Vector-Based Segmentation of Text Connected to Graphics in Engineering Drawings

Dov Dori Liu Wenyin

Faculty of Industrial Engineering and Management
Technion-Israel Institute of Technology, Haifa 32000, Israel
{dori,liuwy}@ie.technion.ac.il

Abstract: A method for segmentation of text that may be connected to graphics in engineering drawings is presented. It consists of three steps: growing individual characterbox regions, using a recursive merging scheme by stroke linking; merging the detected characterboxes into a textbox and determining its orientation; and re-segmenting the textbox back into the refined characterbox that can be input to an OCR subsystem. The method can segment dimensioning text as well as other classes of text. It handles both isolated and touching characters, aligned at any slant. The capability of segmenting characters that touch either themselves or graphics, which is an important feature in handling real life drawings, is obtained by focusing on intermediate vector information rather that on the raw pixel data. We present the details of the algorithm and show both successful and unsuccessful examples from an experimental set of 36 dimensioning textboxes, in which 94% segmentation rate was achieved with 3% false alarm rate.

Keywords: Text Segmentation, Engineering Drawing Interpretation, Primitive Recognition

1 Introduction

In spite of the current common use of Computer Aided Design (CAD) systems to produce and manage engineering drawings, a CAD conversion system capable of understanding paper based engineering drawings and translating them into CAD representation is highly demanded [1]. As one of the two rather different classes of primitives in engineering drawings, text needs to be processed separately from graphics. Hence, segmentation of text from graphics, followed by text recognition, is a basic step in text processing of engineering drawings, which, in turn, is an important part of engineering drawings interpretation.

Due to the complexity of text patterns in engineering drawings, there is no single text segmentation method which works effectively on real life paper based drawings. Most current systems can only cope with high quality drawings, where the text characters are detached both from the graphics and from each other. Fletcher and Kasturi [2] developed an algorithm for text string separation from mixed text/graphics image. It is based on the generation of connected components and the application of Hough Transform to group together the components into logical character strings, which may then be separated from the text. Lai and Kasturi [3] presented a system for detecting dimension-sets in engineering drawings that follow the ANSI drafting standard. It is also based on the generation of components and composing them into text strings, which are associated with dimension lines. Neither one of these algorithms treat the problem of text/graphics connectivity, nor do they detect text strings consisting of a single character. Chai and Dori [4] proposed an algorithm for textbox extraction, that is preceded by OZZ vectorization, arc segmentation and arrowhead recognition [5]. The textbox extraction is done by clustering short bars that are close to each other through a region growing process. As noted by the authors themselves, the algorithm is designed only for detection of text areas without string extraction, and may lead to some detection errors. On the basis of [4], Dori and Velkovich [6] proposed a higher level text segmentation and recognition method. The improvement is in that the algorithm handles the connectivity problem. It groups the basic textboxes into basic and logical textbox and extracts the basic textboxes for string separation and OCR. These methods depend to some extent on the presence of other primitives, such as arrowheads, which must be correctly recognized from the drawings prior to text segmentation. Since most methods work with rather ideal drawings, the rate of correct text segmentation for complex and noisy real life drawings is usually not satisfactory.

In this paper, a robust method for text segmentation is developed and implemented. The method does not refer to the pixel data of the image, neither does it depend on arrowheads as evidence of the existence of text. It depends only on bars and polylines, which are basic primitives extracted by the vectorization process. Dimensioning text and other kinds of text with various orientations, including special symbols, such as surface quality and other manufacturing instruction symbols, are detected by this method.

2 The Text Segmentation Algorithm

2.1 Problem Definition

The following definitions, demonstrated in Fig. 1, are used throughout our work.

Stroke: wire, the basic vector extracted from the binary character image by vectorization. It may be a bar, which is a straight line segment with non-zero line width, a polyline, which is a chain of bars, or a circular arc with non-zero line width.

Charbox: characterbox, the bounding rectangle of the character image, i.e., a minimal rectangle enclosing the strokes of this character, without any non-text element or stroke of any other character.

Coarse Charbox: an upright (straight) charbox that bounds the character (which may be slanted) as if it were horizontal or vertical.

Normal Charbox: an upright charbox that tightly bounds the character in its normal orientation, i.e., after it was rotated to a horizontal position.

Fine (Tight) Charbox: a charbox that tightly bounds the character, whose slant is the same as the orientation of the character. The fine charbox is calculated by finding its normal charbox and rotating the normal charbox back to its original orientation.

Text: a string of any kind of characters and symbols.

Textbox: a minimal rectangle, enclosing a particular text without any non-text element, and any part of other text. The textbox of a text is the union of the charboxes of its character components.

Coarse Textbox: an upright rectangle formed by the union of the coarse charboxes of its constituent characters.

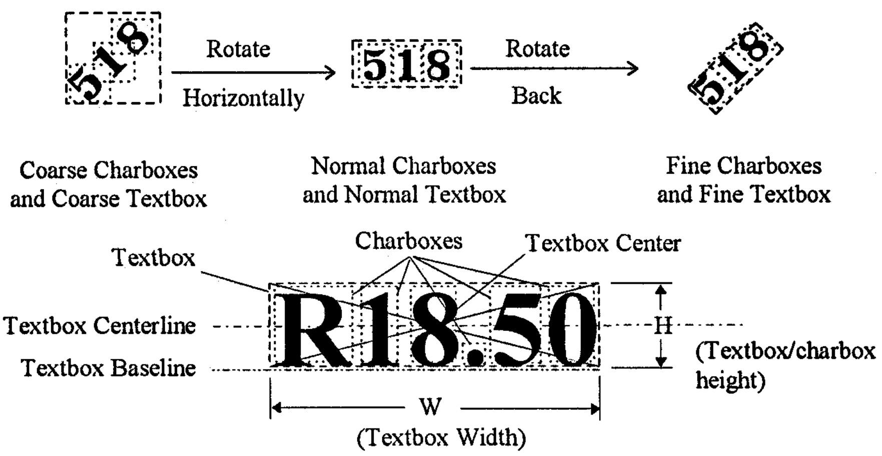


Figure 1. Textbox example and related concepts.

Normal Textbox: an upright rectangle that is formed by the union of the normal charboxes of its constituent characters and tightly bounds the character string in its horizontally rotated image.

Fine (Tight) Textbox: an oriented textbox that tightly bounds the text, whose slant coincides with the orientation of the character string.

Textbox Center: the intersection of the two diagonal lines of the textbox.

Textbox Baseline: a straight line passing through the basis of the textbox.

Textbox Centerline: a straight line passing through the textbox center.

Basic textbox [6]: a textbox, in which the text is a single string.

Logical textbox [6]: a textbox, in which all elements are logically connected and refer to a common element of an engineering drawing. A logical textbox is usually used for the dimensioning text, which consists of a basic textbox as the nominal dimension and optionally one or two basic textboxes as its tolerance (e.g., 5±0.5).

Using these definitions, the problem of text segmentation in engineering drawings can be decomposed into three main steps of work. Fig. 2 is a top-level Object-Process Diagram (OPD) representation [8] of this subsystem. These three steps are shown in Fig. 3, which is the blow-up and unfolding of Fig. 2. The first step is recursive merging–segmentation of charboxes from the graphics, which is done as a specialization of the unified generic object recognition methodology [9]. In this step, we find coarse charboxes–upright bounding rectangles of isolated characters. In the second step–grouping, we perform statistics on the size of individual characters, which is used to combine adjacent charboxes into textboxes with determined orientation. Finally, in the third step–refinement, the textboxes are re-segmented into precise individual charboxes, which can be used as input to OCR.

Our text segmentation takes place immediately following vectorization [7], which results in strokes: bars and polylines. We do not work with the pixel data, neither do we need to depend on detected arrowheads as evidence for the existence of text, although such information may be helpful when searching for missed text.

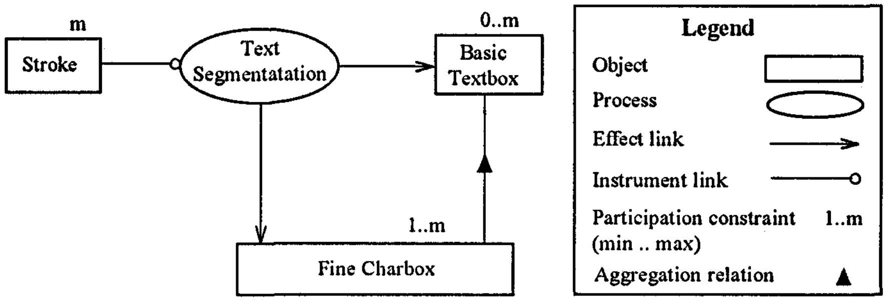


Figure 2. Top level OPD of the Text Segmentation Algorithm.

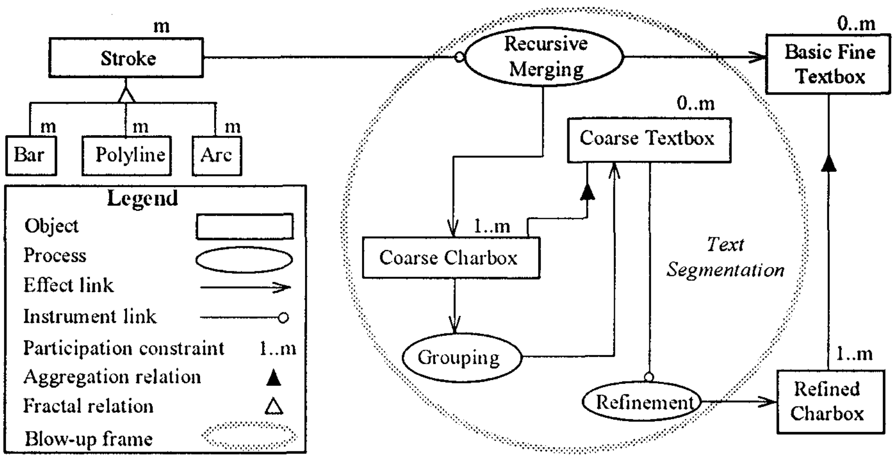


Figure 3. Blow-up of Text Segmentation and unfolding of stroke of Figure 2.

2.2 The Initial Charbox Segmentation

Based on the features of characters, we start to find each charbox by finding the first key component as a stroke, whose length is less than both a predefined maximal character size and *n* times the stroke width, where typically n=10. A recursive algorithm is then used to realize the merging procedure for growing the charbox region by collecting strokes that are close enough and meet the above two maximal length requirements.

The recursive merging algorithm for charbox growing works in a depth-first fashion. The charbox of the current character is initialized with the bounding rectangle of the first stroke—the found key component. The algorithm accepts this stroke (and the rectangle) as its input parameter, and returns a complete charbox. Pseudo-code of the recursive charbox merging algorithm is listed in Fig. 4.

The algorithm merges the areas covered by linked strokes. Its main purpose is finding the minimal bounding boxes of single isolated characters, but it may also return coarse charboxes resulting from two or more touching characters. Merging graphic elements with characters that touch the graphics rarely occurs, because graphic objects are usually longer than character strokes, so they are precluded from being stroke candidates in the first place. In case a non-text element is contained in the charbox, it may be carried on to the OCR stage and prevent correct recognition.

```
Algorithm 1: Recursive Coarse Charbox Merging
Rectangle Recursive_Charbox( Stroke S )
Begin
  For  each stroke close to Stroke S and is unmarked
  Begin // finding the stoke list SL
    If the stroke is shorter than max_char_height
       and shorter than n times its width
    then put it into Stroke list SL;
    Else return NULL;
  End;
  For  each stroke s in SL
  Begin // merging strokes in SL
    Mark stroke s;
    Rectangle r = Recursive_Charbox(s);
    If (r = NULL) return NULL;
    Else CBOX = Union of CBOX and r;
    If CBOX > MaximumCharbox return NULL;
  End;
  return CBOX;
End.
```

Figure 4. Pseudo-code of Algorithm 1–Recursive Coarse Charbox Merging.

2.3 Grouping Charboxes into Textboxes

As shown in the OPD of Fig. 3, the single coarse charboxes extracted by Algorithm 1 are the input for the grouping process. According to ISO and ANSI drawing standards, the height of the character is usually 1.5 times its width. The character size can therefore be expressed by its area. The coarse charboxes are first used to compute the average area (height by width in pixel units) of the charbox. Let S_i be the area of the ith charbox and N the total number of charboxes, then the average charbox area S_c is

$$S_c = \left(\sum_{i=1}^{N} S_i \right) / N$$

Let W_c be the average charbox width and H_c the average charbox height (see Fig. 1). Since $H_c = 1.5W_c$, we get that

$$W_c = \sqrt{S_c} / 1.5$$

Having found the average character width and height, neighboring charboxes are grouped into textboxes, as shown by the grouping process in the OPD of Fig. 3. The sizes of the individual charboxes are used to determine the search directions. For example, if the height of the coarse charbox is less than its width, the character is likely to be more vertical than horizontal, so the top and bottom, rather than the left and right areas of the character, should be searched for additional character components of the textbox to which the character inside the coarse charbox belongs.

The orientation of a textbox is roughly determined first by the centers of its component characters. This orientation is then tested and refined by finding a bar beneath/under (in the sense of the roughly determined orientation) this textbox. If such a bar is found, it is likely to be the tail of the leader of the corresponding dimension-set, and this text should have a center line parallel to this bar. In that case, the textbox orientation is set to be identical to that of the bar tail. If such a bar cannot be found, the orientation of the textbox is determined by the ratio of its height to its width when the textbox is horizontal or vertical, or by the coarse textbox orientation otherwise. The reason is that, for a single character the height is bigger than the width, and for a character string consisting of more than one characters the height is smaller than the width. If there is only one charbox in the textbox and the height is bigger than the width, or, there are more than one charboxes in this textbox and the width is bigger than the height, the coarse orientation is taken as the refined, final orientation. Otherwise, the refined orientation is orthogonal to the coarse orientation.

The charbox average size and the textbox orientation are useful in the refinement process for detecting characters touching other characters or graphic objects and lines. The orientation is used to detect graphics-touching characters, which were overlooked in the first step. The average size is used to filter out these graphics-touching characters. For example, new charboxes which are too large are excluded as non-characters. The average size is also used to divide charboxes that are larger than a certain threshold into two or more individual charboxes.

2.4 Charbox Refinement

As the OPD in Fig. 3 shows, the final step of our text segmentation algorithm is refinement. The textbox is re-segmented into more accurate, refined charboxes. This step concurrently solves the following four problems: (1) finding the individual fine (tight) charboxes and the fine textbox according to the textbox orientation; (2) cutting merged charboxes, which may contain more than one individual character, into two or more correct and fine charboxes; (3) separating graphics-touching characters, which may belong to the textbox under consideration, from the graphics, and adding them to that textbox; and, (4) arranging the collection of charboxes in this textbox in the correct reading order from left to right, for correct OCR results. The refinement proceeds in the following way.

Initially, the normal textbox and its normal charbox list are empty. Every coarse charbox in the coarse charbox list of the coarse textbox undergoes the following operations. First, we rotate the stokes of the currently taken coarse charbox around the center of the coarse textbox (which is also supposed to be the center of the final fine textbox) to an upright position and calculate its normal charbox. If this normal charbox is wider than a threshold, then we try to split it into two or more charboxes, each having approximately an average width, while avoiding split of integral strokes. The resulting normal charbox (or each of the resulting charboxes, if split occurs) is inserted into the normal charbox list of this textbox, ordered from left to right. If the inserted normal charbox shares a significantly overlapping position in the width direction with its neighbor normal charboxes in the list, they may belong to the same individual character in this text. In this case we try to merge the overlapping textbox such that the resulting merged normal charbox contains only one character. The current normal textbox is then adjusted to enclose all the normal charboxes in the current list. A possibly skewed rectangular area (PSR), which stretches from the current fine textbox (which can be obtained by rotating the current normal textbox to its original orientation) to both (left and right) sides by an average character width, is searched for strokes of touching characters, which may touch both graphics and/or characters. Every stroke, whose bounding box is less than the maximal charbox and most of it lies within the search area, is taken as a temporary character and its coarse charbox is added to the coarse charbox list of the coarse textbox. This charbox waits its turn to be treated as a coarse charbox, which may then be merged with yet another charbox. Upon termination of the above processing on the current coarse charbox, the next coarse charbox in the coarse charbox list undergoes the same processing. This is repeated until all the coarse charboxes in the coarse charbox list of the current coarse textbox are processed. At that point, the normal textbox and its normal charboxes are rotated back to their original orientation, and the final fine textbox and its constituent fine charboxes are computed.

The refinement process described above is listed in the Pseudo-code of Fig. 5 and illustrated by the OPD in Fig. 6. The horizontally rotated pixel image of each Refined Charbox can be used as input to any OCR algorithm and the recognized characters would be combined into a whole string.

```
Algorithm 2: Charbox Refinement
Fine_Textbox Charbox_Refinement(Coarse_Textbox CT)
Begin
  Set NCL /*the Normal Charbox List*/ and NT /*the Normal Textbox*/ empty.
  For (each CC /*Coarse Charbox*/ in CCL /*Coarse Charbox List of CT*/)
  Begin
   // 1. Rotate Horizontally:
      NC = Rotate_Horizontally(CC);//Rotate CC to get Normal Charbox.
   //2. Split&Merge:
      If (CC >= Threshold) TNCL = Try_Split(CC);
      for (each TNC in TNCL) //take a Temporary Normal Charbox in TNCL.
      Begin
        Insert(TNC, NCL); //insert TCC into NCL.
        if(TNC overlaps its neighbor) Try_Merge(TNC,NCL);
      End;
      NT = Union(NCL);// normal textbox bounds all of its normal charboxes.
   //3. Search for graphics-touching characters:
      TCCL = Search(PSR);//Search PSR for Temporary Coarse Charboxes.
       for (each TC in TCCL) Add(TC, CCL); //Add TC in CCL.
  End;
  for (each NC in NT)
  Begin
    FC = Rotate_Back(NC);//Rotate back NC to get the Fine Charbox (FC).
    Add(FC, FCL); //Add FC to the Fine Charbox List.
  End;
  return Union(FCL);
End.
```

Figure 5. Pseudo-code of Algorithm 2–Charbox Refinement.

3 Experimental Results

We applied our text segmentation algorithm on a series of portions of medium to poor quality engineering drawings, in which a total of 36 textboxes appeared. 94% were correctly segmented with about 3% false alarm rate. Table 1 shows the statistics of the experimental results of five test drawings. Figs. 7-9 show three of them. In Fig.

7, all eight textboxes are horizontal, but four of them are underlined with a thick line, such that all the characters are connected to the graphics. Nevertheless, all are successfully detected, including the ~ symbols, which are correctly associated with the appropriate textboxes. In Fig. 8, one textbox is missed. It is the dimension text 11, which is really hard to detect, because each of the two 1's touches the back of one arrowhead, and 1 is hard to segment even when it is not connected to graphics. However, the two 45° textboxes are successfully detected in their correct slants along with their degree symbols. The two = symbols are not detected because we have not programmed the system to recognize = as text. Likewise, the two triangles, symbolizing surface quality, are correctly dismissed in the refinement process as non-text. In Fig. 9, the ~R8 and 20° are correctly detected in the precise orientation, as are the vertical ∅770 and ∅774±1 dimensioning textboxes.

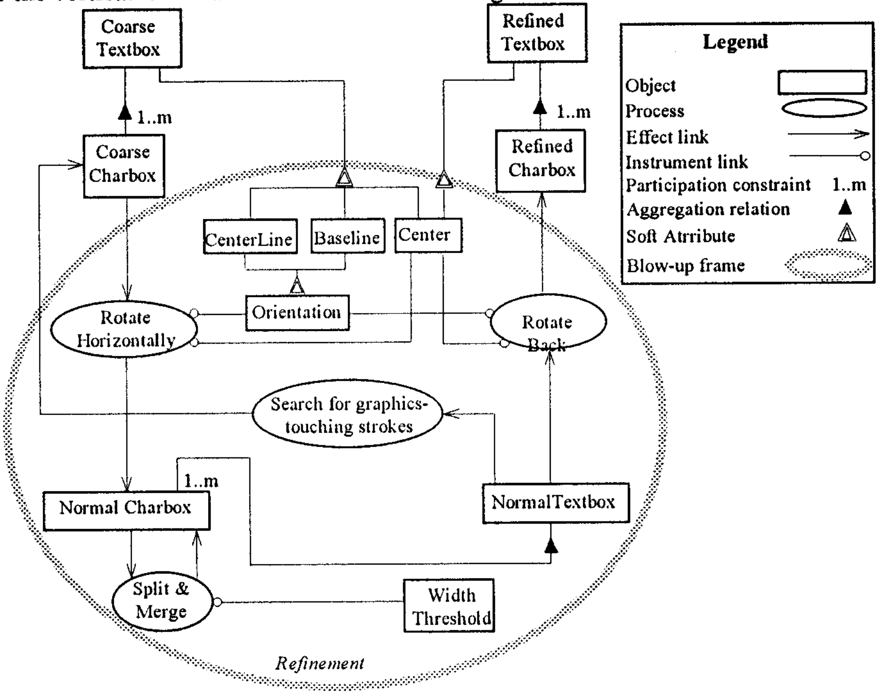


Figure 6. The Refinement process of OPD of Figure 3 blown up.

Table 1: Statistics of the experimental results of text segmentation

Test Drawing	Actual # of Textboxes	# of detected Textboxes	% detection	# of false alarms	% false alarms
Figure 7	8	8	100	0	0
Figure 8	8	7	88	0	0
Figure 9	4	4	100	0	0
Unlisted 1	9	9	100	1	11
Unlisted 2	7	6	86	0	0
Total	36	34	94	1	3

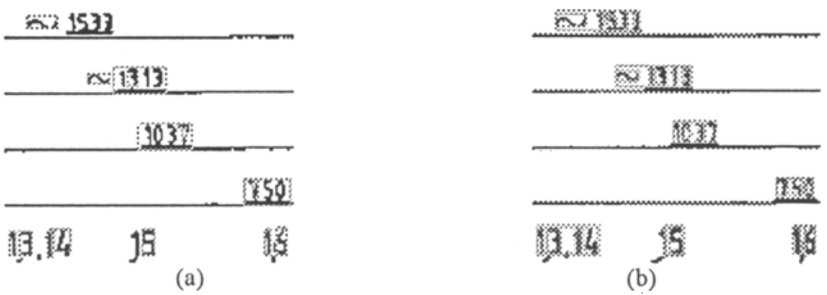


Figure 7. Portion of an engineering drawing: (a) Coarse Charbox Segmentation; (b) Textboxes and their constituent Refined Charboxes.

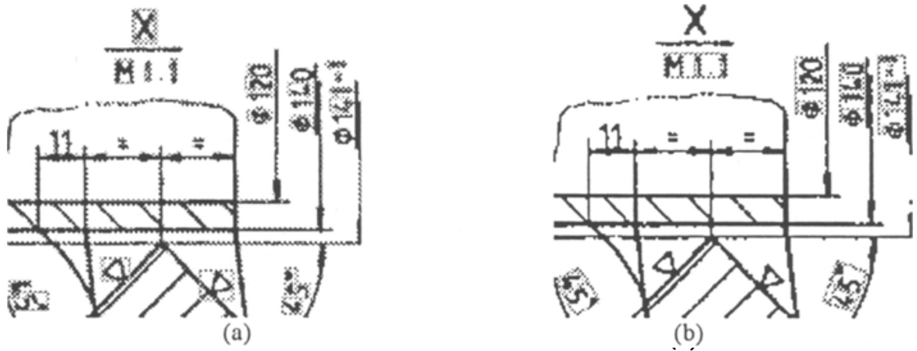


Figure 8. Portion of an engineering drawing: (a) Coarse Charbox Segmentation; (b) Textboxes and their constituent Refined Charboxes.

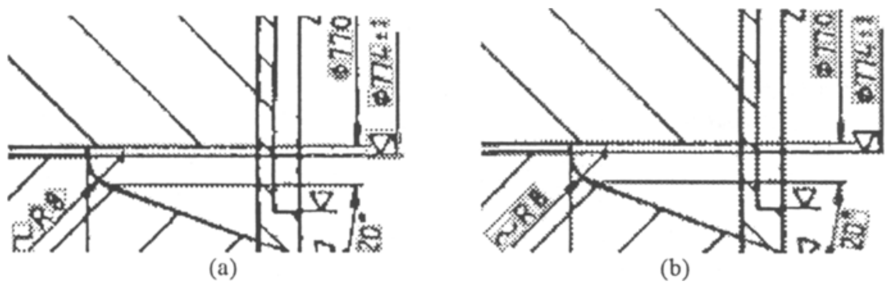
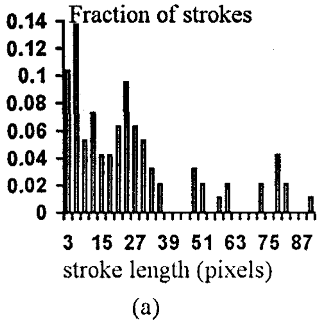


Figure 9. Portion of an engineering drawing: (a) Coarse Charbox Segmentation; (b) Textboxes and their constituent Refined Charboxes.

4 Discussion and Future Work

A method for accurate textbox are charbox segmentation from engineering drawings has been described and implemented. The method handles text at any orientation and carries out accurate segmentation of individual characters even if they touch each other and/or graphic primitives. Thus, an integrated OCR subsystem can be supplied with raster images of characters that can be recognized correctly with high likelihood. Applied to a set of intermediate to poor quality engineering drawings, we obtained 94% textbox detection rate with 3% false alarms.

There is an obvious tradeoff between the detection and false alarm rates: Increasing the detection rate by taking a higher threshold for the stroke length is accompanied by an adverse increase in the false alarm rate. To better understand this phenomena we show in Fig. 10 the histograms of length distribution of text strokes (left) and graphics bars (right) of Fig. 8. Clearly, there is an overlap at the short strokes region. Increasing the stroke length threshold from 12 to 20, for example, increases the overlapping, which, in turn, yields more false alarms. However, we are ready to increase the rate of false alarms if this is the price for increasing the detection rate, because those false alarms can be easily recognized as such due to considerations of incorrect syntax and/or lack of OCR results.



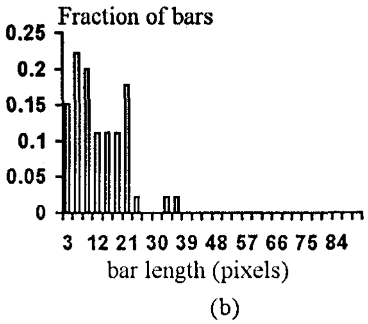


Figure 10. Histogram of length distribution of (a) the text strokes and (b) the graphic lines of the test drawing in Figure 8.

We have tried a neural network based OCR subsystem and obtained satisfactory results. Work is under way to integrate the various components and apply verification based on comparing actual direct dimension measurements from the drawing with OCR output.

References

1 D. Dori and K. Tombre, "From Engineering Drawings to 3D CAD Models: Are We Ready Now?", Computer Aided Design, 1995, 27(4), 243-254.
2 L.A. Fletcher and R. Kasturi, "A Robust Algorithm for Textbox String Separation from Mixed Text/Graphics Images", IEEE PAMI, 1988, 10(6), 900-918.
3 C.P. Lai and R. Kasturi, "Detection of Dimension Sets in Engineering Drawings", Proc. of 2nd ICDAR, Tsukuba, Japan, 1993, 606-613.
4 I. Chai and D. Dori, "Extraction of Text Boxes from Engineering Drawings", Proc. SPIE/IS&T Symposium on Electronic Imaging Science and Technology, Conference on Character Recognition and Digitizer Technologies, San Jose, 1992, SPIE Vol. 1661, 38-49.
5 D. Dori, Y. Liang and I. Chai, "Spare Pixel Recognition of Primitives in Engineering Drawings", Machine Vision and Applications, 6, 1993, 69-82.
6 D. Dori and Y. Velkovitch, "Segmentation and Recognition of Dimensioning Text from Engineering Drawings", Pre Proc. GREC'95, The Penn. State U., USA, Aug., 1995, 141-150.
7 Liu W. and D. Dori, "Sparse Pixel Tracking: A Fast Vectorization Algorithm Applied to Engineering Drawings", Proc. of the 13th ICPR, Vienna, Austria, Aug.,1996
8 D. Dori, "Object-process Analysis: Maintaining the Balance between System Structure and Behaviour", J. Logic Computation, 1995, 5(2), 227-249.
9 Liu W., D. Dori, Tang L. and Tang Z., "Object Recognition in Engineering Drawings Using Planar Position Indexing", PreProc. GREC'95, The Penn. State U., Aug., 1995, 53-61.