An Improved System for Automatic Navigation among Architectural and Construction Documents

Author 1, Author 2

e-mail: [author@email.com](mailto:author@email.com)

Author 1, Author 2

e-mail: [author@email.com](mailto:author@email.com)

*Abstract*— Architectural and Construction industry deals in many multi dimension paper documents. Architects assign different symbols for referencing other documents for details of a particular section of a large drawing. The general contractor (GC) or the sub-contractor (SC) will go through these drawings and then shuffle through a pile of documents to navigate to the referenced sheet. The time taken will increase manifolds if the number of such references is very high. It becomes humanly arduous to navigate through a large pile of drawing documents where all the symbols refer to different sheets which in turn refers to another sheet and so on. We have proposed a method to automate these manual tasks with a good accuracy and efficiency.

Keywords—Architectural Documents; Automatic Navigation; Optimized Circle Detection; SVM; Hyperlinking; Relationship Graphs;

# Introduction

For any architectural project, many multi-dimensional drawing documents are made and printed on large sheets. A big project contains many such drawing documents and the number increases continuously as the project grows. Architectural drawings need utmost attention to details; so that contribution can be flawless. When the scale of the project is very large it is impossible to fit the whole project in a single drawing sheet, even if the size of the sheet is very big. In such cases, it becomes necessary to assign symbols to some section of a drawing and show these sections elaborately in a different sheet. This eases the task of the architect because it becomes easy to handle a very large project when it is broken into smaller pieces. Therefore, for a fairly large construction project it is convenient to sub-divide it into smaller sections so that it can be managed efficiently and understood easily by the architects involved in the project. But this increases burden for the ground workers of the project, because when a small section is shown in a separate sheet they have to refer these sheets from the complete project sheet and each referenced sheet might have other references in turn.

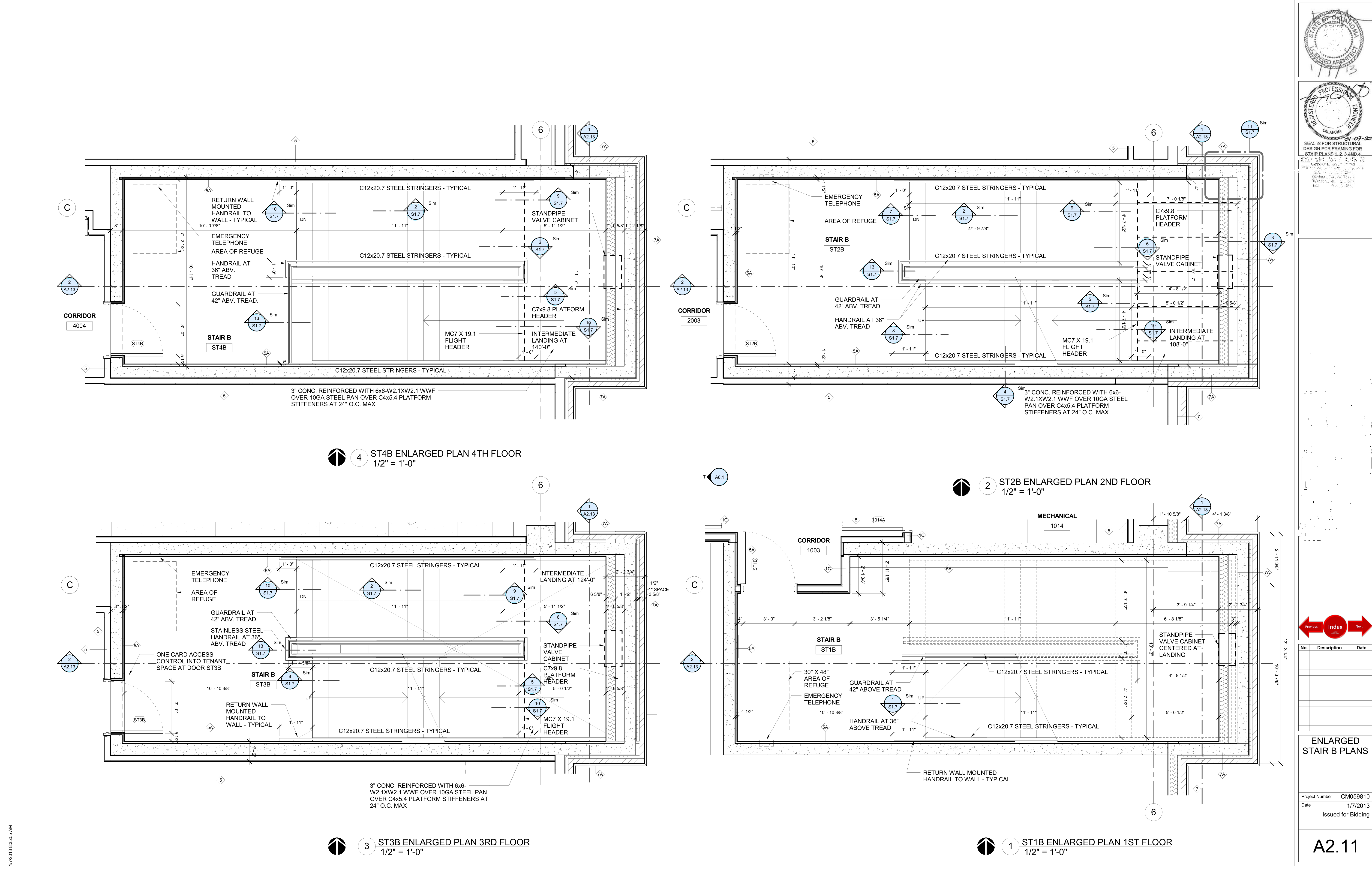


Fig. 1. A typical scanned architectural drawing document sheet with small circular symbols (Callouts) for reference sheet.

These large architectural drawings are made in CAD software and then printed on large drawing sheets. As the document are printed on the paper, any textual information that was there on the original CAD drawings will be lost and it will be difficult to search for any possible sheet numbers from these drawings. Since the drawing documents are in paper format and in the age of digitization architects also felt the need to preserve their sheets and hence they scanned the sheets as very large images using high utility architectural drawing scanners. Storing the documents in digital format help to preserve them for a longer period and easily accessible across different localities of the project owners. Fig. 1 represents a scaled down version of a typical scanned architectural drawing document. But scanning the documents doesn’t solve the problem of navigating from one document to another, rather it increases the difficulty for retrieving any information from the documents because of the additional noise that may be introduced due to scanned documents.

Navigating manually from one document to another by looking at the symbols is a cumbersome task and it is difficult to remember different connections of the relationship graph. If the drawing sheets are not properly indexed it will add more difficulty to the process. To solve the problem, a “Callout” detection based system is introduced whereby at certain place of the drawing there is a circular shape within which some text and numeric codes are printed, indicating that the user can go to another drawing sheet to explore more detail. In a previous paper [1], we proposed an automatic navigation software for such drawings which had many limitations. This paper takes care of those limitations, along with providing space for supporting new types of callouts. In this paper, we have used an optimized method for automatically detecting callout symbols. We have also changed our pre-processing pipeline for incorporating variety of callout shapes and having different textures.

# sections of engineering documents

There are different sections of an engineering drawing document. Some information contained inside an architectural sheet which is important for creating the navigations are as follows.

### Project Disciplines

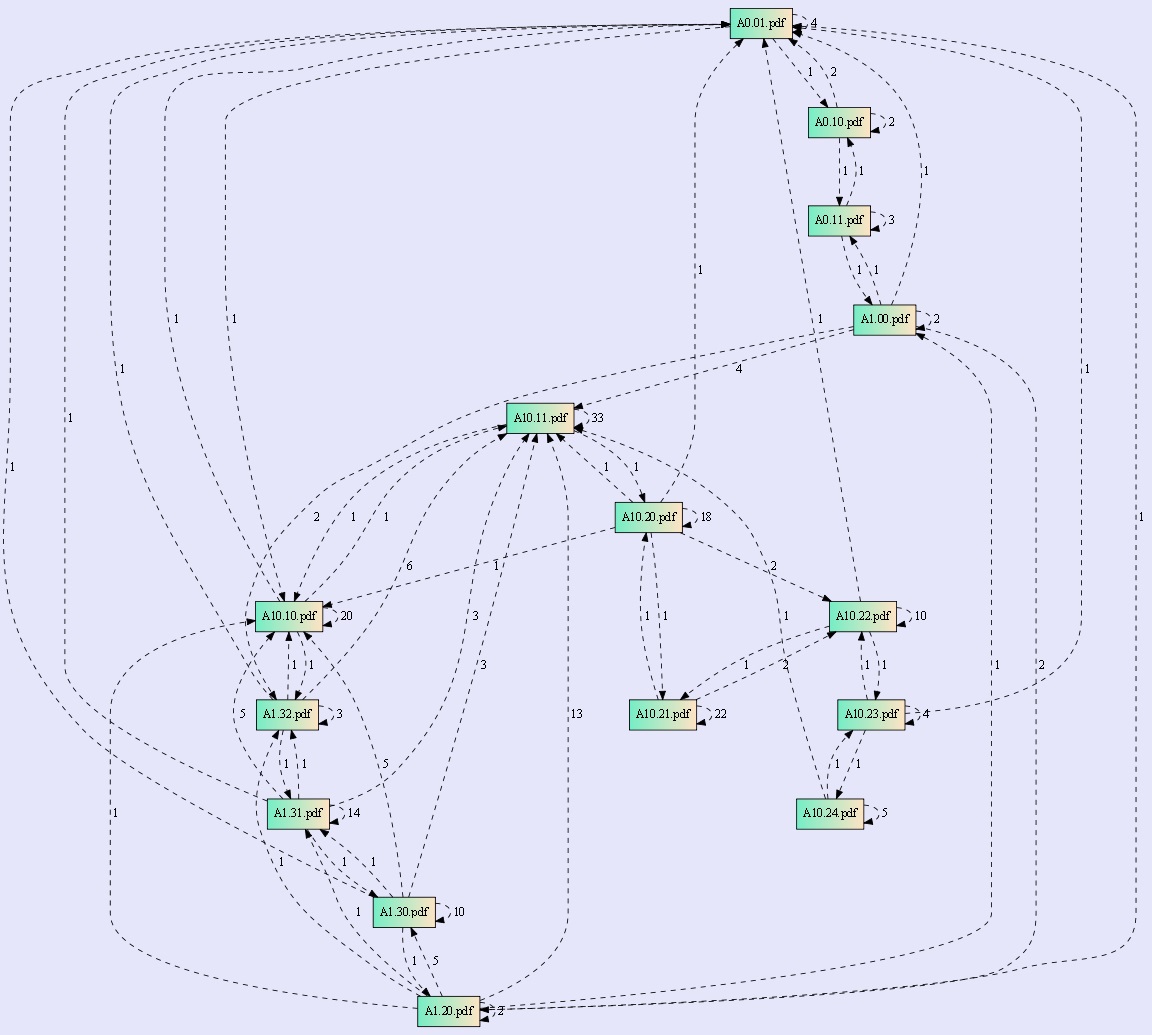


Fig. 2. A typical relationship graph for multiple sheets in a project

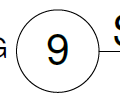
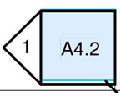
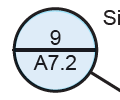
Architectural (Engineering) documents come from different disciplines (see Table 1), thus many people are involved in the process, which increases the number of documents further. People involved for these disciplines come from different technical background and may not have any synergy among them. For example, a civil drawing will be developed by an engineer or an architect but an electrical drawing will be made by an electrical engineer, now an engineer and an architect comes from a different technical background altogether hence their designs may differ drastically. However, these different types won’t cause much changes in the concept of referring to other sheets but the symbols used might change as per different architect or different discipline.

|  |  |  |
| --- | --- | --- |
| Architectural | General | Mechanical |
| Civil | Structural | Plumbing |
| Electrical | Telecommunication | Fire Alarm |
| Security | Fire Protection | Demolition |
| Telecom | Interior | Foodservice |
| Parking Controls | Landscape | Automation Control |
| Theatrical Lighting | Theatrical Rigging |  |

Table 1: Different disciplines of engineering drawing documents

### Sheet Number

Sheet numbers are the unique address of a document in a project. These sheet numbers can be found on the lower right corner in the most drawing documents [2], but this scenario might change as the architect changes. We can get the sheet numbers from these architectural drawing by a method known as indexing [3]. In indexing the sheet numbers are found by following some rules such as the initial character of a sheet number represents the discipline of the drawing, then the drawing sheet type and then the drawing sequence number [4]. These rules might change as per architects. Once the index of a drawing sheet is found it is easy to rename these drawing sheets as their index, which will be used later in creating hyperlinks for navigation from a source file to destination sheet. When all the sheets in a project are indexed then their names will be their respective sheet number and any reference sheet can be searched in the project with this sheet number information.



1. *(b) (c) (d)*

Fig. 3. Different segments of a drawing document necessary for hyperlinking (a) Callout shape containing Destination and Section text strings (b) Pentagonal callout shape (c) The location of Section string inside the drawing document. (d) Title block containing sheet number of the drawing document, normally present at lower right corner of an engineering drawing

### Connectivity Graph

The documents from different sections are related in some way or the other therefore they may be referred by different engineers while working on a project. All the drawings in a complete project form a directed graph which signifies the connection of different drawings inside the project. The nodes of the graph denote an architectural sheet and the edge from this node to a destination node denotes a connection between these sheets. The weight of the node denotes the number of connections from a source sheet to a destination sheet. Fig. 2 represents a sample relationship graph. Green rectangles in Fig. 2 represents different “sheet numbers” in a project and dashed curves are the representations of connections between source drawing and reference drawing inside a very large architectural project.

### Callout Shapes

The symbols which are used to refer a source sheet to a destination sheet or different sections within a sheet are called “*Callouts”* or *“Anchors”* in architectural terminology. A callout symbol is divided into two halves by a straight line passing through middle of the callout. The straight line can have any orientation but the ideal orientation of a callout is horizontal. When the callout is horizontal, the upper half of the callout contains a code which represents a “*Section Number*” and the bottom half represents the “*Destination Sheet Number*”. *Fig. 3(a)* and *3(b)* represent a callout shape. *Destination Sheet Number* is the sheet number or sheet identifier of the referenced sheet and *Section Number* is the location within destination sheet being referred to. *Fig. 3(c)* represents a destination location containing a circle with text printed inside it, this text is same as the section text of the callout. This text information is very important in forming the relationship graph of different documents in the entire project.

No other prior work has been done in this domain except our previous work which also dealt with only circular shapes and failed in case of highly noisy images. There are few researches in information retrieval from the engineering drawing documents for indexing the documents, version control etc. L. Najman in his research has worked on locating the title block in engineering drawings [3]. R. Sulaiman has worked on extracting information which is usually the most common structure used to construct a title block [2]. Thus, by extracting information as sheet number or drawing number one can easily index the document. Some research has been done in architectural document domain to separate out text and graphics components [5-8] which are mainly the measurement text along with other information. But no prior work is present in creating Automatic navigations for solving the critical issue of document management. Therefore, as per our knowledge this work is pioneer in this domain.

This paper is arranged into three parts, in first part, we revamped the method used for locating the anchor and destination regions in our previous research along with improved pre-processing for making it possible for OCR engine to easily extract the alphanumeric code from the shape. In second part, we have discussed on recognition of the alphanumeric code printed inside the regions found in the first step using Optical Character Recognition. In third part, we have formed a relationship graph using the alphanumeric code information found in earlier stages. This connection graph is used in order to link different documents to one-another at these locations.

# Process

We receive these documents in PDF format and for processing we convert these PDF files into PNG image files. As the size of these drawings is very large, the processing is very time consuming. Therefore, we performed all pixel level calculations on a lower resolution image and translated these calculations to higher resolution for OCR.

The method of navigation creation is divided into four parts (1) Locate the callout/anchor regions (2) Text-Graphics separation for extracting the section and destination text from the callout in order to increase the OCR accuracy (3) Recognition of alphanumeric code to link the appropriate document pairs (4) Hyperlinking the documents based on the extracted information. In this paper, we decided to solve this issue by increasing support for more number of shapes, also increasing the shape detection speed using Optimized Hough Transform.

## **Callout/Anchor Shape Localization**

As found in the Fig. 3 (a), (b), we have two kind of callout shapes, only thing common between them is that both have holes in them. We followed a two-stage approach for callout shape detection. In the first stage, we reject all the components which are not callout candidates in order to reduce the computation of subsequent steps for all the components. For this we detect all the components having holes in that connected component. The second stage is further divided into two parts, first part employs an efficient and optimized circle detection algorithm to detect near circular shapes and second part detects pentagonal shapes. Candidate removal is an important step to reduce the number of computations.

### Hole Detection

The main motivation behind this step is to reduce the search space for callout shape detection, because the shape that is necessary for getting the hyperlink information is having a hole in it.

We start by computing the contours of the whole image and reject any possible straight lines and other small and very large components. After that we arrange these contours in a two-level hierarchy, at top level is the parent contour at the second level is the child contour. Parent contour is the external boundary of the component. Child contour is the boundary of the hole. If there is another contour inside a child contour, we still put it at the top level. Each contour represents as a curve/polygon. Once we have recognized the components which contains holes, we only consider those components in the image and discard the rest. This way, we have discarded most graphics components including linear components which doesn’t have any hole in them and take the remaining components for further processing.

### Shape Detection

Once the Connected components having holes are identified we make a fresh image from these components discarding the rest. After that we employ a two stage method for identifying circular shapes and pentagonal shapes.

1. Circular Shape Detection

Circular shapes can be found by several methods such as Hough Transform [9], Randomized Hough Transform [10] etc. We used an optimized random point selection and circle equation validation logic and calculating the number of votes that a radius and centre would get for these random points. Since the shapes are not perfect circles we need to modify different parameters to fit to our needs. Before we start the circle detection process we convert the image into edge image using Canny edge detection. Edge image will convert all the components into their boundary components thus reducing the number of foreground pixels significantly in a graphics rich image.

In two-dimensional space, circle can be represented as *eq. 1*, where ***a****,* ***b*** are the center of the circle and ***r***is the radius. Most of the architects follow same standards for making the *Callouts* and therefore the circles are almost similar in size. Knowing the fact that the circles are of similar size we can fix the radius range and iterate through this range to get the circles.

(1)

(2)

let us consider **()**, **()**, **()** are three non-collinear points on the circle. Now the radius can be obtained by equation 2 for any from the selected 3 points. Circle radius and centre can be obtained by following steps:

Step-1:

Store all the edge pixels ***,*** to the list ***P*** and divide the ***P*** into four lists depending on their axis quadrants so that picking a pixel for each quadrant increases the probability of edge pixels of the image are on the circle and non-collinearity. Initialize the fail counter variable ***f*** to 0. Let , , , **,** and be the five given thresholds. Where denotes the number of failures that we can allow for circle detection. If there are less than pixels in ***P***, we stop the task of circle detection. The distance between any two chosen edge pixels of the possible circle should be greater than , and are the distance thresholds and ratio thresholds, respectively.

Step-2:

Check for ***f*** = if satisfies stop the execution and return null parameters else randomly pick and remove one pixel from each four different quadrant pixels lists.

Step-3:

Find the circle which is possible with picked edge pixels and where distance between any two of three chosen pixels is greater than and the distance between the fourth pixel and the boundary of the obtained circle is greater than **;**  go to Step-4. Otherwise put back removed pixels to its corresponding lists and increment fail count ***f***; go to Step-2.

Step-4:

Assume ***C*** is the obtained possible circle. Initialize counter ***c*** to 0. For each pixel in list ***P*** check whether the difference between distance of chosen pixel to center of ***C*** and radius ***r*** of ***C*** is less than . If yes increment ***c***.

Step-5:

If ***c* >= 2π**go to Step 6. Otherwise, regard the component as not a circle, perform increment ***f***; and go to Step-2.

Step-6:

The possible circle ***C*** has been detected as a true circle. So, add this to detected circular shape list. Also, we will remove this component from the remaining components for detecting the pentagonal shape, since we have found that the given component is a circular shape.

1. Pentagonal Shape Detection

As the *Callout* shape is not always a circular shape so we have to detect pentagonal shapes also. Pentagonal shape can be found from a generalized polygon detection algorithm.

We first collect all the connected components (CCs) that are not part of the detected circular shapes. Then pentagonal shapes can be found in below steps:

Step-1:

We get all the curves from the image and approximate each curve using Douglas-Peucker algorithm [11] for a possible polygon. We add all the curve pixels ***,*** to the list ***V***. This list will contain an ordered set of points and lines and the distance dimension **ε.** The algorithm recursively divides the line between two points **.** Initially it is given all the points between the first and last point . We also maintain a list for all the points to be kept in the final approximate curve. We automatically include first and last point in the list **.** It then finds the point **,** that is furthest from the line segment with the first and last points as end points; this point is obviously furthest on the curve from the approximating line segment between the end points. If the point is closer than ε to the line segment, then any points not currently marked to be kept can be discarded without the simplified curve being worse than ε. If the point furthest from the line segment is greater than ε from the line segment, then that point must be kept. The algorithm recursively calls itself with the first point and the worst point and then with the worst point and the last point, which includes marking the worst point being marked as kept.

When the recursion is completed a new curve can be generated consisting of all and only those points that have been marked as kept.

Step-2:

Once we have approximated all the curves, we then try to mark two polygons (Rectangle and Triangles) present in the image. As shown in the Fig. 3(b), we can see that *section number* and *destination sheet number* are divided in two portions, section number is present inside a triangular shape and destination sheet number is present inside a rectangular shape. Therefore, we only consider triangle and rectangle shapes. For triangle shape, we check if the approximated polygon is having only three sides and the angle is between range [ **-**  **,** here is the lower angle threshold and is the upper angle threshold**.** If this condition is satisfied, then we add this curve to the list of triangles obtained. Similarly, for rectangle the number of sides must be 4 and the angles of the shape must be in the range [ **-**  .

Step-3:

After step 1 and step 2 we have got two lists containing triangles and rectangles. Now, we have to find pairs of triangle and Rectangle **,** such that combination of both of them can provide us a proper callout shape. For this, we check for overlapping edge between triangle and rectangle . Also, the line joining the centroid of rectangle and centroid of triangle must be perpendicular to the common edge . If this condition satisfies then we term the combination of these shapes as a valid polygon callout.

Table 2 lists the accuracy for circle and pentagon detection when tested across 1000 files.

|  |  |  |
| --- | --- | --- |
| Testing Files | Number of Pentagon Callouts | Pentagon Detection Accuracy |
| **1000** | **1187** | **88.1%** |

Table 2(a): Pentagonal Callout Detection Accuracy

|  |  |  |
| --- | --- | --- |
| Testing Files | Number of Circular Callouts | Circle Detection Accuracy |
| **1000** | **2169** | **97.7%** |

Table 2(b): Circular Callout Detection Accuracy

## **Text-Graphics Separation**

Most important step in extracting the alphanumeric code from the callout is removing the unnecessary sections from the callout image, including the small noise, straight lines, colour annotations, unnecessary symbols etc. We have followed a cascaded approach for removing these noises and graphics because all the images don’t possess similar noise patterns. We have done a basic classification of callouts based on certain rules. If number of small connected components (area < ) are greater than some threshold , we classify these images into textured images shown in Fig. 4 *(a)*, *(e).* If the height of the largest connected component is less than percent of the total height of the image, then we classify this image as Fig 4 *(q)*. If the black pixel density of the image is larger than some threshold we classify this image as one of the image shown in Fig. 4 *(b)*. If the number of channels present in image is not equal to 1 then we classify the image as shown in Fig. 4 *(j)*. We undertook the following steps for graphics and noise removal from the image.

1. *(b) (c) (d) (e)*

*(f) (g) (h) (i) (j)*

*(k) (l) (m) (n) (o)*

*(p)*  *(q) (r) (s) (t)*

Fig. 4. (a) – (l) Callout images with texture, (m)-(t) different kind of callout images found in architectural documents

### Color component removal

Most of the callout images that we receive contains the alphanumeric code in black colored font, but in some cases because of the annotations created on the original file we come across different color components coming inside the callout image. We removed any such regions from the callout images based on similar color flood filling. In this kind of flood fill we don’t remove the black pixels instead we apply the flood fill on colored components and later discard these color components.

### Adaptive Binarization

In some cases, the fixed binarization doesn’t work so we have employed a two-level adaptive binarization in which we pass the image through two levels of fixed binarization thresholds and then based on the black pixel density difference we fix a binarization for removing gradients of gray values.

### Callout Orientation Correction

For cases shown in Fig. 4 *(t)*, the callout is rotated 90 degrees, and we need to correct the orientation in order to recognize the alphanumeric code correctly. For orientation correction, we first detect straight lines in the given image and for the straight line which is closest to the centre of the image we get the angle of the line. Now we check black pixel density in the upper and lower portion of the line, and we rotate the image keeping higher black pixel density on the lower side of the image. More black pixel density signifies more number of characters and usually destination sheet number has more number of characters than section number.

### Linear Component Removal

For cases shown in Fig. 4 (d), (e), (f), (h), (i), (k) etc. we note that there are many straight components with a random angle, which should be cleaned. In cases, like Fig. 4 (i) the line passes through a character and we need to take care when removing the lines, because it may remove a portion of the character. We first detect the straight lines using Hough Line Transform (ref). Then delete these lines from the image also, copy these lines in a mask image for retrieving lost character potions. Once all the lines have been deleted from the image we do *Image Inpainting* [12] using the line images as the mask.

### Texture Removal

For images as shown in Fig 4. (a),(c),(d) etc. there is a need to remove the texture in order to process the image correctly. We have classified these textures into 3 categories, small salt-pepper texture, brick like random line texture and colour texture. For removing the small noisy texture, we removed it based on some threshold for component height and width. For removing the brick texture, we took a template of single brick and tried to find this structure in the image. We subtracted the texture region from the original image. For third kind of texture we converted the image to gray scale and then based on adaptive binarization we removed this texture.

### Touching Character Separation

For cases shown in Fig 4. (r), (s) (b) etc. we can see that the alphanumeric code is touching the circle, therefore we need to remove the circle in such a way that it doesn’t hamper the alphanumeric code. We Categorized these cases into two categories, first is the images like Fig 4. (r) where the characters are touching with circle boundary but all the characters lie inside the circle. Second category is the images like Fig. 4 (s), where the characters are overflowing the circle. We do horizontal scanning of the image and check for average thickness of the circle. For categorization, we check if there is variation in circle thickness as we move in the rows of the image. For overflow case the change in thickness is very high compared to the touching character case. Based on this assumption we break the connection between a character and the circle. For cases shown in Fig 4(q), we have to first detect the position where the circle is broken and after that we took horizontal scans from these breakage points to obtain the alphanumeric code.

### Removing Small and Large Components

Once we have removed the circle and other linear components we try to remove the very small noise and very large components from the image. We compute the Connected Components (CCs) from the image and then based on below mentioned rules we remove the components.

- If the height or width of the connected components is greater than some threshold and smaller than some threshold , then we reject this component

- If the density of the component is smaller than some threshold and larger than some threshold ,then we reject the component

- If the ratio is greater than some threshold and smaller than some threshold , then we reject the component. Here, is the density of the outer edge of the character and is the original component density.

- If the ratio *height*/*width* of the components is greater than some threshold and smaller than some threshold then we reject the component.

### Lost Component Retrieval

In the process of rejecting small and large components we end up losing characters like dot (“.”), hyphen (“-”), pipe (“|”), comma (“,”), i, l etc. For retrieving the lost components, we check for their position in the original image with respect to the remaining component image. For (“-”) the position should be between the lower and upper limit of the line. Similarly, for (“.”) the location should be in the lower half of the line.

Fig 5 shows the outputs from different processes on the callout images.



*5.1(a) 5.1 (b) 5.1 (c) 5.1 (d) 5.1 (e)*



*5.2(a) 5.2 (b) 5.2 (c) 5.2 (d) 5.2 (e)*



*5.3(a) 5.3(b) 5.3 (c) 5.3 (d) 5.3(e)*

*5.4(a) 5.4 (b) 5.4 (c) 5.4 (d) 5.4 (e)*

Fig. 5. Pre-processing and output after each step

## **Alphanumeric Code Recognition (OCR)**

After the text components are separated out in text-detection step we need to recognize these characters in order to proceed to hyperlink creation step. This is very important and compulsory step for hyperlink creation. Our OCR engine was built using Support Vector Machine (SVM) classifier.

We build an in-house optical character recognition (OCR) engine using support vector machine classifier. An important step for accurate classification is selection of proper features. The features can be of two types– local and global. Local features involve windowing the image whereas global feature takes some characteristics of the whole image. The image of the character is divided into 5 × 5 blocks for local features. The features include the directional components of the border pixels in 4 directions. Thus 5x5x4=100 features are obtained. Some global features like aspect ratio, longest vertical run, normalized height of the leftmost, rightmost and lowermost black pixel, Euler number etc. are also used. Thus, a total of 108 features are used for classification in the OCR system developed at ARC Document Solutions in collaboration with Indian Statistical Institute (ISI) [13].

## **Hyperlink Creation**

Once we have got the characters which are printed inside the Callout shape we can get the information from the local files matching these alphanumeric codes in order to create a connection graph. A connection graph is a relationship graph which shows the link between two documents and how they are connected.

The information which we found from the Callout shape is basically the sheet number of the destination sheet. Every engineering document contains a sheet number information which gives the index of the sheet in the project. This information is very important in indexing the engineering drawing documents. The sheet number information is provided at a specific location inside the engineering document this is known as “*Title block*”. Fig 2(b) represents an example of a title block, having sheet number **A2.11**. Thus, once we have found this indexing information from the file then we index all the drawing documents according to their sheet numbers. Now the alphanumeric code found from the Callout shape also represents a sheet number, i.e. the sheet number written inside the Callout shape is the sheet number of the document which is connected with this particular location.

For example, after the indexing of drawing documents according to their sheet number, we take a document **A2.11** and the Callout shape from this same image contains a code **S-502**, therefore there is a connection from drawing A2.11 to drawing S-502. *Fig. 8* represents a sample connection graph.

After this connection graph is formed we can simply burn this information into the original document image. And we will end-up with clickable links which will enable the navigation among several plan documents within the complete project.

# Experimental Results

This section presents some of the results and accuracy of the system. A separate paragraph defines the different type of test images used and how we collected the data.

## Data Collection

The main challenge of any image processing project is to collect appropriate data. In this problem also, we needed to collect data which contains structure that we were going to recognize. Almost all the engineering drawing documents of a large project, contains this structure and they navigate by creating hyperlinks at this structure, based on the information printed inside the structure. We collected around 980 different drawing documents including few with no such structure in them. This was done in order to test the generic nature and robustness of the algorithm.

We used the drawing documents which had circles containing information about the same document and information about some other document at the same time to test the hyperlink creation.

## Results and Accuracy

We conducted these tests for around 980 different drawing documents with several circles presents inside them. Some of the circles were very complicated therefore it was very difficult to extract the text from them. The accuracy of circle detection came out to be around 97.7% as is shown in *Table 2(b)*. But the circle shapes are too much noisy and we needed to get rid of the noise in order to expect best results from the OCR engine. In pre-processing step for the OCR engine, in some cases we lost the Section and Destination information, but the overall callout shape retention in text-graphics separation step was very decent with 97.47% circles retained with text characters were extracted from the image successfully. The next step was to recognize the text obtained from the text-graphics separations step. The OCR accuracy from the system developed to recognize 52 characters, 10 digits and few symbols including dot (.) and dash (-), came out to be around 98.86%. The whole end to end automatic hyperlinking system accuracy came out to be nearly 94.46%.

# conclusion

# Acknowledgment

This research project took place at ARC Document Solutions, Kolkata, and the researchers had the full cooperation of ARC Document Solutions and Indian Statistical Institute (ISI) Kolkata.

##### References

1. P. Banerjee, S. Choudhary, S. Das, H. Majumder, B. B. Chauduri, “Automatic Hyperlinking of Engineering Drawing Documents”, Proc. 12th IAPR Workshop Document Analysis System, April 2016
2. Riza Sulaiman, M. F. M. Amran and N. A. Abd Majid, “A Study on Information Extraction Method of Engineering Drawing Tables,” Universiti Kebangsaan, Malaysia, vol. 50, International Journal of Computer Applications (0975-8887), July, 2012.
3. Laurent Najman, Olivier Gibot and Mickaël Barbey, “Indexing Technical Drawings using Title Block Structure Recognition,” Océ Print Logic Technologies SA, Document Analysis and Recognition, vol. A247, pp. 587–591, 2001.
4. https://www.archtoolbox.com/representation/graphic-symbols/condocsh

eetorder.html

1. Mohd. Fahmi Mohamad Amran, Riza Sulaiman, Suziyanti Marjudi, Saliyah Kahar and Zuraidy Adnan, “Extraction Method of Version Number in Engineering Drawing, ” Applied Mechanics and Materials vol. 110-116, pp. 4856-4860, 2012.
2. K. Tombre, S. Tabbone, L. Pé,lissier, B. Lamiroy and P. Dosch, "Text/Graphics Separation Revisited," Proc. Fifth IAPR Int',l Workshop Document Analysis Systems, Aug. 2002.
3. Cao, Y. et al, "Using Engineering Drawing Interpretation for Automatic Detection of Version Information in CADD Engineering Drawing". Automation in Construction 14, pp. 361-367, 2005.
4. Ab. Jabal, M. F. et al, "A Comparative Study on Extraction and Recognition Method of CAD Data from CAD Drawings". International Conference on Information Management and Engineering, pp. 709-913, 2009.
5. J. Illingworth and J. Kittler, “The Adaptive Hough Transform,” PAMI-9, Issue: 5, 1987, pp 690-698
6. Xu L, Oja E. Randomized Hough transform. In: Ramón J, Dopico R, Dorado J, Pazos A, eds. Encyclopedia of Artificial Intelligence. Hershey(PA): IGI Global, 2008, 1354–1361
7. Prasad, Dilip K.; Leung, Maylor K.H.; Quek, Chai; Cho, Siu-Yeung (2012). "A novel framework for making dominant point detection methods non-parametric". Image and Vision Computing. 30 (11): 843–859
8. M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In Proc. ACM Conf. Comp. Graphics (SIGGRAPH), pages 417–424, New Orleans, LU, July 2000.
9. P. Banerjee and B. B. Chaudhuri, “An Approach for Bangla and Devanagari Video Text Recognition,” Proc. International Workshop on Multilingual OCR (MOCR), Washington DC, USA, article no. 8, 2013.
10. M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester,  "Image inpainting",  *Proc. ACM Conf. Comp. Graphics (SIGGRAPH)*,  pp.417 -424, 2000
11. Lianyuan Jiang, Peihe Tang, Haohao Yuan, Chungui Li, Yalan Zhang,

“Fast Randomized Algorithm for Circle Detection by Efficient Sampling”

Journal of Theoretical and Applied Information Technology 20th February 2013. Vol. 48 No.2.