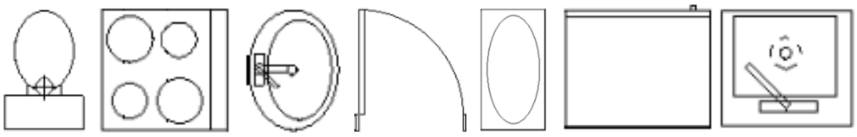


Figure 3.15: Query symbols in the real-world dataset. From left to right: toilet, range, bathroom sink, door, bathtub, refrigerator, and kitchen sink.

Similarly to Section 3.2.1, the evaluation is done both instancewise and pixelwise. The right column of Figure 3.13 illustrates the performance on the real-world dataset. Figure 3.13d shows the recall value in the vector-based phase of the proposed method based on the IoU between the correctly retrieved symbols and the ground truth. Figure 3.13e shows the instancewise precision-recall curve, while Figure 3.13f shows the pixelwise precision-recall curve. From Figure 3.13e-f, we can see that the best results, in terms of F-score, correspond to a correlation threshold of 1.7. The best threshold is thus comparable with that for SESYD dataset (1.8). As the results on SESYD for both threshold values of 1.7 and 1.8 are very close, we suggest using 1.7 as a general value that offers a good trade-off between the accuracy and retrieval powers of the system. Table 3.3 shows the number of detected instances and false positives for each individual symbol for both our proposed approach (correlation threshold of 1.7) and Nayef and Breuel's method [1] (using their own implementation[1]) on the real-world dataset. We can see from Table 3.3 that both methods yield the same number of false positives, while our method yields a significantly higher

[1] http://nayef.iupr.com/research/source-code

Table 3.3: Total number of instances for each symbol in the real-world dataset, along with the number of truly and falsely detected symbols by the proposed method and by a competing method [1].

Symbol	Quantity	Nayef and Breuel [1]		Proposed Method	
		True Positives	False Positives	True Positives	False Positives
Toilet	20	4	0	20	0
Range	21	5	0	20	0
Bathroom sink	20	3	0	13	0
Door	53	20	0	41	0
Bathtub	20	13	0	19	0
Refrigerator	21	5	20	19	22
Kitchen sink	21	10	2	17	0
Total	176	60	22	149	22

number of true positives. According to Figure 3.13, the best performance of our system was $P = 87.21\%$, $R = 85.23\%$, $P_p = 88.01\%$, $R_p = 77.23\%$. This compares to $P = 75.31\%$, $R = 34.66\%$, $P_p = 53.51\%$, $R_p = 40.06\%$ for Nayef and Breuel's method [1]. From these numbers, we can conclude that the proposed method significantly performs better on the real-world images, which are more challenging than SESYD images in terms of the presence of occlusion and the complexity of the textual and graphical information. Figure 3.16 illustrates how the proposed method is coping with some of these challenges. The bottom row of Figure 3.14 shows three sample images from the dataset and the resulting spotted symbols by the proposed system.

As shown in Table 3.3, all of the false positives in the proposed method come from the refrigerator ('fridge') symbol which possesses a limited number of simple primitives and can be easily mistaken for other parts of the image during the correlation-based refinement. The limited number of simple primitives is indeed a problem common to most symbol spotting methods and even problematic for human comprehension, due to the lack of enough distinct primitives in the query symbols. In terms of missed detections, doors and bathroom sinks are more problematic than other symbols for a similar reason. Even if

only one of the ellipses in the image cannot be found by the ellipse detection algorithms (due to a low resolution or heavy occlusion), then the remaining primitives are not enough for the symbol to be detected in the matching phase.

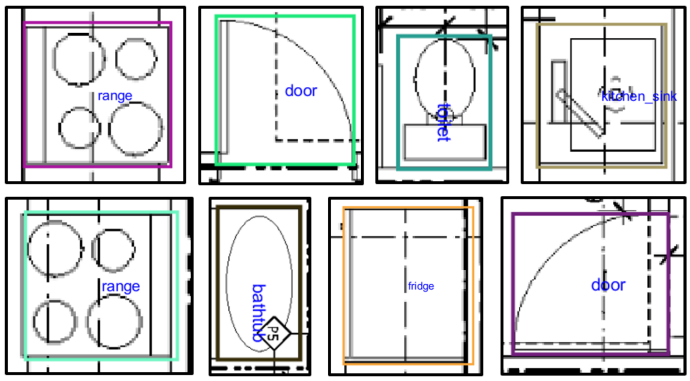


Figure 3.16: Typical examples of the performance of the proposed method in the presence of occlusion and overlap: all symbols are correctly detected.

3.3 Computational Considerations

Symbol spotting algorithms usually work in two stages, offline and online (Figure 3.1). In the offline stage, the input dataset of images is described by a set of descriptors. In the online stage, the query image is first analyzed using the same descriptors and its occurrences are then retrieved from the data structure obtained in the offline stage. Since the image dataset needs to be described only once, the computation load of the offline stage, which is the most time-consuming part of the algorithm (see Table 3.4), is not a concern. On the other hand, a fast retrieval process (online stage) is desirable. We implemented our system with Matlab R2020a on a desktop computer with a 3.4GHz Intel Core i7-6700 processor.

Table 3.4 states the processing time of the description phase (Section 3.1.1 and Section 3.1.2) for the SESYD and real-world datasets, along with the retrieval time in the matching and refinement phases (Section 3.1.3 and Section 3.1.4). Since the image resolution in SESYD is high, we resized all images to half of their original size, without losing any

critical information about the geometry of the primitives. The description is done offline and thus only once for each input image. The proposed description phase is highly parallelizable, as the neighboring connected components can be processed independently for finding ellipses. Thus, the numbers in the "Vector-Based Description" column of Table 3.4 could be significantly decreased with parallel computing (likewise for the "Pixel-Based Refinement" phase).

Table 3.4: Processing time of the different phases of the proposed method: per image for the vector-based description phase (done offline), and per symbol for the vector-based spotting and pixel-based refinement phases (done online).

Dataset	Average Image Size	Phase		
		Vector-Based Description (per image)	Vector-Based Spotting (per symbol)	Pixel-Based Refinement (per symbol)
SESYD (orig. images)	1317×1945	9.14 min	0.6702 sec	1.40 min
SESYD (noise #1)	1262×1645	21.67 min	0.5964 sec	1.15 min
SESYD (noise #2)	1415×1712	18.01 min	0.5180 sec	1.13 min
SESYD (noise #3)	1199×1605	8.40 min	0.6793 sec	0.51 min
Real-world images	735×624	0.82 min	0.1260 sec	0.50 min

3.4 Conclusion

This chapter presented a novel method for spotting architectural symbols in born-digital floor plans. Its first phase, a vector-based approach, provides a coarse localization of potential symbol instances; a geometric descriptor, which includes a new partial ellipse detection mechanism, encodes the parameters of the primitives composing the symbols, which are then matched to those in architectural drawings using a new primitive-aware proximity graph and a voting-based approach. The second phase, a pixel-based approach, provides a refined detection of symbols using a modified cross-correlation function. Experimental results show that our proposed method outperforms other symbol spotting methods on the

public SESYD dataset, used in the GREC Symbol Recognition and Spotting contests. Our method is robust to noise as shown via the binary degradation models in SESYD. To test our method on real-world challenges, we also evaluated it on a set of real-world floor plans and showed its good performance in presence of overlapping elements and occlusions. Our method does not require any training data and is thus compatible with on-the-fly querying tasks.

In the next chapter we shift our attention to a DL-based based approach, to detect symbols with different graphical designs.

Chapter 4

Symbol Spotting on Digital Architectural Floor Plans Using a Deep Learning-based Framework

Symbol spotting refers to the retrieval of graphical symbols embedded in larger images, typically in the form of a ranked list of regions of interest more likely to contain the symbols. Unlike symbol recognition, which aims to automatically label an already isolated symbol, spotting happens in context. It is typically carried out on-the-fly; no prior information about the shape of the symbols is known, and therefore machine learning-based methods are not helpful. This limitation can be in fact construed as a positive, as it eliminates the need for a training set. Annotated real-world datasets can be very difficult to obtain and few are publicly available; this is especially true for architectural floor plans, due to the intellectual property often restricting their use and publication, and to their sheer complexity and density of embedded information, which makes the annotation process a daunting task. On-the-fly symbol spotting circumvents the training process from annotated real-world datasets via an interactive process: the user crops a query patch in the image and

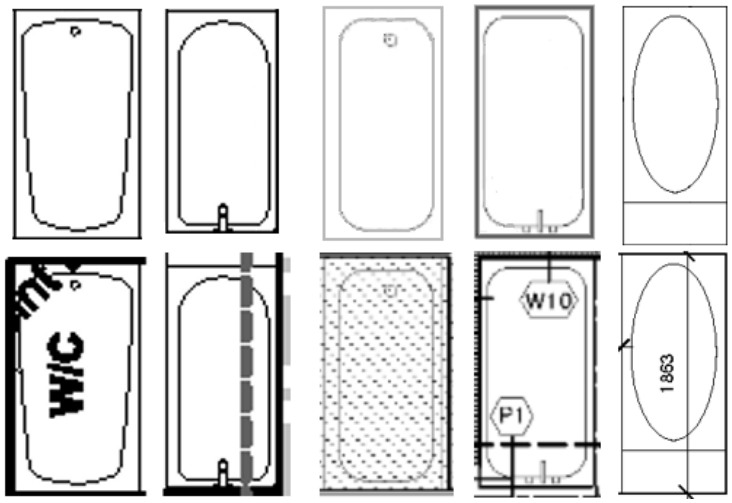


Figure 4.1: First row: 5 different graphical notations of the bathtub symbol. Second row: corresponding symbol instances in real-world scenarios with occlusions, clutter and various levels of degradation.

the system finds all similar patches within the image based on the statistical and geometrical information of the query patch. It is assumed that the user-identified patch contains a symbol.

One crucial drawback of on-the-fly symbol spotting is that it cannot cope with graphical notation variability. Being able to deal with such variability is very important in the context of designing a scalable method which is applicable to various semantically equivalent graphical representations. This is particularly true for architectural floor plans, as there can be so many graphical notations for a given symbol. Figure 4.1 (top row) illustrates some of the graphical notation variability for the bathtub symbol. In this chapter, we relax the "on-the-fly" property of traditional symbol spotting and instead tackle this semantic challenge by proposing a deep learning-based method that is scalable to various semantically equivalent graphical representations.

Another important consideration in architectural plans is the presence of various levels of occlusion and clutter. In real-world plans, the quantity of information that has to be conveyed by architects for the proper construction or renovation of the building is significant,

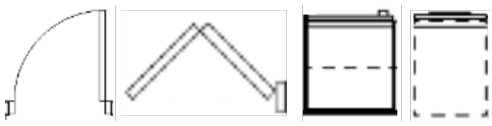


Figure 4.2: Some examples of trivial symbols consisting of few and less informative primitives (from left: entry door, closet door, refrigerator and dishwasher).

yielding often to heavy occlusion and clutter. Figure 4.1 (bottom row) shows instances of bathtub symbols as they appear within architectural floor plans, suffering from heavy clutter and occlusion. Such occlusion, clutter and degradation can strongly degrade the performance of symbol spotting systems. If, as a result, the shape of the symbols appears considerably distorted, state-of-the-art symbol spotting methods cannot detect the degraded symbols. In this chapter, we aim to provide a method that is robust to heavy occlusion and clutter.

A third issue relates to the graphical simplicity of symbols. Simple (trivial) symbols that do not have complex structures, such as those shown in Figure 4.2, can be challenging for many traditional symbol spotting methods. As can be seen from the figure, the constituent primitives of these symbols are limited and structural-based methods cannot extract well-informed descriptions. In this chapter, we successfully address the detection of symbols of varying graphical complexity (from very simple to highly complex).

4.0.1 Contributions

This chapter proposes a DL-based framework for detecting symbols in digital real-world architectural floor plans. Our contributions are two-fold.

1. We leverage recent advances in DL by adapting a YOLO-based [97] object detection network to the problem of symbol spotting in real-world architectural floor plans. We propose a training strategy based on tiles, which allows us to circumvent many issues particular to DL object detection networks, including the size of the objects of interest relative to the size of the images, aspect ratios, and data augmentation.

2. Our proposed DL-based symbol spotting framework successfully addresses the main issues of traditional on-the-fly symbol spotting, namely graphical notation variability, occlusion and clutter, and variable graphical complexity of symbols.

The remainder of this chapter is structured as follows: Section 4.1 details our symbol spotting approach, Section 4.2 discusses experimental results, and Section 4.3 presents concluding remarks.

4.1 Proposed Method

The recent success of DL-based systems and CNN has revolutionized the object detection field. Popular networks such as Single Shot Multibox Detector (SSD) [109], YOLO [110, 97, 111] and Faster R-CNN [112] can be used to detect thousands of classes in natural scenes. Their success is due in large part to the existence of large annotated datasets such as Pascal VOC [113], MS COCO [100], and ImageNet [99].

In this work, we first build a dataset of real-world architectural floor plans. We then use this dataset to train a single shot detector based on the YOLOv2 [97] architecture for spotting architectural symbols within architectural floor plan images. Figure 4.5 offers an overview of our proposed framework. The dataset preparation and our approach based on YOLOv2 are presented in detail next.

4.1.1 Dataset Preparation

From a library of proprietary digital architectural drawings, designed by 10 architectural firms, we selected 115 different units showing various levels of difficulty in terms of density of visual information, occlusion, and clutter. Architects typically share floor plans in the PDF format. We converted the PDFs into 150 DPI images, and annotated 12 architectural symbol classes, such as bathroom sinks, windows, and entry doors (see Figure 4.3). We

cannot make the dataset of real-world architectural plans public due to intellectual property issues, but are working towards securing the necessary permissions for a future release.

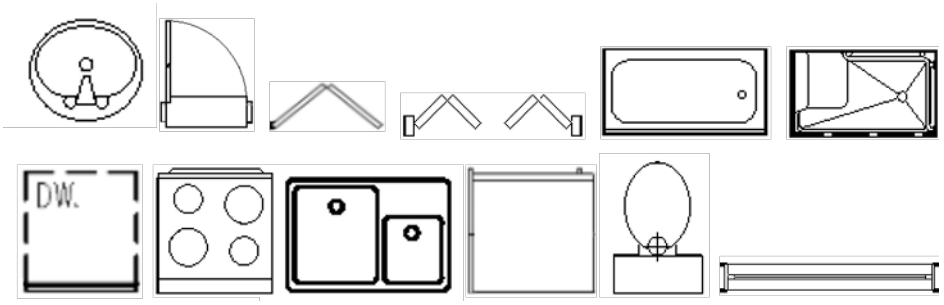


Figure 4.3: Examples of each symbol class. First row from left: bathroom sink, entry door, single folding door, double folding door, bathtub, shower. Second row: dishwasher, range, kitchen sink, refrigerator, toilet, and window.

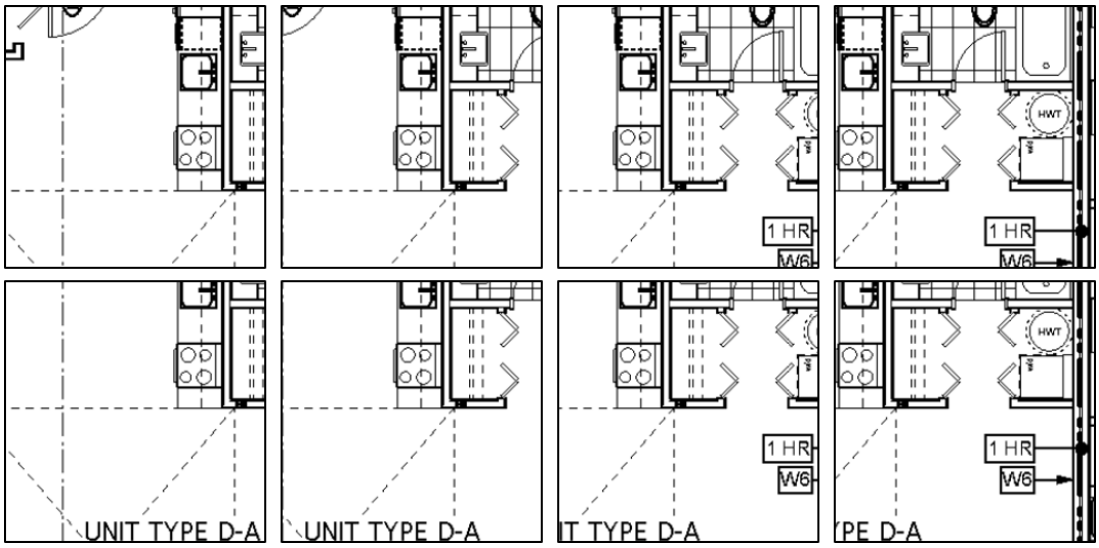


Figure 4.4: Data augmentation via image tiling strategy. The "range" symbol appears at various locations within the tiles, which also include various other symbols.

We face several problems when dealing with architectural floor plan images in the context of DL systems. First, the average floor plan size is 5400×3600 pixels, whereas individual symbols are very small (e.g. 70×80 pixels for a bathroom sink). As a result, symbols tend to disappear in the output feature map of CNNs, preventing them from being detected. In addition, floor plan images have diverse aspect ratios and resizing them to a fixed size,

as required by CNN architectures, dramatically changes the symbol morphology and thus decreases the classification performance.

We propose a tiling strategy to tackle the above problems, which uses a scale parameter α and stride size S. First, all the $[\alpha M \times \alpha M]$ overlapping tiles that have a starting point at least S pixels apart are extracted from the floor plan images. $[M \times M]$ is the required input size of the utilized CNN, which is usually less than $[256 \times 256]$ [110]. Tiles that do not encompass at least one complete symbol are automatically discarded from the training dataset. The tile size must be selected so that tiles are larger than symbols. Also, selecting larger tiles can boost their contextual information, as in architectural plans, the occurrences of some symbols might be spatially correlated. For instance, we can expect to see a bathroom sink symbol in the vicinity of a toilet symbol. At the same time, the tiles must be small enough so that the symbols still appear in the deeper layers and output of the CNNs. Tiles are also useful for data augmentation. Figure 4.4 shows neighboring tiles containing a 'range' symbol captured at different locations within the tiles. The tile size in the figure is $[224 \times 224]$ (i.e. $\alpha = 1$ and $M = 224$, required by ResNet50 [114]) and $S = 50$ pixels.

4.1.2 Symbol Spotting Using YOLOv2

Single shot object detection architectures based on image grids (such as YOLO) seem appropriate and accurate enough to localize architectural symbol boundaries, compared to more complex and heavier two-stage classification architectures (such as Faster R-CNN [112]), due to the following considerations. Floor plan images differ from natural scene images (for which most CNN-based object detection systems were developed) on several aspects. Floor plans are typically binary or grey-level with a small number of possible symbol classes, compared to colorful natural scene images with a large vocabulary of objects. Additionally, floor plans have a simpler background and chances of overlap between symbols is low (this does not apply of course to other parts of the image such as textual in-

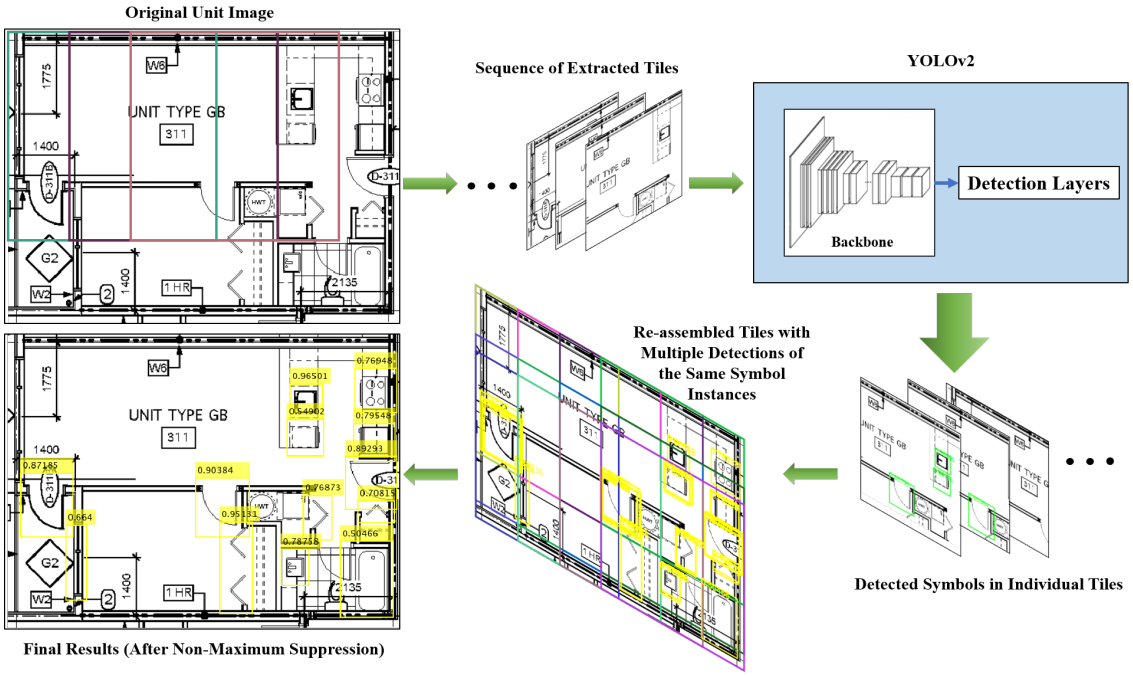


Figure 4.5: Proposed symbol spotting framework. Overlapping tiles from the input image are passed through YOLOv2 and individually processed. Non-maximum suppression is carried out to remove multiple detections of the same symbol instances due to the tiling strategy.

formation and measurements, which may have a considerable overlap with symbols). Here, we use YOLOv2 [97] as a single shot object detection architecture. We selected YOLOv2 instead of YOLO [110] because of its higher localization accuracy and recall performance. The improvements of YOLOv3 [111] consist mainly in a prediction across three scales and a better feature extraction network, but at the cost of a slower and heavier system. As architectural symbols have similar sizes and simpler structures compared to objects in natural scenes, YOLOv3 cannot offer a noticeable improvement over YOLOv2. In YOLOv2, the input image is divided into non-overlapping grids. Each grid can only predict one object. A backbone CNN (e.g. Darknet19) extracts features and for each grid, a set of prior anchors are evaluated based on a loss function which penalizes localization, confidence and classification errors.

In the training phase, we use the tile dataset (described in Section 4.1.1) to train the

network. In the inference phase, the input image is first broken down into tiles. Each tile is then passed through the network and symbols are detected. Figure 4.5 shows the inference process. The detected symbols in the overlapping tiles are shown in the bottom-right image. As a given instance of a symbol typically appears in several tiles, it is detected multiple times. To refine and concatenate the results, we perform a non-maximum suppression step as follows. For overlapping detections, we compare all pairs of bounding boxes. If their overlap is larger than a threshold (a percentage of the size of the smaller bounding box), the bounding box with the highest classification score is retained. In cases of close scores, the larger bounding box is selected and the smaller one is removed. The bottom-left image in Figure 4.5 shows the final results for a 10% threshold.

4.2 Results and Discussion

We assess our framework on a real-world floor plan dataset and on SESYD, a public dataset of synthetic documents. For both datasets, we evaluate the performance on individual tiles first, and then assess entire floor plans. We provide a comparative analysis of our approach with respect to state-of-the-art symbol spotting methods for SESYD only, as code implementations of these methods are either unavailable or not functional on our real-world dataset.

4.2.1 Real-World Images

From the 115 units of the dataset (see Sec. 4.1.1), we used 90 units for extracting tiles. The remaining 25 units are used as a test set for evaluating the framework on entire floor plans. Given $S = 50$ and a tile size of $[227 \times 227]$, the 90 units generated 4707 tiles containing at least one complete symbol. We randomly selected 80% of those 4707 tiles for training the network, with the remaining 20% tiles used for validation. During training,

we employed the Adam optimizer [115] with a mini-batch size of 30, a fixed learning rate of 10^{-4}, and data augmentation with horizontal and vertical flipping and rotation and scale changes. Moreover, 10 prior anchors were calculated from the size of the symbols. Anchor boxes (or priors) are template bounding boxes that are used in each cell in the image as initial guesses of the object locations. We experimented with three different backbones, the original Darknet19 [97], as well as ResNet50 [114] and Xception [116].

Table 4.1: Performance evaluation on the tile validation set for two datasets and different backbones.

Dataset	Backbone	*mAP*	AP_{50}	AP_{75}
Real-world	ResNet50 [114]	**72.40**	**96.20**	**90.13**
	Darknet19	61.53	93.7	72.41
	Xception [116]	51.03	87.58	55.01
SESYD	ResNet50	78.15	97.92	91.42

Table 4.2: Performance evaluation per symbol class and globally on the real-world test dataset for different backbones.

Symbol	ResNet50		Darknet19	
	AP_{50}	AP_{75}	AP_{50}	AP_{75}
Bathtub	91.67	91.67	95.83	95.83
Toilet	100.00	50.87	100.00	77.27
Kitchen Sink	91.07	77.91	88.21	51.74
Bathroom Sink	84.97	56.63	83.02	46.73
Closet Door (double)	86.96	39.47	91.30	50.43
Entry Door	86.35	82.47	89.81	83.60
Oven	100.00	95.83	91.67	87.50
Window	75.64	31.78	77.75	33.23
Refrigerator	87.50	76.38	91.49	66.79
Closet Door (single)	88.76	59.99	95.46	23.77
Dishwasher	78.89	66.35	67.00	67.00
Shower	100.00	100.00	100.00	100.00
AP	**89.32**	**69.11**	**89.30**	**65.32**
mAP	**59.03**		**56.50**	

Table 4.1 (first three rows) shows the performance on the tile validation set, whereas Table 4.2 shows the performance per symbol class and the global performance for the test set of 25 entire floor plans. In the tables, mAP, AP_{50} and AP_{75} represent the mean average precision and the average precision for IoUs equal to 50% and 75%, respectively. The Intersection over Union (IoU) is obtained as follows:

$$IoU(A, B) = |A \cap B| \,/\, |A \cup B| \tag{4.1}$$

where A and B are the bounding boxes of the detected symbol and the ground truth symbol. From Table 4.1, we can see that the ResNet50 backbone significantly outperforms Darknet19 and Xception, with Xception having the lowest performance. Looking specifically at the AP_{50} metric, as an IoU of 50% is acceptable in symbol spotting, the average precision is very high. From Table 4.2, again focusing on AP_{50}, we can see that our method performs strongly for most symbols, with some yielding 100% precision. The lowest score is obtained for the window symbol, which is a particularly difficult case due to its triviality and varying aspect ratio. Incorporating contextual information of walls could help to improve the window detection results, as the windows are always installed in the walls. This can be done, for instance, by integrating the output of a wall detection/segmentation method into the system and then look for only windows (or door) in those locations.

Fig. 4.6 shows examples of detected entry doors, using the ResNet50 backbone, with the bounding boxes and detection scores (max = 1) highlighted. This figure showcases the efficiency of our DL-based symbol spotting system compared to the traditional methods. Our system successfully addressed occlusion and boundary degradation, which can highly affect the raster-to-vector conversion and thus the structural representation of symbols in methods such as [83, 34, 36], rotation, which is one of the weaknesses of the pixel-based methods such as [29, 65], and graphical notation variability. Furthermore, as entry doors have a limited number of primitives, some of them cannot survive the vectorization step

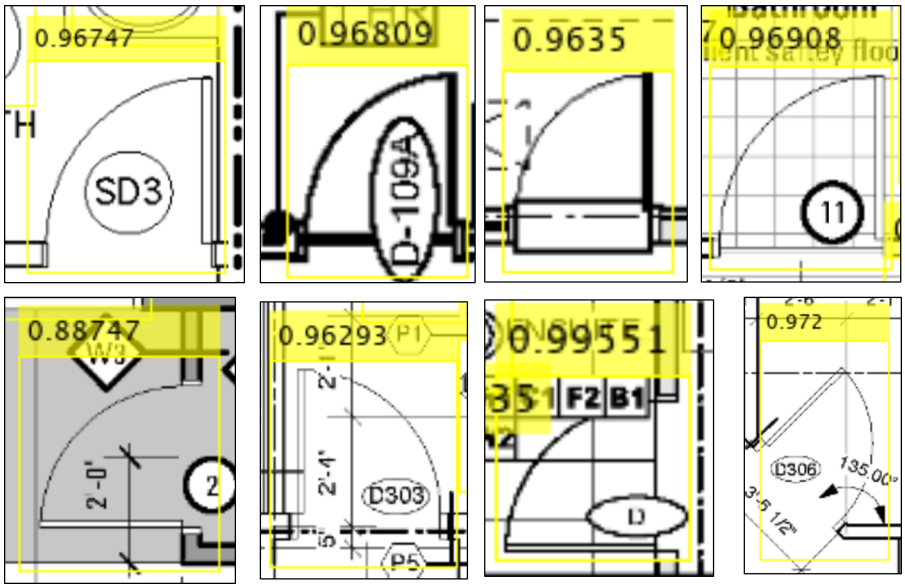


Figure 4.6: Detected entry doors and scores (max = 1) for various levels of occlusion and overlap, in real-world dataset.

required by vector-based methods. They also include very small closed regions that can easily make the symbol unrecognizable by methods that employ closed regions as constituent primitives, such as [42, 48, 84].

Fig. 4.7 shows symbol spotting results for four units with different designs and layouts, using the ResNet50 backbone. Qualitatively speaking, the results are excellent, and we see that our approach works well even in the presence of high levels of noise, occlusion and image degradation. Considering the varied sources of the plans, we can also conclude that our method successfully bridges the semantic gap related to intra-class graphical notation variability. Also, in terms of computational time, thanks to the high speed of YOLOv2, each image in the dataset can be processed in a matter of few seconds with a NVIDIA GeForce 1660ti GPU.

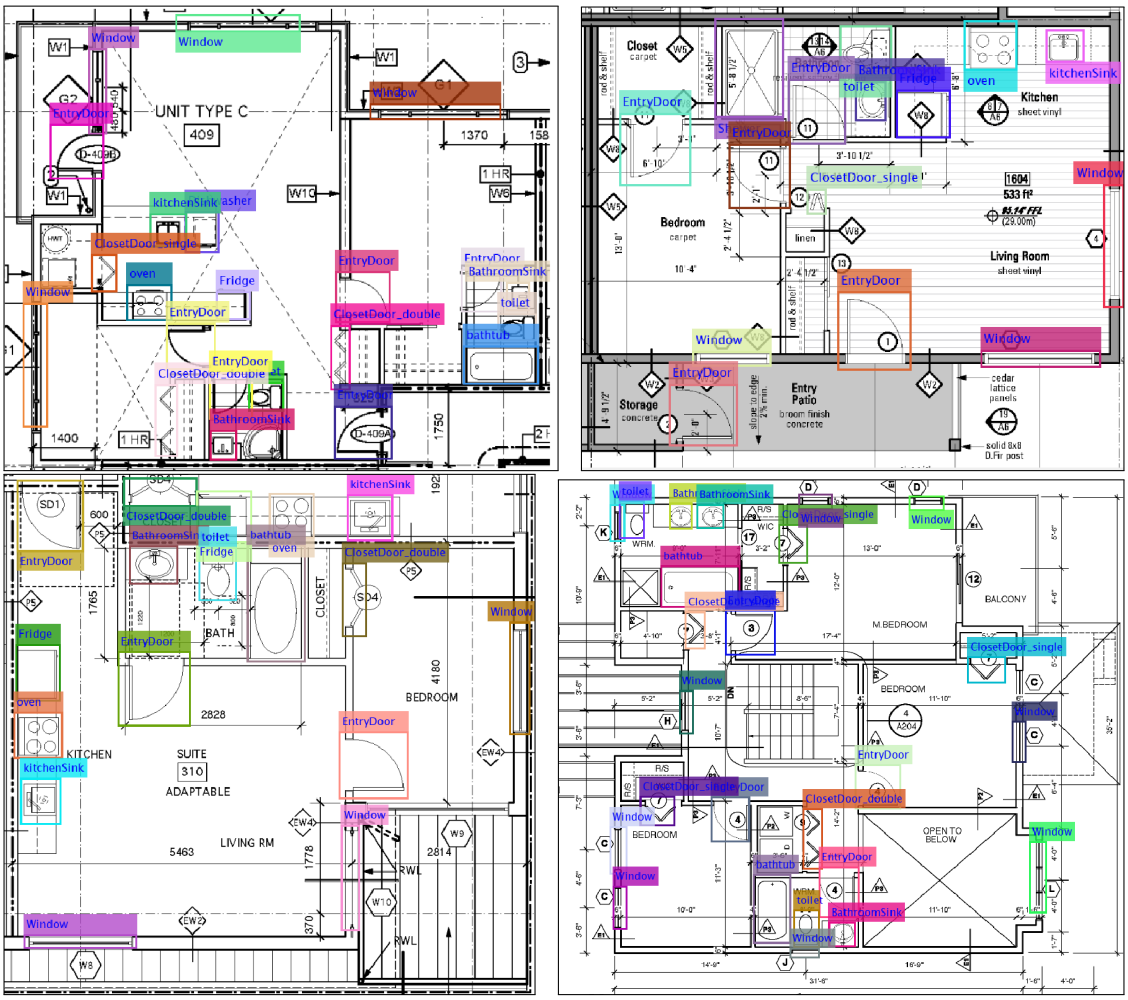


Figure 4.7: Examples of spotted symbols in real-world floor plan images.

4.2.2 SESYD Dataset

We also provide an evaluation on the public synthetic images of the SESYD dataset, which is the standard dataset in the field. Its synthetic floor plan collection includes 1000 floor plan images (some of which have very similar unit layouts), containing up to 16 different query symbol classes, with only one graphical notation per symbol class. For training the system, we randomly picked 50 floor plan images and extracted the tile images. Since images are large and the floor plans are sparser than real-world floor plans, we used [680×680] tiles with $S = 100$ to include more contextual information around each symbol.

This yielded 11,753 images divided into subsets of 9402 and 2351 tiles for training and validation purposes, respectively. To test our system on entire images, we used the selection from the GREC Symbol Recognition and Spotting contest [107]. This contest set contains 20 images from the original dataset of 1000 images (ideal) and three degraded versions (60 images). All of our results on SESYD are obtained with the ResNet50 backbone, as it yields a better performance on the real-world dataset.

Table 4.3: Performance evaluation per symbol class and globally on the GREC contest test dataset (from SESYD).

Symbol	Ideal		Noise 1		Noise 2		Noise 3	
	AP_{50}	AP_{75}	AP_{50}	AP_{75}	AP_{50}	AP_{75}	AP_{50}	AP_{75}
armchair	87.10	63.96	88.89	61.85	90.48	54.37	100.00	90.97
bed	100.00	100.00	92.11	92.11	89.47	56.64	94.74	92.03
door1	100.00	34.73	100.00	48.34	100.00	53.68	100.00	53.62
door2	100.00	100.00	100.00	100.00	0.00	0.00	100.00	100.00
sink1	100.00	0.00	100.00	0.00	100.00	0.00	100.00	0.00
sink2	98.86	98.86	98.38	98.38	100.00	62.39	100.00	100.00
sink3	82.35	82.35	92.31	88.46	100.00	100.00	95.83	91.30
sink4	100.00	32.11	100.00	60.24	100.00	39.13	100.00	47.87
sofa1	100.00	76.91	100.00	34.97	97.30	57.44	98.08	54.48
sofa2	100.00	96.48	100.00	65.72	100.00	82.02	100.00	45.08
table1	100.00	20.62	100.00	22.03	100.00	15.02	100.00	15.04
table2	100.00	42.00	100.00	81.08	100.00	45.63	100.00	40.38
table3	100.00	72.02	100.00	38.34	100.00	100.00	100.00	100.00
tub	100.00	100.00	95.00	78.62	100.00	71.33	100.00	74.69
window1	62.20	0.00	58.27	0.00	71.01	0.00	59.31	0.00
window2	11.65	0.00	36.78	0.00	35.93	0.00	13.21	0.00
AP	**90.13**	**57.50**	**91.36**	**54.38**	**89.30**	**65.32**	**91.32**	**56.59**
mAP	**54.08**		**53.93**		**47.27**		**54.85**	

Table 4.1 (last row) shows the results on the validation tile set, and Table 4.3 shows the performance per symbol class and the global performance for the test set of 80 entire floor plans. Looking at AP_{50} in both tables, our framework yields a very high precision rate, with 100.00% for many of the symbol classes. Again, the window classes (window1 and window2) are the most problematic ones, and would benefit from additional contextual

Table 4.4: Instancewise and pixelwise evaluation of symbol spotting approaches on SESYD.

Method	Eval.	P	R	F	Queries
Nguyen *et al.* [33]	Instance	70.00	88.00	79.50	6
Broelemann *et al.* [40]	Instance	75.17	93.17	83.21	6
Dutta *et al.* [44]	Instance	62.33	95.67	75.50	All
Le Bodic *et al.* [42]	Instance	90.00	81.00	85.30	All
Nayef and Breuel [1]	Instance	98.90	98.10	98.50	12
Winner in [107] (ideal)	Pixel	62.00	99.00	76.00	All
Winner in [107] (noise #1)	Pixel	64.00	98.00	77.00	All
Winner in [107] (noise #2)	Pixel	62.00	93.00	74.00	All
Winner in [107] (noise #3)	Pixel	57.00	98.00	72.00	All
Proposed method (ideal)	Instance	98.56	97.31	97.93	
	Pixel	77.35	98.97	86.83	All
Proposed method (noise #1)	Instance	99.32	97.15	98.22	
	Pixel	77.69	97.28	86.39	All
Proposed method (noise #2)	Instance	99.11	99.11	99.11	
	Pixel	76.48	97.65	85.78	All
Proposed method (noise #3)	Instance	99.46	93.93	96.62	
	Pixel	77.63	97.34	86.38	All

information.

Table 4.4 compares our results with other published symbol spotting approaches. In this context, the evaluation metrics in the literature differ from the ones commonly used for assessing object detection networks, and are computed instancewise and pixel-wise. For the instancewise metrics, detected symbols that have some overlap with the ground truth are all counted as positive detections, and precision, recall and F-score values are calculated accordingly. Pixelwise metrics, based on relevant and non-relevant retrieved pixels, refine the localization assessment [92]. In Table 4.4, P, R and F stand for precision, recall and F-score, respectively. The 'Queries' column indicates how many of the 16 symbols in the dataset are employed in the evaluation. From the table, it can be seen that our method significantly outperforms all other methods, including the winner of GREC contest. This method, which is published in [1], has comparable performance with our proposed ap-

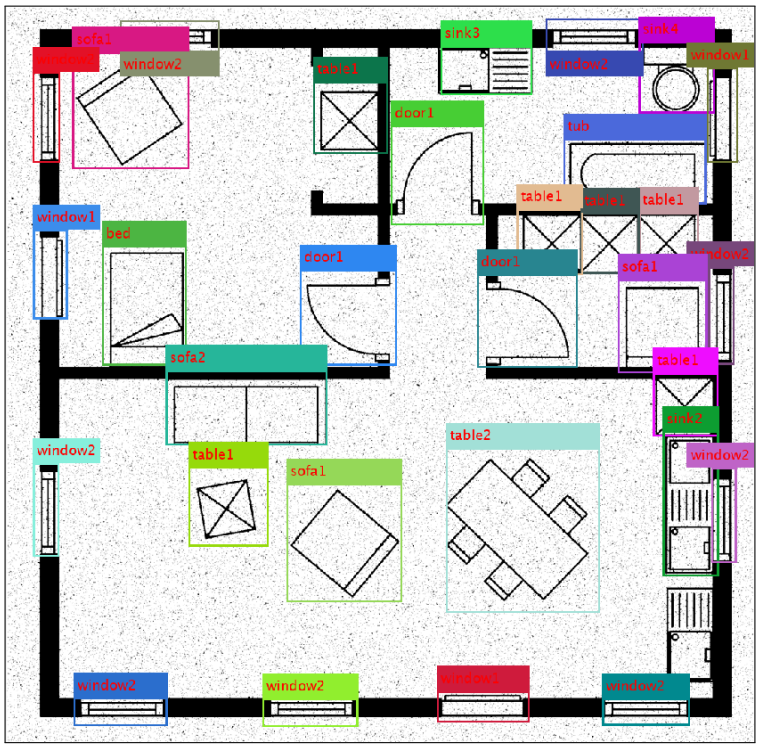


Figure 4.8: Examples of spotted symbols on a degraded SESYD floor plan (noise #3).

proach but note that the evaluation is conducted on a limited subset of the symbols (12 out of 16 queries) by probably excluding the more challenging cases.

Fig. 4.8 shows examples of spotted symbols on a SESYD layout for the noise #3 degraded version. All symbol instances are correctly detected except for one kitchen sink (bottom right). Although SESYD does not include graphical notation variability, occlusion nor clutter as the real-world dataset does, it does allow us to conclude that our approach is able to deal with various degradation and noise levels.

4.3 Conclusion

This chapter proposed a novel approach to symbol spotting utilizing a DL-based framework, showcased on the challenging application of real-world digital architectural floor plan analysis. We adapted an object detection network based on the YOLO architecture,

and proposed a training strategy based on tiles, allowing us to address many issues of the network regarding the relative small size of symbols compared to entire floor plans, aspect ratios, and data augmentation. Experiments on a dataset of real-world floor plans demonstrate that our proposed method successfully spots symbols in conditions under which traditional symbol spotting methods cannot cope, i.e. symbols with low intra-class similarity and of variable graphical complexity, even in the presence of occlusion and clutter. The ResNet50 backbone within the YOLO framework yields the best results compared to the original Darknet19 and Xception backbones. Additional experiments on the public SESYD dataset also confirm that our method can deal with various degradation and noise levels and outperforms existing symbol spotting methods.

Chapter 5

Conclusions

The objective of this research was to explore different approaches of locating architectural symbols in 2D top-view floor plans. Our main focus was specifically on real-world images, where occlusion, noise and overlaps between symbols and other graphical notations (such as measurements, annotations, revisions, etc.) make the localization a more challenging and daunting task than synthetic datasets. To offer an effective solution, we first conducted a thorough literature review on both symbol spotting and DL-based symbol detection approaches in Chapter 2 (published in [18]) with a focus on architectural floor plans.

By considering the current limitations of symbol spotting methods, we then proposed a new approach (published in [19]) in Chapter 3. In the proposed method, we first introduced a new descriptor to capture the salient geometric properties of the constituent primitives of symbols. For this, some voting-based techniques along with a new ellipse detection algorithm was employed to encode the primitive in the format of five parameters of an ellipse. A primitive-aware graph is used to extract the spatial relations between adjacent primitives and then a statistical method finds the instances of a symbol in the input image. Finally, a modified cross-correlation function is proposed to precisely prune and fine-tune the detected symbols at the pixel level. The method outperforms the current state-of-the-

art on the public SESYD dataset and shows promising results on our custom real-world dataset.

In Chapter 4 we relaxed the on-the-fly property of symbol spotting methods and leveraged the recent advances in DL-based object detection neural networks to detect architectural symbols (published in [20]). We used a tiling strategy for training YOLOv2 which allows us to 1) detect small symbols in large input images, 2) collect semantic information around each symbol, 3) not change the aspect ratio of the input image and consequently the morphology of symbols and finally 4) data augmentation. The results show good performance on both noisy images of SESYD and more importantly, a challenging dataset of real-world images. In the next section, we highlight the pros and cons of both proposed methods, which enables us to perform a more generic comparison between the traditional symbol spotting techniques vs. DL-based methods.

5.1 Traditional vs. Learning-based Methods

In Chapter 4, we mentioned the fact that some recent neural network-based methods relax the on-the-fly requirement of symbol spotting to reframe the issue as an object detection problem. Although this makes for a different problem with its own set of applications, we believe that it is relevant to discuss the relative performances of traditional symbol spotting methods and of learning-based methods.

The main drawback of learning-based systems, compared to traditional symbol spotting methods, is their requirement for a (large) training set. It is an arduous task to generate a training dataset that is large enough to train learning-based systems, especially DL ones, and to accurately represent the graphical notation variability of architectural symbols. On the other hand, DL-based methods offer automatic detection of symbols that have various graphical notations. They are also more robust to noise and occlusion compared to symbol

spotting approaches.

In terms of pure performance, we can make a comparison between traditional and learning-based methods by referencing our works in chapters 3 and 4. For having a fair comparison, we refer to the performance of both systems on SESYD dataset and reiterate their instancewise and pixelwise results in tables 3.1, 3.2 and 4.4. These tables demonstrate that DL-based approach offered improvement on recall values for all the ideal and noisy images (in Table 4.4, all the detected symbols with at least 50% IoU with the ground truth are considered as true positives). However, this method shows inferior localization accuracy (P_p) compared to the proposed geometry-based symbol spotting method. This is mainly because of the richer presentation of symbols in the geometry-based symbol spotting and its pixel-based refinement stage, which can allocate more accurate bounding boxes to each retrieved symbol. In YOLOv2 the network starts with some initial anchors and then tries to predict offset values to adjust the anchors to the object locations; this can affect this method's ability to outline accurate bounding boxes.

To conclude:

- the training set requirement of learning-based systems makes them unable to detect variations in symbol designs that were not represented in the training set while symbol spotting approaches have more flexibility in this case thanks to their on-the-fly property,

- learning-based systems cannot recognize new symbol classes that were not part of the symbol vocabulary of the training set, at least not without an updated and large-enough training set and a new iteration of training,

- learning-based systems can be faster than traditional techniques and seem to offer generally a better performance on images that are similar to their training set.

- The proposed system based on YOLOv2 shows great performance in dealing with

real-world challenges such as heavy occlusion and clutter. Symbol spotting approaches are not powerful enough in these scenarios since the redundant information in the image degrades the performance of the description step.

- Single shot object detection networks seem to be more suitable for architectural floor plans as they are computationally fast, light and may need less data to train compared to two stage networks. The limited number of classes, not having overlap between symbols and large backgrounds in this application make the localization strategy of single shot networks efficient enough to detect symbols, thus there is no need for having an extra region proposal network in the architecture to boost the accuracy.

- Symbol spotting approaches, specifically our geometry-based method, offer richer representation of detected symbols compared to object detection networks. As explained in Chapter 3, our approach finds all the constituent primitives of the symbols and the matching phase can also give the exact similarity transformation that each symbol underwent (since the output of object detection networks are just vertical bounding boxes around detected symbols). This rich information offers better localization accuracy and more importantly, can be employed for vectorization purposes. This is the most important advantage of spotting methods over the object detection networks.

5.2 Future Work

The analysis of symbols in digital architectural floor plans is a very industry-specific application, so there are limited resources publicly available for researchers. Because of intellectual property limitations, many of the proposed methods in the literature are not reproducible and it is hard to fairly compare the new approaches with the state-of-the-art. Our ongoing research involves a collaboration with architectural firms in order to assem-

ble a public dataset of real-world architectural plans with a large and varied vocabulary of symbols. We are currently in the process of securing permissions from various architectural firms to release a public dataset of real-world images.

We will explore parallel computing to speed up the processing time of our symbol spotting approach, especially for the description phase. As mentioned, the proposed methods are parts of a bigger pipeline that is being developed for our industrial partner and so we used Matlab here just to showcase the performance of our systems. The entire pipeline is eventually going to be professionally developed by our industrial partner for commercial purposes.

Another research direction includes the integration of contextual information relating to walls and rooms to further improve the detection results of our systems. For the symbol spotting system, a feedback to/from other parts of pipeline which are responsible for locating walls and rooms can improve the performance of both ends. For the DL-based approach, relying on a single network for detecting all the floor plan elements can incorporate more contextual information and may lead to better performance rather than employing different modules for each individual task.

Bibliography

[1] Nibal Nayef and Thomas M Breuel. On the use of geometric matching for both: Isolated symbol recognition and symbol spotting. In *International Workshop on Graphics Recognition (GREC'11)*, pages 36–48. Springer, 2011.

[2] Chen Liu, Jiajun Wu, Pushmeet Kohli, and Yasutaka Furukawa. Raster-to-vector: Revisiting floorplan transformation. In *Int. Conf. Comput. Vis. (ICCV'17)*, pages 2195–2203, 2017.

[3] Ahti Kalervo, Juha Ylioinas, Markus Häikiö, Antti Karhu, and Juho Kannala. Cubicasa5k: A dataset and an improved multi-task model for floorplan image analysis. In *Scand. Conf. Image Anal. (SCIA'19)*, pages 28–40. Springer, 2019.

[4] Marçal Rusiñol and Josep Lladós. *Symbol Spotting in Digital Libraries*, volume 1. Springer-Verlag London, 2010.

[5] A. K. Chhabra. Graphic symbol recognition: An overview. In *Int. Workshop Graph. Recognit. (GREC'97)*, pages 68–79. Springer, 1997.

[6] Rangachar Kasturi and Huizhu Luo. Research advances in graphics recognition: An update. In *Advances in Document Image Analysis*, pages 99–110. Springer, 1997.

[7] Luigi P Cordella and Mario Vento. Symbol recognition in documents: a collection of techniques? *International Journal on Document Analysis and Recognition (IJDAR'00)*, 3(2):73–88, 2000.

[8] Josep Lladós, Ernest Valveny, Gemma Sánchez, and Enric Marti. Symbol recognition: Current advances and perspectives. In *International Workshop on Graphics Recognition (GREC'01)*, pages 104–128. Springer, 2001.

[9] Karl Tombre, Salvatore Tabbone, and Philippe Dosch. Musings on symbol recognition. In *International Workshop on Graphics Recognition (GREC'05)*, pages 23–34. Springer, 2005.

[10] Marçal Rusiñol and Josep Lladós. *Symbol Spotting in Digital Libraries*, volume 1. Springer, 2010.

[11] Salvatore Tabbone and Oriol Ramos Terrades. An overview of symbol recognition. *Handbook of Document Image Processing and Recognition*, pages 523–551, 2014.

[12] KC Santosh. Complex and composite graphical symbol recognition and retrieval: A quick review. In *International Conference on Recent Trends in Image Processing and Pattern Recognition (RTIP2R'16)*, pages 3–15. Springer, 2016.

[13] Kenneth M Sayre. Machine recognition of handwritten words: A project report. *Pattern recognition*, 5(3):213–228, 1973.

[14] Mathieu Delalandre, Ernest Valveny, Tony Pridmore, and Dimosthenis Karatzas. Generation of synthetic documents for performance evaluation of symbol recognition & spotting systems. *International Journal on Document Analysis and Recognition*, 13(3):187–207, 2010.

[15] E. Valveny, M. Delalandre, R. Raveaux, and B. Lamiroy. Report on the symbol recognition and spotting contest. In *Int. Workshop Graph. Recognit. (GREC'11)*, pages 198–207. 2011.

[16] Muhammad Muzzamil Luqman, Thierry Brouard, Jean-Yves Ramel, and Josep Llados. A content spotting system for line drawing graphic document images. In *20th International Conference on Pattern Recognition (ICPR'10)*, pages 3420–3423. IEEE, 2010.

[17] Jan Rendek, Bart Lamiroy, and Karl Tombre. A few steps towards on-the-fly symbol recognition with relevance feedback. In *International Workshop on Document Analysis Systems*, pages 604–615. Springer, 2006.

[18] Alireza Rezvanifar, Melissa Cote, and Alexandra Branzan Albu. Symbol spotting for architectural drawings: state-of-the-art and new industry-driven developments. *IPSJ Transactions on Computer Vision and Applications*, 11(1):2, 2019.

[19] Alireza Rezvanifar, Melissa Cote, and Alexandra Branzan Albu. Geometry-based symbol spotting in born-digital architectural floor plans. *Journal of Electronic Imaging*, 30(4):043015, 2021.

[20] Alireza Rezvanifar, Melissa Cote, and Alexandra Branzan Albu. Symbol spotting on digital architectural floor plans using a deep learning-based framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW'20)*, pages 568–569, 2020.

[21] Melissa Cote, Alireza Rezvanifar, and Alexandra Branzan Albu. Automatic generation of electrical plan documents from architectural data. In *Proceedings of the ACM Symposium on Document Engineering 2020*, pages 1–4, 2020.

[22] Karl Tombre and Bart Lamiroy. Graphics recognition from re-engineering to retrieval. In *Seventh International Conference on Document Analysis and Recognition (ICDAR'03).*, pages 148–155. IEEE, 2003.

[23] Ernest Valveny and Philippe Dosch. Symbol recognition contest: A synthesis. In *International Workshop on Graphics Recognition (GREC'03)*, pages 368–385. Springer, 2003.

[24] Philippe Dosch and Ernest Valveny. Report on the second symbol recognition contest. In *Proceedings of the 6th International Conference on Graphics Recognition: Ten Years Review and Future Perspectives (GREC'05)*, volume 3926, pages 381–397. Springer, 2005.

[25] Ernest Valveny, Philippe Dosch, Alicia Fornes, and Sergio Escalera. Report on the third contest on symbol recognition. In *International Workshop on Graphics Recognition (GREC'07)*, pages 321–328. Springer, 2007.

[26] Salvatore Tabbone, Laurent Wendling, and Karl Tombre. Matching of graphical symbols in line-drawing images using angular signature information. *International Journal on Document Analysis and Recognition*, 6(2):115–125, 2003.

[27] Su Yang. Symbol recognition via statistical integration of pixel-level constraint histograms: A new descriptor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(2):278–281, 2005.

[28] Alicia Fornés, Sergio Escalera, Josep LLadós, Gemma Sánchez, Petia Radeva, and Oriol Pujol. Handwritten symbol recognition by a boosted blurred shape model with error correction. *Pattern Recognition and Image Analysis*, pages 13–21, 2007.

[29] Sergio Escalera, Alicia Fornés, Oriol Pujol, Petia Radeva, Gemma Sánchez, and Josep Lladós. Blurred shape model for binary and grey-level symbol recognition. *Pattern Recognition Letters*, 30(15):1424–1433, 2009.

[30] Sergio Escalera, Alicia Fornés, Oriol Pujol, Alberto Escudero, and Petia Radeva. Circular blurred shape model for symbol spotting in documents. In *16th IEEE International Conference on Image Processing (ICIP'09)*, pages 2005–2008. IEEE, 2009.

[31] Sergio Escalera, Alicia Fornés, Oriol Pujol, Josep Lladós, and Petia Radeva. Circular blurred shape model for multiclass symbol recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 41(2):497–506, 2011.

[32] Thi Oanh Nguyen, Salvatore Tabbone, and O Ramos Terrades. Symbol descriptor based on shape context and vector model of information retrieval. In *Document Analysis Systems, 2008. DAS'08. The Eighth IAPR International Workshop on*, pages 191–197. IEEE, 2008.

[33] Thi-Oanh Nguyen, Salvatore Tabbone, and Alain Boucher. A symbol spotting approach based on the vector model and a visual vocabulary. In *2009 10th International Conference on Document Analysis and Recognition (ICDAR'09)*, pages 708–712. IEEE, 2009.

[34] Hervé Locteau, Sébastien Adam, Eric Trupin, Jacques Labiche, and Pierre Héroux. Symbol spotting using full visibility graph representation. In *Workshop on Graphics Recognition*, pages 49–50, 2007.

[35] Jean-Yves Ramel, Nicole Vincent, and Hubert Emptoz. A structural representation for understanding line-drawing images. *International Journal on Document Analysis and Recognition*, 3(2):58–66, 2000.

[36] KC Santosh, Bart Lamiroy, and Laurent Wendling. Symbol recognition using spatial relations. *Pattern Recognition Letters*, 33(3):331–341, 2012.

[37] KC Santosh, Bart Lamiroy, and Laurent Wendling. Spatio-structural symbol description with statistical feature add-on. In *International Workshop on Graphics Recognition (GREC'11)*, pages 228–237. Springer, Berlin, 2013.

[38] KC Santosh, Bart Lamiroy, and Laurent Wendling. Integrating vocabulary clustering with spatial relations for symbol recognition. *International Journal on Document Analysis and Recognition (IJDAR'14)*, 17(1):61–78, 2014.

[39] Klaus Broelemann, Anjan Dutta, Xiaoyi Jiang, and Josep Lladós. Hierarchical graph representation for symbol spotting in graphical document images. *Structural, Syntactic, and Statistical Pattern Recognition*, pages 529–538, 2012.

[40] Klaus Broelemann, Anjan Dutta, Xiaoyi Jiang, and Josep Lladós. Hierarchical plausibility-graphs for symbol spotting in graphical documents. In *International Workshop on Graphics Recognition (GREC'13)*, pages 25–37. Springer, 2013.

[41] Pierre Le Bodic, Hervé Locteau, Sébastien Adam, Pierre Héroux, Yves Lecourtier, and Arnaud Knippel. Symbol detection using region adjacency graphs and integer linear programming. In *10th International Conference on Document Analysis and Recognition (ICDAR'09)*, pages 1320–1324. IEEE, 2009.

[42] Pierre Le Bodic, Pierre Héroux, Sébastien Adam, and Yves Lecourtier. An integer linear program for substitution-tolerant subgraph isomorphism and its use for symbol spotting in technical drawings. *Pattern Recognition*, 45(12):4214–4224, 2012.

[43] Alessio Barducci and Simone Marinai. Object recognition in floor plans by graphs of white connected components. In *21st International Conference on Pattern Recognition (ICPR'12)*, pages 298–301. IEEE, 2012.

[44] Anjan Dutta, Josep Lladós, Horst Bunke, and Umapada Pal. Near convex region adjacency graph and approximate neighborhood string matching for symbol spotting in graphical documents. In *2013 12th International Conference on Document Analysis and Recognition (ICDAR'13)*, pages 1078–1082. IEEE, 2013.

[45] Anjan Dutta, Josep Lladós, and Umapada Pal. Bag-of-graphpaths descriptors for symbol recognition and spotting in line drawings. In *Graphics Recognition. New Trends and Challenges (GREC'11)*, pages 208–217. Springer, 2013.

[46] Nibal Nayef and Thomas M Breuel. Statistical grouping for segmenting symbols parts from line drawings, with application to symbol spotting. In *International Conference on Document Analysis and Recognition (ICDAR'11)*, pages 364–368. IEEE, 2011.

[47] KC Santosh, Laurent Wendling, and Bart Lamiroy. Bor: Bag-of-relations for symbol retrieval. *International Journal of Pattern Recognition and Artificial Intelligence*, 28(06):1450017, 2014.

[48] Marçal Rusiñol and Josep Lladós. A region-based hashing approach for symbol spotting in technical documents. In *Proceedings of the International Workshop on Graphics Recognition (GREC'07)*, pages 104–113. Springer, 2007.

[49] Dengsheng Zhang and Guojun Lu. Review of shape representation and description techniques. *Pattern Recognition*, 37(1):1–19, 2004.

[50] Shivani Agarwal, Aatif Awan, and Dan Roth. Learning to detect objects in images via a sparse, part-based representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1475–1490, 2004.

[51] Frederic Jurie and Bill Triggs. Creating efficient codebooks for visual recognition. In *Tenth IEEE International Conference on Computer Vision (ICCV'05)*, volume 1, pages 604–610. IEEE, 2005.

[52] Josef Sivic and Andrew Zisserman. Video google: Efficient visual search of videos. In *Toward category-level object recognition*, pages 127–144. Springer, Berlin, 2006.

[53] Do Thanh Ha, Salvatore Tabbone, and Oriol Ramos Terrades. Spotting symbol over graphical documents via sparsity in visual vocabulary. In *International Conference on Recent Trends in Image Processing and Pattern Recognition (RTIP2R'16)*, pages 59–70. Springer, 2016.

[54] Muhammad Muzzamil Luqman, Jean-Yves Ramel, and Josep Lladós. Multilevel analysis of attributed graphs for explicit graph embedding in vector spaces. In *Graph Embedding for Pattern Analysis*, pages 1–26. Springer, 2013.

[55] Karl Tombre and Bart Lamiroy. Pattern recognition methods for querying and browsing technical documentation. In *Iberoamerican Congress on Pattern Recognition*, pages 504–518. Springer, 2008.

[56] KC Santosh, Bart Lamiroy, and Jean-Philippe Ropers. Inductive logic programming for symbol recognition. In *10th International Conference on Document Analysis and Recognition (ICDAR'09)*, pages 1330–1334. IEEE, 2009.

[57] Josep Lladós, Enric Martí, and Juan J. Villanueva. Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1137–1143, 2001.

[58] Christian Ah-Soon. A constraint network for symbol detection in architectural drawings. In *International Workshop on Graphics Recognition (GREC'97)*, pages 80–90. Springer, 1997.

[59] Christian Ah-Soon and Karl Tombre. Network-based recognition of architectural symbols. *Advances in Pattern Recognition*, pages 252–261, 1998.

[60] Christian Ah-Soon and Karl Tombre. Architectural symbol recognition using a network of constraints. *Pattern Recognition Letters*, 22(2):231–248, 2001.

[61] Bruno T Messmer and Horst Bunke. Automatic learning and recognition of graphical symbols in engineering drawings. In *International Workshop on Graphics Recognition (GREC'95)*, pages 123–134. Springer, 1995.

[62] Bruno T Messmer and Horst Bunke. Fast error-correcting graph isomorphism based on model precompilation. In *International Conference on Image Analysis and Processing (ICDAR'97)*, pages 693–700. Springer, 1997.

[63] David W Jacobs. Robust and efficient detection of salient convex groups. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(1):23–37, 1996.

[64] Edward Harrington Lockwood. *A book of curves*. Cambridge University Press, Cambridge, 1967.

[65] Jonathan Weber and Salvatore Tabbone. Symbol spotting for technical documents: An efficient template-matching approach. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR'12)*, pages 669–672. IEEE, 2012.

[66] Philippe Dosch and Josep Lladós. Vectorial signatures for symbol discrimination. In *International Workshop on Graphics Recognition (GREC'03)*, pages 154–165. Springer, 2003.

[67] Yehezkel Lamdan and Haim J Wolfson. Geometric hashing: A general and efficient model-based recognition scheme. In *1988 Second International Conference on Computer Vision*, pages 238–239. IEEE Computer Society, 1988.

[68] Marçal Rusiñol, Josep Lladós, and Gemma Sánchez. Symbol spotting in vectorized technical drawings through a lookup table of region strings. *Pattern Analysis and Applications*, 13(3):321–331, 2010.

[69] Marçal Rusiñol, Agnés Borràs, and Josep Lladós. Relational indexing of vectorial primitives for symbol spotting in line-drawing images. *Pattern Recognition Letters*, 31(3):188–201, 2010.

[70] Nibal Nayef and Thomas M Breuel. Graphical symbol retrieval using a branch and bound algorithm. In *17th IEEE International Conference on Image Processing (ICIP'10)*, pages 2153–2156. IEEE, 2010.

[71] Thomas M Breuel. Fast recognition using adaptive subdivisions of transformation space. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'92)*, pages 445–451. IEEE, 1992.

[72] Thomas M Breuel. Implementation techniques for geometric branch-and-bound matching methods. *Computer Vision and Image Understanding*, 90(3):258–294, 2003.

[73] Nibal Nayef and Thomas M Breuel. On the use of geometric matching for both: Isolated symbol recognition and symbol spotting. In *International Workshop on Graphics Recognition (GREC'11)*, pages 36–48. Springer, Berlin, 2013.

[74] Nibal Nayef and Thomas M Breuel. Combining geometric matching with svm to improve symbol spotting. In *Document Recognition and Retrieval XX (DDR'13)*, page 86580G. International Society for Optics and Photonics, 2013.

[75] Rashid Qureshi, Jean-Yves Ramel, and Hubert Cardot. Graph based shapes representation and recognition. *Graph-Based Representations in Pattern Recognition*, pages 49–60, 2007.

[76] Rashid Jalal Qureshi, Jean-Yves Ramel, Didier Barret, and Hubert Cardot. Spotting symbols in line drawing images using graph representations. In *Int. Workshop Graph. Recognit. (GREC'07)*, pages 91–103. Springer, 2007.

[77] R Qureshi, J Ramel, and Hubert Cardot. Graphic symbol recognition using flexible matching of attributed relational graphs. In *6th IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP'06)*, pages 383–388, 2006.

[78] J. Lerouge, M. Hammami, P. Héroux, and S. Adam. Minimum cost subgraph matching using a binary linear program. *Pattern Recognit. Lett.*, 71:45–51, 2016.

[79] Muhammad Muzzamil Luqman, Mathieu Delalandre, Thierry Brouard, Jean-Yves Ramel, and Josep Lladós. Employing fuzzy intervals and loop-based methodology for designing structural signature: An application to symbol recognition. *arXiv preprint arXiv:1004.5427*, 2010.

[80] Muhammad Muzzamil Luqman, Jean-Yves Ramel, Josep Lladós, and Thierry Brouard. Subgraph spotting through explicit graph embedding: An application to content spotting in graphic document images. In *2011 International Conference on Document Analysis and Recognition (ICDAR'11)*, pages 870–874. IEEE, 2011.

[81] Anjan Dutta, Josep Llados, and Umapada Pal. Symbol spotting in line drawings through graph paths hashing. In *International Conference on Document Analysis and Recognition (ICDAR'11)*, pages 982–986. IEEE, 2011.

[82] Anjan Dutta, Josep Lladós, and Umapada Pal. A bag-of-paths based serialized subgraph matching for symbol spotting in line drawings. *Pattern Recognition and Image Analysis*, pages 620–627, 2011.

[83] Anjan Dutta, Josep Lladós, and Umapada Pal. A symbol spotting approach in graphical documents by hashing serialized graphs. *Pattern Recognition*, 46(3):752–768, 2013.

[84] Marçal Rusiñol, Agnés Borràs, and Josep Lladós. Relational indexing of vectorial primitives for symbol spotting in line-drawing images. *Pattern Recognition Letters*, 31(3):188–201, 2010.

[85] Sounak Dey, Anjan Dutta, Josep Lladós, Alicia Fornés, and Umapada Pal. Shallow neural network model for hand-drawn symbol recognition in multi-writer scenario. In *Proceedings of the 14th IAPR International Conference on Document Analysis and Recognition (ICDAR'17)*, volume 2, pages 31–32. IEEE, 2017.

[86] Pau Riba, Anjan Dutta, Josep Lladós, and Alicia Fornés. Graph-based deep learning for graphics classification. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR'17)*, pages 29–30. IEEE, 2017.

[87] S. Ghosh, P. Shaw, N. Das, and K. C. Santosh. GSD-Net: Compact network for pixel-level graphical symbol detection. In *Int. Conf. Doc. Anal. Recognit. Workshops (ICDARW'19)*, volume 1, pages 68–73. IEEE, 2019.

[88] Zahra Ziran and Simone Marinai. Object detection in floor plan images. In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition (ANNPR'19)*, pages 383–394. Springer, 2018.

[89] Shreya Goyal, Vishesh Mistry, Chiranjoy Chattopadhyay, and Gaurav Bhatnagar. BRIDGE: Building Plan Repository for Image Description Generation, and Evaluation. In *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR'19)*, pages 1071–1076. IEEE, 2019.

[90] Ernest Valveny, Philippe Dosch, Adam Winstanley, Yu Zhou, Su Yang, Luo Yan, Liu Wenyin, Dave Elliman, Mathieu Delalandre, Eric Trupin, Sébastien Adam, and Jean-Marc Ogier. A general framework for the evaluation of symbol recognition methods. *International Journal on Document Analysis and Recognition*, 9(1):59–74, 2007.

[91] Mathieu Delalandre, Ernest Valveny, and Jean-Yves Ramel. Recent contributions on the sesyd dataset for performance evaluation of symbol spotting systems. 2011, (retrieved from semanticscholar.org, Dec. 2021).

[92] Marçal Rusiñol and Josep Lladós. A performance evaluation protocol for symbol spotting systems in terms of recognition and location indices. *International Journal on Document Analysis and Recognition*, 12(2):83–96, 2009.

[93] Mathieu Delalandre, Jean-Yves Ramel, and Nicolas Sidere. A semi-automatic groundtruthing framework for performance evaluation of symbol recognition and spotting systems. In *Graphics Recognition. New Trends and Challenges (GREC'11)*, pages 163–172. Springer, 2013.

[94] Mathieu Delalandre, Jean-Yves Ramel, Ernest Valveny, and Muhammad Muzzamil Luqman. A performance characterization algorithm for symbol localization. In *International Workshop on Graphics Recognition (GREC'09)*, pages 260–271. Springer, 2009.

[95] Michael Felsberg. Five years after the deep learning revolution of computer vision: State of the art methods for online image and video analysis. Linköping University Electronic Press, 2017.

[96] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, 2015.

[97] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'17)*, pages 7263–7271, 2017.

[98] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR'14)*, pages 580–587, 2014.

[99] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. "imagenet large scale visual recognition challenge". *International Journal of Computer Vision*, 115(3):211–252, 2015.

[100] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Proceedings of the European Conference on Computer Vision (ECCV'14)*, pages 740–755. Springer, 2014.

[101] Guillaume Renton, Pierre Héroux, Benoît Gaüzère, and Sébastien Adam. Graph neural network for symbol detection on document images. In *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW'09)*, volume 1, pages 62–67. IEEE, 2019.

[102] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[103] Zicheng Guo and Richard W Hall. Parallel thinning with two-subiteration algorithms. *Communications of the ACM*, 32(3):359–373, 1989.

[104] Mohammad Awrangjeb and Guojun Lu. Robust image corner detection based on the chord-to-point distance accumulation technique. *IEEE Transactions on Multimedia*, 10(6):1059–1072, 2008.

[105] Paul VC Hough. Machine analysis of bubble chamber pictures. In *Proc. of the International Conference on High Energy Accelerators and Instrumentation, Sept. 1959*, pages 554–556, 1959.

[106] Yonghong Xie and Qiang Ji. A new efficient ellipse detection method. In *Object recognition supported by user interaction for service robots*, volume 2, pages 957–960. IEEE, 2002.

[107] Ernest Valveny, Mathieu Delalandre, Romain Raveaux, and Bart Lamiroy. Report on the symbol recognition and spotting contest. In *Int. Workshop Graph. Recognit. (GREC'11)*, pages 198–207. Springer, 2011.

[108] Tapas Kanungo, Robert M. Haralick, Henry S. Baird, Werner Stuezle, and David Madigan. A statistical, nonparametric methodology for document degradation model validation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1209–1223, 2000.

[109] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single shot multibox detector. In *Proceedings of the European Conference on Computer Vision (ECCV'16)*, pages 21–37. Springer, 2016.

[110] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*, pages 779–788, 2016.

[111] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[112] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS'15)*, pages 91–99, 2015.

[113] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The PASCAL Visual Object Classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[114] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*, pages 770–778, 2016.

[115] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[116] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'17)*, pages 1251–1258, 2017.