Functional Parts Detection in Engineering Drawings: Looking for the Screws

María A. Capellades[1] and Octavia I. Camps[1,2]

[1] Dept. of Electrical Engineering
[2] Dept. of Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802

Abstract. Functional parts – i.e. mechanical parts with intrinsic functionality – such as screws, hinges and gears, are appealing high level entities to be used in line drawing understanding systems. This is because their functionality can be used by a reasoning agent to infer surrounding objects and because they are usually drawn following standards making them easier to be detected. In this chapter, an algorithm for the automatic detection of the schematic representation of screws in mechanical engineering drawings is being presented as a first step towards a function-based line drawing understanding system. All the running parameters required by the algorithm are set according to the American National Standards Institute standards and by using a rigorous experimental protocol characterizing the algorithm performance in the presence of image degradation, thus eliminating the need for *ad hoc* parameter tuning. Experimental results on several real line drawings are also presented.

1 Introduction

The automatic interpretation of engineering drawings remains a difficult task due to their complexity and diversity. In particular, the dual nature of engineering drawings, carrying information in both graphical and textual form, coupled together with the fact that images of paper drawings are, in general, of poor quality and vary with the different individual's drawing styles add more difficulty to the analysis.

We propose to bridge the gap currently existing between the early segmentation stage (lexical stage) and the final interpretation stage (semantic stage) of the interpretation process by using *functional models* as high-level reasoning entities and *physics-based degradation models* to eliminate *ad hoc* tuning of parameters.

Most man-made objects are made to perform one or more functions, and these functions dictate their shape. The concept of function-based vision is not new. It can be found in the literature since the late 70's [15, 19, 2, 3, 6, 14] and has received significantly more attention [7] after the work by Stark and Bowyer [17]. However, until now, line drawings understanding systems have not incorporated any of these advances.

The use of *function reasoning* in line drawing understanding will not only ease the interpretation task, but it will also help to create more complete models with functional information. Thus, we use as high-level reasoning entities mechanical parts with intrinsic functionality. We call these parts, *functional parts*. Examples are screws (tie parts together), hinges (provide articulations between parts), gears (convert power ratios), etc. Functional parts are particularly appealing since:

1. their functionality and inter-relationships with surrounding parts can be used by a high-level reasoning agent to infer the remaining objects in the drawing;
2. they are usually drawn following standard representations, thus they are potentially easier to identify using pattern recognition techniques than arbitrary parts;
3. they provide logical units that can be used to query a database of drawings (consider for example, the task of recalling all parts that have a given type of screw, known to be defective).

However, before a system can reason on any high-level entity, it must first segment the drawing to extract these entities. The results obtained at this first stage, on which all following stages rely on, are highly dependent on the image quality of the document. This quality is degraded by processes commonly used on documents such as printing, photocopying, and scanning. Thus, it is important to take these problems into account when designing the feature extraction algorithms. This is usually done in an *ad hoc* manner by performing expensive trial and error runs to tune the running parameters of the algorithms. Recently, Kanungo *et al* [9] have proposed a physics-based model for the local distortions introduced by printing and scanning processes. We use this model to generate degraded *synthetic* data to characterize the performance of the detection algorithms and to *automatically* select their optimal running parameters.

In this chapter, we focus in developing a tool for the recognition of screws in mechanical engineering drawings following the American National Standards Institute (ANSI) standards for schematic thread representation [13], as a first step towards a function-based line drawing understanding system. The input to the algorithm consists of a list of all the lines found using a modified version of the Orthogonal Zig-Zag (OZZ) algorithm [4]. The screws are detected using knowledge about their standard representation (shown in Fig. 1). All the running parameters required by the algorithm are set according to the ANSI standards and by using a rigorous experimental protocol characterizing the algorithm performance in the presence of image degradation.

2 Screw-Detection Algorithm

The screw-detection algorithm takes advantage of the unique structure of the schematic representation of threads in section in standard drawings [13]. In this representation (see Fig. 1), the threads are drawn as parallel lines separated all by a fixed distance, and with a periodicity in width and length. The difference in

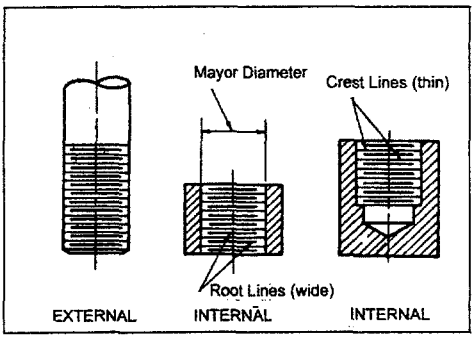


Fig. 1. Schematic Representation of Threads in Section

the lengths of two consecutive lines cannot exceed a given value, and the same applies for the distance between them. All these features must be taken into account when looking for parallel lines that match this pattern. If the number of the consecutive lines that meet the requirements specified above is larger than a threshold, then a screw is said to be found.

2.1 Line Detection: The OZZ Algorithm

The first step towards detecting the screws is to find all the lines in the drawing. This task can be accomplished by using a modified version of the Orthogonal Zig-Zag (OZZ) algorithm [4] briefly described next.

The OZZ algorithm for line detection was developed by Dori *et al.*[4]. The main reason why we use this algorithm is because it is sparse pixel: it avoids massive pixel addressing, visiting some of the pixels in the image only once and never visiting the rest of them. The image is scanned horizontally and then vertically skipping a number of pixels between one scanning cycle and the next. The number of pixels to be skipped is a parameter called *screen-skip*, and its value is determined depending on the minimum line length to be recognized in the drawing.

The implementation of the OZZ algorithm used in this work differs in certain aspects from the original one described in [4]. One of the problems with the OZZ, and with almost any line detection algorithm, is that under the presence of noise it may break a line into shorter segments. In [4], a series of tests are performed to merge line segments. This is done for every possible pair of lines, until no more merging can be done. This results in a computational complexity quadratic in the total number of lines N, that can be devastating if the drawing in consideration is fairly complex. Instead, we have developed a new merging algorithm with complexity proportional to $N \log N$, described next.

2.2 Merging Multiple-Segment Detected Lines

As it was mentioned before, some lines will be broken by the OZZ algorithm into several shorter segments due to noise and other artifacts. In order to correct this,

a merging procedure was developed that does not require to check every possible pair of lines. The procedure takes advantage of the fact that the several segments in which a line breaks into are approximately collinear and have more or less the same width.

The procedure is slightly different depending on the direction of the lines to be merged. Say that we are interested in merging horizontal segments. The list of horizontal segments detected by the OZZ is stored in a heap and sorted by their row coordinates using a heapsort [16] with complexity $N \log N$. A checking procedure is then performed for every group of segments that have row coordinates differing by no more than a threshold *maxdispersion*. This parameter accounts for the maximum shifting in the row coordinates of the multiple segments into which a line was broken by the OZZ. Then, the algorithm goes through every group of segments that have the row coordinates within *maxdispersion*, sorts the segments by the column coordinates of one of their endpoints, also using a heapsort, and checks if the column coordinates of opposite endpoints of consecutive segments in the list differ within a tolerance. If they do, then the two segments are merged, and the checking continues. This is illustrated in Fig. 2a, where checking is performed only in the small square regions between segments.

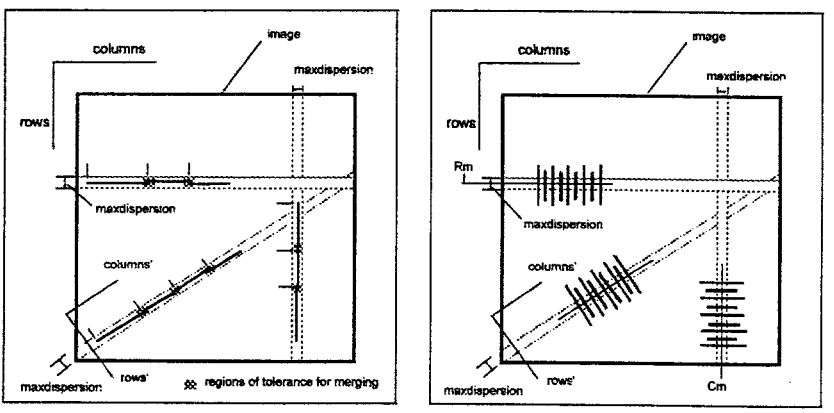


Fig. 2. (a) Merging multiple-segments procedure (b) Screw-detection procedure

For the vertical lines the procedure is identical, only interchanging rows and columns. The process is slightly more complicated for slanted lines, although the philosophy is the same. In this case, one more step is needed before the procedure described above can be applied: the slopes of all the slanted lines found in the drawing need to be sorted using a heapsort. Then, the lines are grouped according to their slope and the previous procedure is applied to each of these groups (Fig. 2a).

2.3 Detecting Horizontal Screws

We describe the algorithm for finding screws with horizontal axes. The algorithms for the other directions are similar. Potential screws with their main axes in the horizontal direction have their root and crest lines vertical. The algorithm first finds groups of vertical lines that have the row coordinate of their midpoints, R_m, within *maxdispersion*. For every one of these groups, the segments are then sorted by the column coordinates of one of their endpoints (both endpoints have the same value for the column coordinate in this case). The algorithm then goes through this sorted list, and for every two consecutive segments in the list three tests are performed. If the width of the segments is less than a threshold th_w, then the algorithm checks if the distance between the two segments is less than a threshold th_d. The third test consists on checking if the difference in the lengths of the two segments is less than a threshold th_l. If the three tests are passed, then a counter is incremented and the checking continues; if not, then it checks if the counter value is larger than th_n. In the case that the counter is greater than th_n, then the hypothesis that a screw has been found is validated, the coordinates of the starting endpoint of the first line meeting the requirements and the ending endpoint of the last line are stored in a file, and the search continues until no more segments in the list of vertical lines are left. All the thresholds, th_w, th_d, th_l, and th_n are set according to the ANSI standards for screw thread representations[1]. The procedure is graphically shown in Fig. 2b and the pseudo-code is given in Fig. 3.

As in the previous section, heaps are used to store the point coordinates and the sort algorithm used is a heapsort. For the cases where the main axis of the screw is vertical or slanted, the procedure is identical, only the coordinates being used need to be changed.

3 Performance Characterization

In order to run the screw detection algorithm given above, five parameters must be set: th_w, th_d, th_l, th_n and *maxdispersion*. The first four parameters correspond to the width, interdistance, length and number of lines in a schematic representation of a screw, respectively. Thus, they are easily set by following the ANSI standards. The fifth parameter *maxdispersion* is the tolerance for the misalignment of the midpoints of the lines representing the screw. Since the alignment of these points is severely affected by the image quality, setting this parameter to a suitable value requires more care.

In the sequel, an experimental protocol for the performance characterization of the algorithm when the image is increasingly degraded and the parameter *maxdispersion* is varied is presented. This characterization not only shows the validity of the algorithm by illustrating how it behaves in the presence of image degradation, but also provides a tool to select the optimal value of the parameter *maxdispersion* to minimize the probabilities of misdetection and false alarm.

```
SCREW DETECTION ALGORITHM

For every vertical line in the image
    do
    Store its midpoint row R_m in a heap
    end do
Sort the heap
Group the lines with R_m within maxdispersion
For every group of lines
    do
    Store the column of one endpoint in a heap
    Sort the heap
    Initialize search
    For every two consecutives lines in the heap
        do
        If their widths < th_w
          and
          the distance between them < th_d
          and
          the difference in their lengths < th_l
        then
              increment counter
        else
              end search
        If the search ended
          then
          Initialize search
          If counter is > th_n
              then
              store results in output file
    until no more lines are left in the heap
```

Fig. 3. Pseudo-code of the detection algorithm for horizontal screws.

3.1 Experimental Protocol

The performance characterization of the algorithm was done following the guidelines given in [5] and [10]. In particular, the methodology in [10] is very useful because it allows to integrate a large number of operating curves relating the probabilities of mis-detection and false alarms for each parameter setting into a single performance curve.

The methodology consists of two steps of standard decision analysis and two steps inspired by psycophysical methods [10].

Step 1: First, the two noise-free images shown in Fig. 4a and Fig. 4b were generated. Fig. 4a is a *target* image consisting of an ideal screw drawing and it will be used to estimate the probability of a misdetection. Fig. 4b is a *no-target* im-

age consisting of a drawing closely resembling the one shown in (a) but that does not follow the ANSI standards. This image was designed to estimate the probability of false alarm.

The images were then degraded to simulate an increasing number of duplication using the perturbation model given in [9]. According to this model, foreground and background pixels are changed following exponential distributions. The probability of a foreground pixel changing to the background was set to $P(0 \mid d, f) = \exp(-0.9d^2)$ and the probability of a background pixel changing to the foreground was set to $P(1 \mid d, b) = \exp(-2d^2)$ where d is the inverse distance, f is foreground and b is background. A total of 1000 perturbed images were generated, divided in five sets corresponding to five different levels of perturbation. The level of perturbation was varied by changing the threshold value th_b (th_f) used to decide if a background (foreground) pixel is actually changed given that its probability of changing indicates it should. The values used for these thresholds were: level 1, th_b=th_f=0.9; level 2, th_b=0.8, th_f=0.9; level 3, th_b=0.8, th_f=0.7; level 4, th_b=0.8, th_f=0.5; and level 5, th_b=0.8, th_f=0.3. Images for levels 1 and 5 are shown in Fig. 4. Note that as the perturbation level is increased the target and no target images become more and more similar.

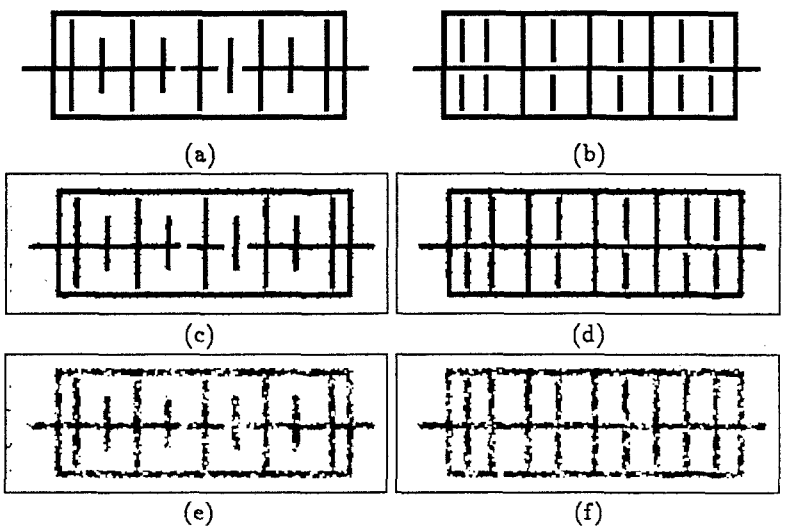


Fig. 4. Test images, level 0: (a)target (b)no-target, level 1: (c) target (d) no-target, and level 5: (e)target (f)no-target

Step 2: The detection algorithm was run on the test images for all the perturbation levels and for values of the parameter *maxdispersion* equal to 0.01, 0.02, 0.04, 0.06 and 0.08 inches. Operating curves of the probability $P(misdetection) = P(no\text{–}target \mid target)$ plotted against the probability $P(falsealarm) = P(target \mid$

no – target) for each of the perturbation levels and *maxdispersion* values were generated.

Step 3: From the operating curves, the value of the probability of error $P(E)$ was calculated for each level and each value of *maxdispersion*. The probability of error was taken as the average of the probabilities of false alarm and misdetection when the last one is equal to 0.2.

Step 4: A plot of $P(E)$ versus level of perturbation is then obtained for each value of *maxdispersion* (Fig. 5a). From this plot, the *critical signal variable* defined as the maximum level of perturbation that an image can have in order to get a probability of error $P(E)$ less than 0.16 is measured. Finally, a plot of the critical signal variable versus the variable of interest, *maxdispersion*, is obtained (Fig. 5b).

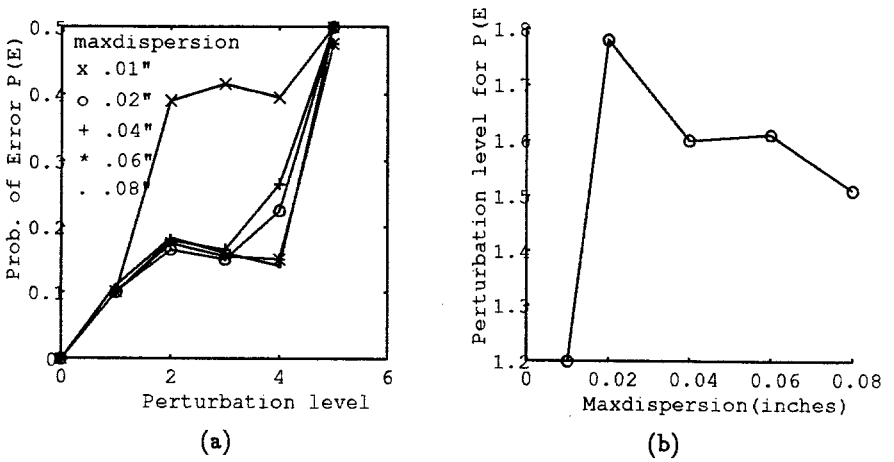


Fig. 5. Results of the Performance Characterization.

From the results shown in Fig.5b, it is seen that the optimal value for the *maxdispersion* parameter – i.e. the *maxdispersion* value corresponding to the maximum level of perturbation and $P(E) = 0.16$ – is between 0.02 and 0.04 inches. In practice it is very difficult to find images degraded more than a level 2. Usually, we find images in the range between 1 and 2.

4 Results with Real Images

The screw-detection algorithm was also tested on fourteen scanned images of modified real mechanical engineering drawings from [12] with up to 10 screws. Some of the images were scanned at a resolution of 300 dpi while others with smaller details were scanned at 450 dpi. All of them were globally thresholded.

The running parameters for the algorithm were set as: *maxdispersion*=0.03", th_w=0.025", th_d=0.15", th_l=0.15", and th_n=5. Table 1 lists for each of the test images, the number of line segments output by the OZZ algorithm, the number of screws in the drawing, the number of misdetections, the number of false alarms, and the running time on a SUN SparcStation 5. Although the number of real images is low since they are difficult to obtain, the overall misdetection and false alarm rates are also given. These rates were 17.6% and 9.8%, respectively. However, three of the false alarms in two of the tested images were found among the text in the drawing. Thus, if the algorithm were to be run after segmenting out the text, the false alarm rate would go down to 3.9%.

Table 1. Results on Real Images

Image	Segments	Screws	MD	FA	Time (*sec.*)
Air1	2311	9	3	1	10.3
Air2	1607	7	1	0	7.1
Air3	1243	1	0	0	3.6
Assembly	798	2	0	0	3.2
Bearings	1548	10	3	1	5.9
Bolt	256	1	0	0	2.2
Bplate	641	2	0	0	3.2
Cast	870	0	0	2	3.9
Collar	141	2	0	0	1.4
Conveyor	978	2	0	0	2.6
Holes	508	0	0	0	2.0
Joint	1069	4	0	0	3.0
Lift1	1344	3	1	1	5.2
Stand	1380	8	1	0	4.7
TOTAL		51	9	5	*Aveg.* 4.2

Three of the test images, their segmentation using OZZ and the bounding boxes of the detected screws are shown in Figs. 6-8. Fig. 6 is an example of a drawing where all screws were detected and there were no false alarms. Fig. 7 is an example where there was one misdetection because the OZZ algorithm detected only four segments for the bottom left screw. Finally, Fig. 8 is an example with no screws that resulted in two false alarms. Interestingly enough, the false alarms are not located on the detail representation of the screw but on the text area, and could have been avoided by segmenting out the text first.

5 Implementation Details

The algorithms were implemented using the C language [11]. The input to the program is a bitmap image in PBM format, or *portable bitmap* format. There are

three different modules in the program, one for each one of the three main groups of lines: vertical, horizontal and slanted lines. Each module performs the OZZ algorithm to detect the corresponding lines, merges the segments and detects the presence of schematic representations of screws. At the end of each module, the information extracted is stored in an output file. For lines, the information consists of the endpoints and width of the line. For the screws, the information consists of the coordinates of the upper-left corner and bottom-right corner of the bounding box of the screw.

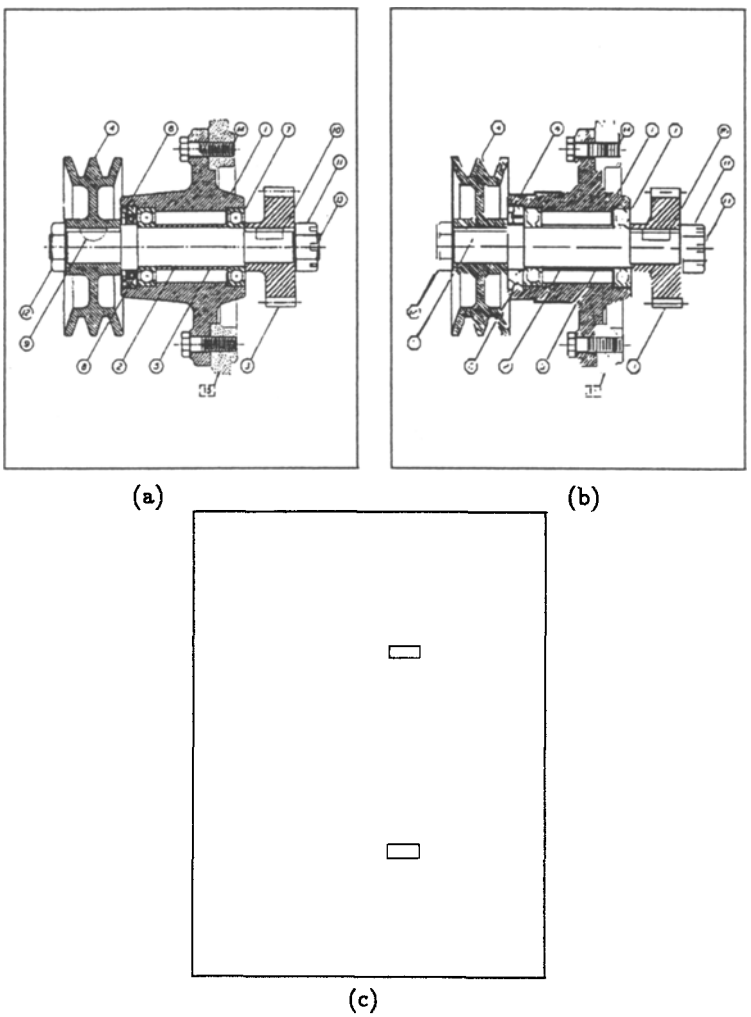
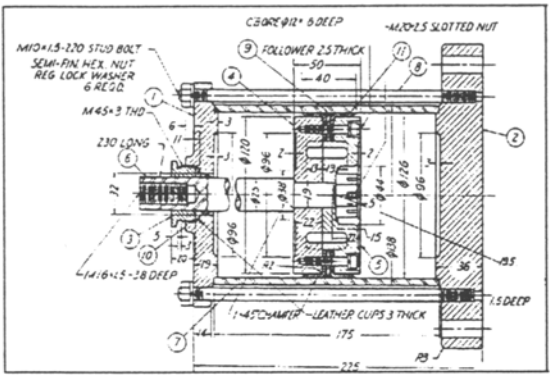
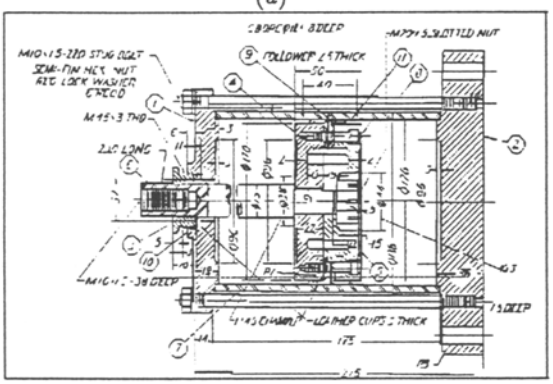


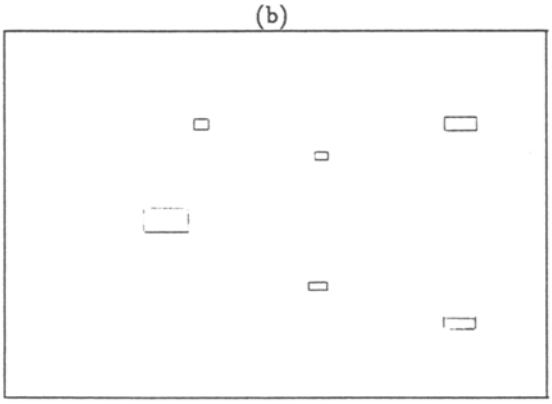
Fig. 6. *Assembly* image: (a)Original (b)after OZZ (c)screws



(a)

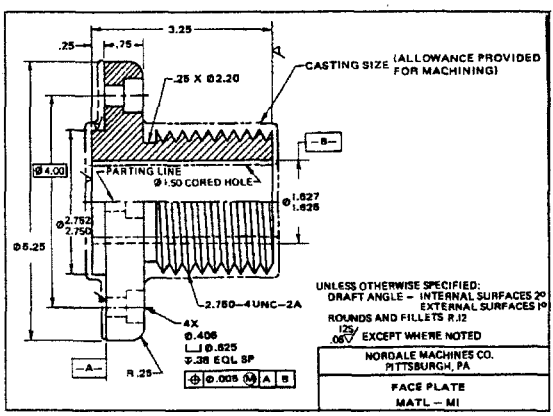


(b)

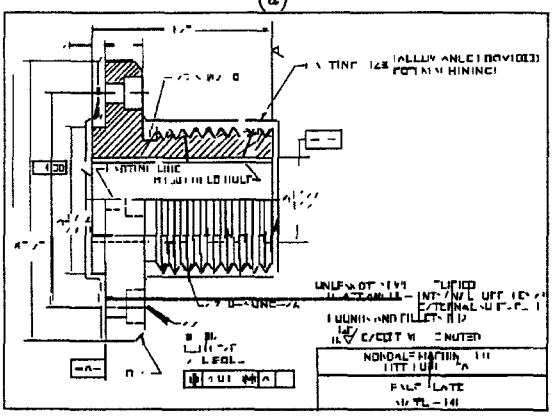


(c)

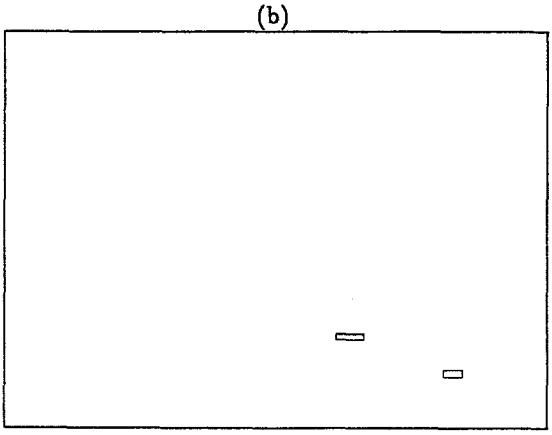
Fig. 7. *Air2* image: (a)Original (b)after OZZ (c)screws



(a)



(b)



(c)

Fig. 8. *Cast* image: (a)Original (b)after OZZ (c)screws

6 Summary

Functional parts can be extracted taking advantage of their standard representation. An algorithm to detect the presence of screws with schematic representations in mechanical engineering drawings was presented. All the parameters needed to run the algorithm were set according to the ANSI standard or using a rigorous experimental protocol. The algorithm performance was characterized by running it through 1000 images under five different levels of degradation. The algorithm was also tested on fourteen real images of moderate complexity with overall misdetection and false alarm rates of 17.6% and 3.9%, respectively.

References

1. ANSI Y14.6: Screw Thread Representation. *American National Institute of Standards*, (1983)
2. Brady, M., Agre, P. E., Braunegg, D. J., Connell, J. H.: The mechanics mate. In T. O'Shea, editor, *Advances in Artificial Intelligence*, pages 79–94. Elsevier, New York (1985)
3. Di Manzo, M., Trucco, E., Giunchiglia, F., Ricci, F.: FUR: Understanding FUnctional Reasoning. *International Journal on Intelligent Systems*, 4 (1989) 159–183
4. Dori, D., Liang, Y., Dowell, J., Chai, I.: Sparse–Pixel Recognition of Primitives in Mechanical Engineering Drawings. *Machine Vision and Applications*, 6 (1993) 69–82
5. Haralick, R.: Performance Characterization Protocol in Computer Vision. *Proc. of NFS/ARPA Workshop on Performance versus Methodology in Computer Vision*, June (1994)
6. Ho, S.: *Representing and using functional definitions for visual recognition.* PhD thesis, University of Wisconsin, Madison, WI (1987)
7. IEEE Computer Society Technical Committee on Pattern Analysis and Machine Intelligence. IEEE Computer Society Workshop on the Role of Functionality in Object Recognition. june (1994).
8. Joseph, S. H., Pridmore, T. P.: Knowledge–Directed Interpretation of Mechanical Engineering Drawings. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(9) (1992) 928–940
9. Kanungo, T., Haralick, R. M., Phillips, I.: Global and Local Document Degradation Models. *Proc. of Second International Conference on Document Analysis and Recognition*, October (1993) 20-22
10. Kanungo, T., Jaisimha, M. Y., Palmer, J., Haralick, R.: A Methodology for Analyzing the Performance of Detection Algorithms. *Proc. of the 4th International Conference on Computer Vision*, May (1993) 247–252
11. Kernighan, B. W., Ritchie, D. M.: *The C Programming Language.* Prentice Hall, Murray Hill, NJ (1988)
12. Luzadder, W. J.: *Fundamentals of Engineering Drawing.* 8th edition, Prentice Hall, Englewood Cliffs, NJ (1981)
13. Luzadder, W. J., Duff, J. M.: *Fundamentals of Engineering Drawing.* Prentice Hall, Englewood Cliffs, NJ (1993)
14. Minsky, M.: *The Society of Mind.* Simon and Schuster, New York (1985)

15. Rosch, E., Mervis, C. B., Gray, W. D., Johnson, D., Boyes-Braem, P.: Basic objects in natural categories. *Cognitive Psychology*, 8 (1976) 382–439
16. Sedgewick, R.: *Algorithms*. Addison–Wesley Publishing Co., New York (1988)
17. Stark, L., Bowyer, K.: Achieving generalized object recognition through reasoning about association of function to structure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, October (1991) 1097–1104
18. Vaxiviére, P., Tombre, K.: CELESSTIN: A System for Conversion of Mechanical Engineering Drawings into CAD format. *IEEE Computer: Special Issue on Document Image Analysis Systems*, July (1992) 46–54
19. Winston, P. H: Learning structural descriptions from examples. In P. H. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, New York (1975)