Automated CAD Conversion with the Machine Drawing Understanding System: Concepts, Algorithms, and Performance

Dov Dori and Liu Wenyin

***Abstract*— The Machine Drawing Understanding System (MDUS) is an experimental CAD conversion system aimed at realizing the entire process of understanding mechanical engineering drawings, from scanning to three-dimensional (3-D) reconstruction. This paper describes the structure, algorithms and current performance of MDUS. The modular structure of the system provides for further improvements and makes it an ideal workbench for researchers wishing to test their own algorithms and incorporate them into the system.**

***Index Terms*—CAD conversion, document analysis and recognition, engineering drawings, graphics recognition, interpretation, object–process methodology, raster-to-vector, vectorization.**

I. Introduction

Although quite a few engineering drawings recognition systems have been developed, a fully automated interpretation is still a tough problem due to the sheer complexity of most real-life engineering drawings, their intricate semantics and the presence of noise. In this paper, we present the Machine Drawing Understanding System (MDUS), whose ultimate goal is to realize the entire process of engineering drawings interpretation, including two-dimensional (2-D) understanding and three-dimensional (3-D) reconstruction. We have implemented functions of 2-D conversion from raster image to 2-D object form: vectorization, text segmentation and recognition, arc segmentation, leader detection, dashed line detection, and hatching line recognition. Compared to previous published work [1], [2], the performance of MDUS has been significantly improved, as shown in the experiments, evaluated by a third party protocol [3].

II. System Structure and Behavior

In this section, we explain the object structures and generic algorithms used in MDUS.

A. Object Structure and Organization

At the basis of MDUS is a hierarchy of graphic object classes that inherit from a root class called Primitive, whose class hierarchy is presented in detail in Section II-D. GraphicDataBase is the database singleton class that stores these graphic objects. At the 2-D conversion and 2-D understanding phases of MDUS, graphic data are managed by Graphic Object Database—an object of the class GraphicDataBase, which contains a reference to the original raster image and lists of the graphic objects. Graphic Object Database also contains an object of the class PositionIndex, which is based on a data structure called the *position index* [4]. It indexes all graphic objects by their planar positions in the drawing, making it possible to realize mapping from positions to graphic objects in the drawing. This approach yields high time efficiency for a variety of higher level segmentation and recognition tasks. Auxiliary classes including Point, Rectangle, SkewRectangle, and Angle, are also defined for convenient storage, search, retrieval, and manipulation of the graphic objects.

Manuscript received May 1, 1996; revised April 2, 1997, January 5, 1998, October 29, 1998, and November 24, 1998. This work was supported in part by the Technion VPR Fund.

D. Dori is with the Faculty of Industrial Engineering and Management, Technion—Israel Institute of Technology, Haifa 32000, Israel (e-mail: dori@ie.technion.ac.il).

L. Wenyin is with the Microsoft Research Sigma Center, Beijing 100080, China (e-mail: wyliu@microsoft.com).

Publisher Item Identifier S 1083-4427(99)03326-3.

B. Graphic Class Hierarchy and the Generic Graphics Recognition Algorithm

Machine drawings and other types of engineering drawings contain various classes of graphic objects that share many common features in terms of both structure and behavior. Organizing them in an inheritance hierarchy, illustrated in Figs. 1 and 2, is therefore not only possible but also necessary for code clarity, efficiency, and reusability. The generic graphics recognition procedure, described briefly below, is applicable to all these classes.

The objectives of the graphic recognition algorithm is to group raw wires, resulting from the vectorization, into meaningful primitive and higher level graphic objects, and recognize them along with their attributes. Wires are primitive graphic objects that include bars, polylines, and arcs. Higher level graphic objects include characters and text, arrowheads and leaders, dashed lines and hatched areas.

To generalize the graphic recognition process, we have devised a generic algorithm for recognizing a variety of classes of graphic objects [4], whose C++ code is listed in Fig. 3. The generic graphic object recognition algorithm is a two step procedure, based on the hypothesize-and-test paradigm. The first step is *hypothesis generation*, which generates the hypothesis of the object being sought by finding and returning its first key component. For each class, the first key component is defined according to the syntax and semantics of this class. For example, a key component of a dashed straight line is a bar (a dash), while both a bar and an arc can be the key component of a dashed arc. The second step is the *hypothesis testing*, which first constructs an initial graphic object from its first component and then extends it gradually, using its other detected components. The construction of the initial graphic object is realized by the C++ statement new, which creates an empty graphic object of the class, and the member function fillWith of the class, which partially fills the empty graphic object attribute values with the parameters conveyed by the first key component prm0. For example, the endpoints and line width of a dashed line are filled directly by the attribute values of its first key component bar. The extension of the graphic object is realized by the member function extend of the class. Being the core of the recognition process of a graphic object, it applies a stepwise extension of the graphic object to the largest extent possible by finding one of its other components in each extension iteration. The syntactic and semantic specifications of the function extend for the concrete classes of graphic objects are defined in the overriding functions in corresponding classes, as discussed in detail in Section III-B.

Finally, the recognition reliability of this presumed graphic object is tested by the function isCredible(). If the object passes the test, it is inserted into the graphic database, otherwise it is deleted. The recognition process of a concrete graphic object class is triggered by an initiation of the template function detect. For example, to detect dashed lines, we call the function detect((DashedLine*)0, (GraphicDataBase*)aGraphicDataBase).

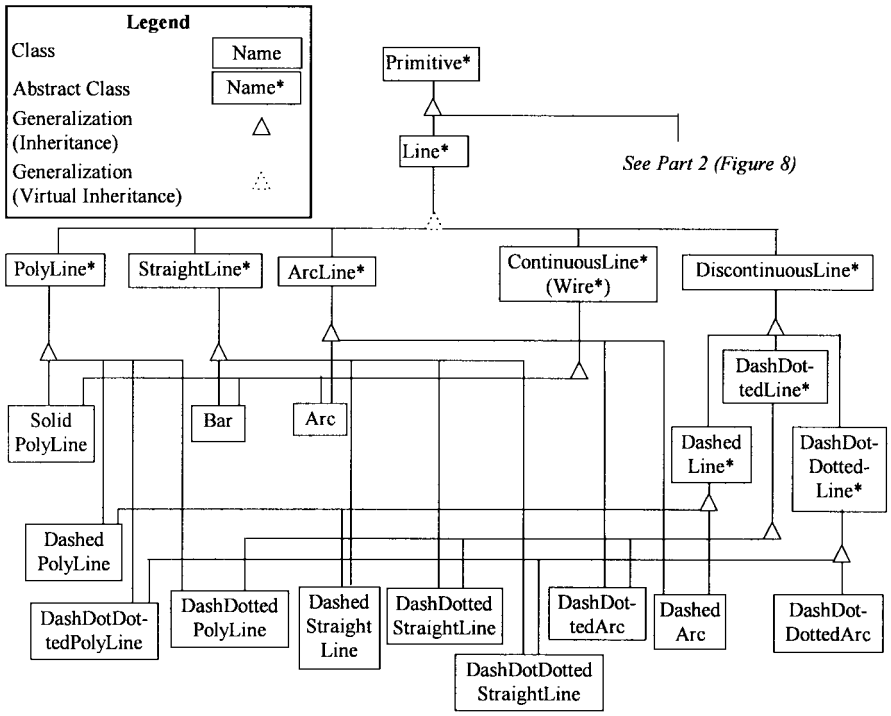


Fig. 1. Class inheritance hierarchy of graphic objects—part 1.

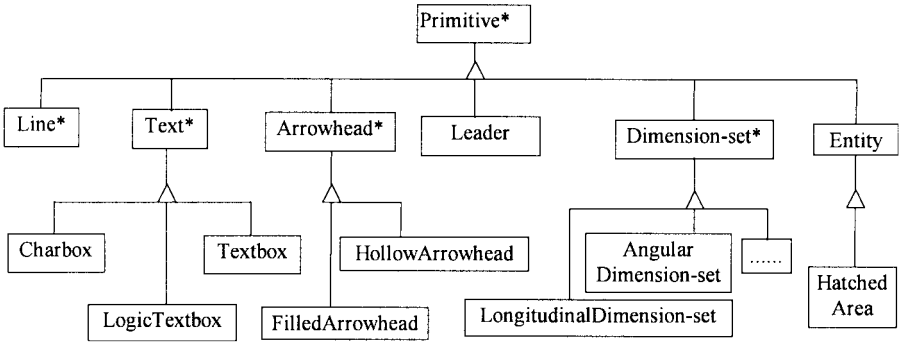


Fig. 2. Class inheritance hierarchy of graphic objects—part 2.

```
template <class PrimType>
void detect(PrimType*, GraphicDataBase* gdbase) {
  Primitive* prm0;
  while ((prm0 = PrimType::firstComponent(gdbase)) != NULL)
    constructFrom((PrimType*)0, gdbase, prm0);
}

template <class PrimType>
PrimType* constructFrom(PrimType*, GraphicDataBase* gdbase, Primitive* prm0) {
  PrimType* prm = new PrimType(1);
  if (prm->fillWith(prm0)) {
    for (int aDirection=0;aDirection<=prm->maxDirection();aDirection++)
      while (prm->extend(gdbase,aDirection));
    if (prm->isCredible()) {
      prm->addToDataBase(gdbase);
      return prm;
    }
  }
  delete prm;
  return NULL;
}
```

Fig. 3. Code of the C++ implementation of the generic graphic object recognition algorithm.

III. Processes and Their Algorithmic Implementation

MDUS has been implemented in C++ and X-Windows environment on both Silicon Graphics type workstations (Irix 5.3) and SUN Sparcstations (Solaris 2.5). The executable codes of both versions are available from the ftp address [5]. In this section we discuss the main recognition processes within MDUS and the algorithms that implement them.

A. Vectorization

Existing vectorization methods are usually not satisfactory for processing real life drawings from both time efficiency and wire recognition quality aspects. Inspired by the OZZ method [1], we have devised the Sparse Pixel Vectorization (SPV) method [6], which further improves the vectorization results of OZZ. The basic idea of the SPV algorithm is to make cross sectional runs (called width runs) of the black area representing the line at intervals along the tracking direction and record the middle points of these sections. These points

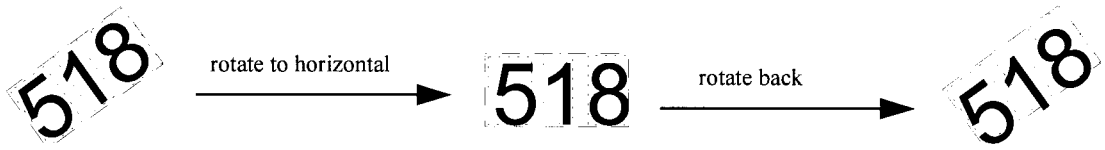
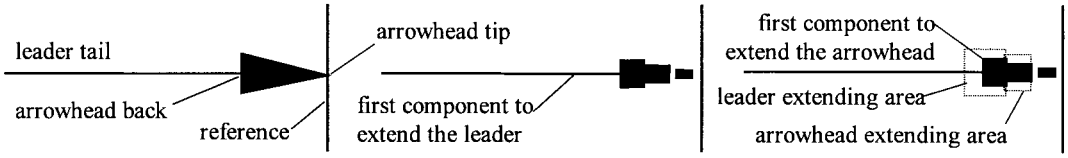


Fig. 4. Illustration of the three text segmentation steps.



(a) (b) (c)

Fig. 5. Illustration of leader detection (a) original image, (b) vectorization result, and (c) arrowhead detection.

are considered to lie along the medial axis of the black area and are used to construct the vectors.

B. Graphic Object Recognition

The Graphic Recognition process consists of a set of specialized graphic object recognition algorithms, all of which are based on the generic graphic recognition algorithm, presented in Section II-D. Each such algorithm is instantiated by specifying the particular syntax of the corresponding class of graphic objects to be recognized. This generic recognition algorithm has been successfully applied on a variety of graphic objects, including textboxes [7], arcs [8], leaders, dashed lines [9], and hatching lines [10].

1) Text Segmentation and Recognition: Being a type of graphic object in engineering drawings, text requires both segmentation and precise recognition. In text segmentation, textboxes and their bounding character boxes (charboxes) are segmented. In character recognition, the image inside the charbox is classified as a particular character or symbol. Character recognition in drawings takes advantage of text-intensive optical character recognition (OCR) achievements. We use an off-the-shelf neural network solution [11]. Unlike most text segmentation methods, which operate at the raster level, our text segmentation method is vector-based. The segmentation, which depends only on wires, consists of three main steps. The first step is segmentation of charboxes from the graphics. In this step, we find coarse bounding rectangles of isolated characters by a recursive merging algorithm, which is a specialization of the generic graphic recognition algorithm. Any short wire can be selected as the first key components of a charbox. The bounding box of this key component is used to construct the initial charbox. In the extension procedure, any stroke-like short solid line segment that touches the set of already found strokes is also considered as a stroke of the current charbox, so the charbox is expanded to contain this new stroke. When no more strokes can be included, or the current charbox is greater than a predefined maximum size, the extension procedure stops. In the second step we gather statistics on the sizes of individual charboxes and obtain the average charbox width and height. The average charbox width and height are used to combine adjacent charboxes into textboxes. The key component of a textbox is a charbox segmented in the first step. The first charbox is used to construct the initial textbox. The textbox is extended to include charboxes that are adjacent to it. The textbox orientation is determined by the centers of its constituent charboxes. Finally, in the third step, the textboxes are resegmented into precise individual charboxes according to the average charbox width and height and the textbox orientation. The three segmentation steps are illustrated in Fig. 4. As we show in [7], our text segmentation algorithm successfully segments connected characters from graphics, a difficult task which is rarely performed by other text segmentation algorithms that rely on connected component analysis.

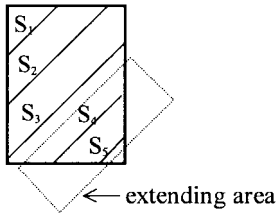


Fig. 6. Illustration of detection of a group of hatching lines.

2) Arc Segmentation: Arcs are important primitives in engineering drawings. The original arc segmentation method we used is Perpendicular Bisector Tracing (PBT) [2]. It accepts as input is a chain of bars resulting from the OZZ vectorization method. Since the current vectorization module, SPV, yields polylines, which usually approximate the arc image more precisely, we apply arc segmentation to each polyline resulting from the vectorization using the Stepwise Recovery Arc Segmentation (SRAS) algorithm [8]. Using the SRAS algorithm, we find the most suitable arc along which all the characteristic points of the polyline fall. Applying strict criteria, such as requiring that the maximal allowed distance between each polyline segment and its corresponding arc piece be less than half the line width, we recognize arcs accurately.

3) Leader Detection: Leaders are an important part of dimensioning annotation in engineering drawings. A leader is characterized as having an arrowhead, a thin tail attached colinearly to the back of the arrowhead, and a reference line, at which the arrowhead points attached orthogonally at the tip of the arrowhead. After vectorization, the arrowhead, shown in Fig. 5(a), frequently becomes several thick short bars, as in Fig. 5(b). We select the tail, which is a thin bar, as the first key component of a leader. The arrowhead is the other component we search for during the extension of the leader. The area for leader extension is a rectangle whose width is three times the tail width and whose length is nine times the tail width that stretches out from one endpoint of the tail, as shown in Fig. 5(c). The thick bar found in this area is assumed to be the back part of the arrowhead. Therefore it is a key component of an arrowhead being detected during the leader extension. If the arrowhead is successfully detected, the leader is completely detected because both its tail and

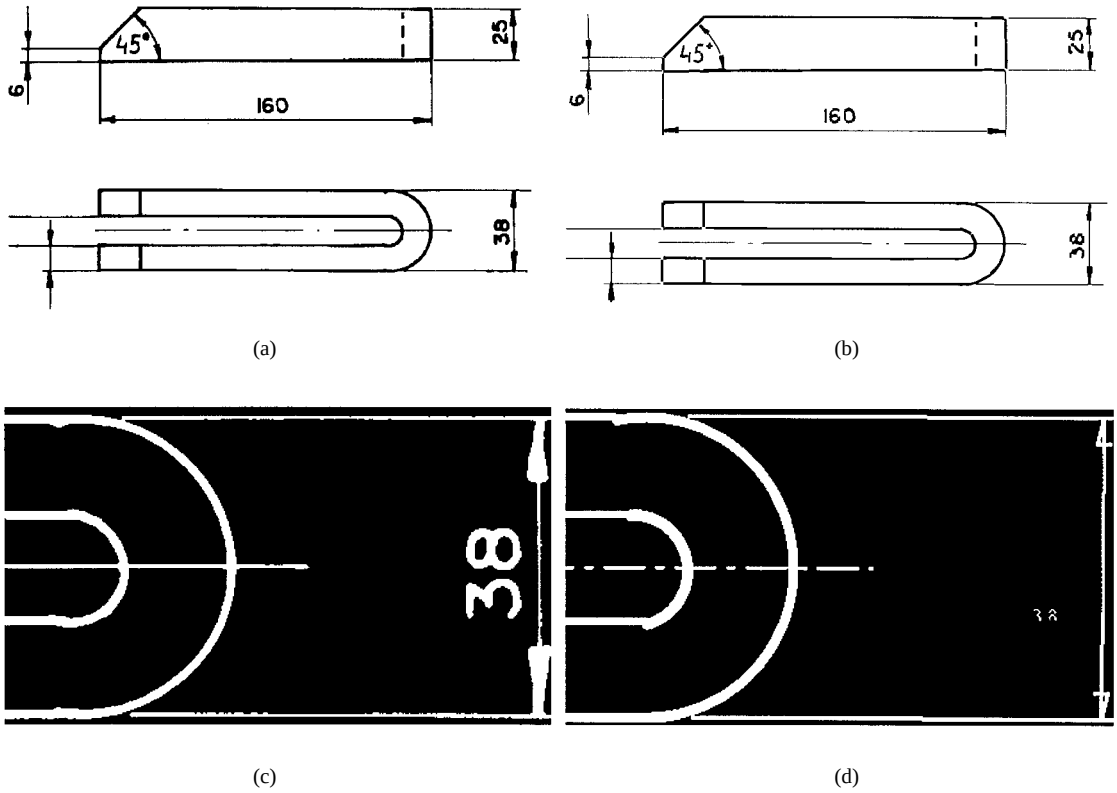


Fig. 7. Results of 2-D conversion: (a) original image, (b) result of Sparse Pixel Vectorization (without post processing), (c) enlarged view of the right-bottom part in (a), and (d) arc segmentation, text segmentation and recognition, leader recognition, and dash-dotted line detection of (c).

arrowhead are found. In the initial construction of the arrowhead, the endpoint that is close to the tail is used as the back of the arrowhead and the other endpoint is used as the tip of the arrowhead. The arrowhead is further extended to its tip direction to find other parts (collinear short thick bars) of the arrowhead one by one. This is done while verifying that their width gradually decreases until it is less than the tail width, or until it begins to increase, or until an orthogonal reference is encountered. To detect bidirectional arrows, we also attempt to extend the leader to the other endpoint of its tail to see if an arrowhead is attached also to the other edge of the tail.

4) Dashed Line Detection: MDUS detects both dashed straight lines and dashed arcs. The syntax specifies that a dashed line consists of at least two equispaced dashes with the same line width. The dashes are constrained by the same geometry (either straight for dashed straight lines or circular for dashed arcs). Further, they may have about equal length in the case of a dashed line, and two alternating lengths, one for long dashes and very small length for short dashes (dots) in the case of a dash-dotted line.

5) Hatching Line Detection: Hatching lines are associated with a border that forms the hatched area representing a cross section of an object. Our current implementation of hatched area detection is limited to the detection of the hatching lines [10]. The syntax of hatching lines specified in the generic recognition algorithm is that they are a group of thin, equispaced parallel slanted bars with their end points usually falling on the area's contour. Therefore, the first key component of a group of hatching lines is a bar that meets the following conditions, such as S1 in Fig. 6:

1) the bar should be thin;
2) the bar's slant should be between 30° and 60°;
3) each of the two bar endpoints has to lie along some other object (which is usually part of the hatched area contour).

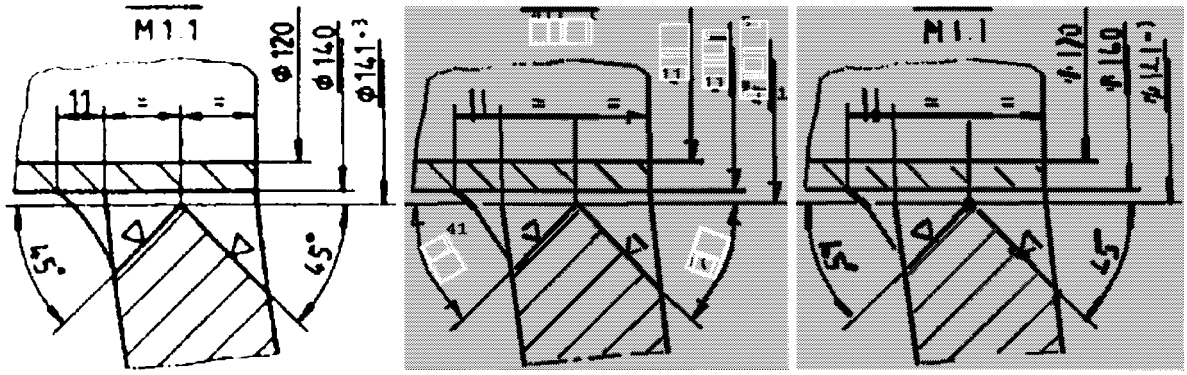
The extension search area is determined as the area extending farther away from the newly found hatching line, which is most remote from the first key component in a direction orthogonal to the hatching line direction. See, for example, the rectangle represented by dots in Fig. 6. The length of the search rectangle is somewhat (e.g., 20%) longer than the length of the key component (or the most recently found hatching line in subsequent iterations), and its width is twice the average space distance between two neighboring hatching lines. Initially, this width is set as the length of the first hatching line. Extending candidates are bars found within the extending area that meet the following conditions:

- The bar's width should be similar to the average width of the hatching lines that have already been detected in the current group.
- The space between each new hatching line and the last detected hatching line should be about equal to the cumulative average of the previous spaces between neighboring lines.
- The slant of the new hatching line should be approximately equal to the cumulative average of the previous line slants.
- The new component should share with the previous component the same contour part, such that it contributes to a meaningful continuity of the hatched area.

When more than one candidate that satisfies the constraints is found, the nearest one to the most recently detected component is selected. Thus, S4 rather than S5 in Fig. 6 is selected as the component that follows S3. After finding each new component, the average space and slant are dynamically updated.

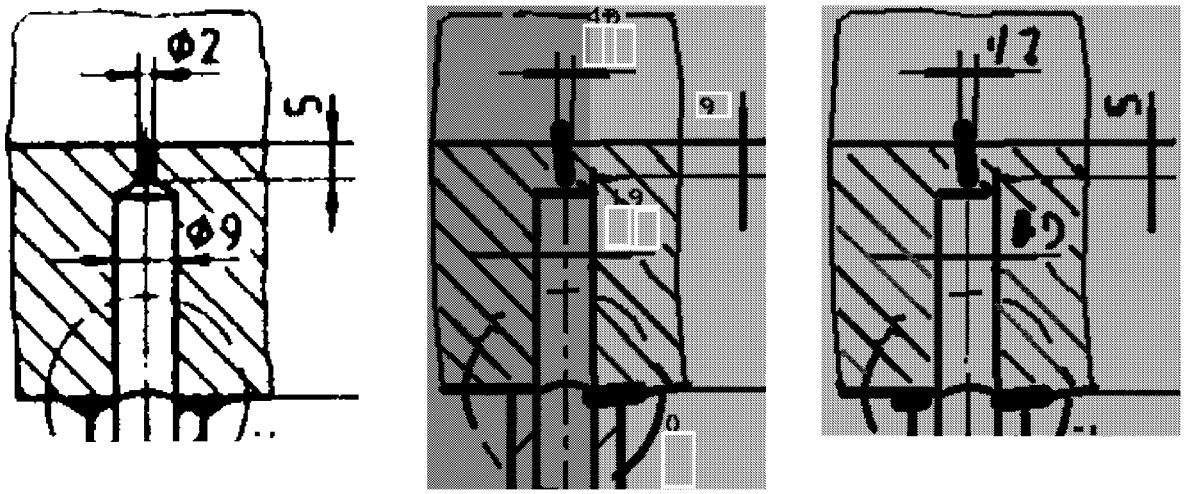
C. Integration and Recognition Order Inter-Dependency

Following vectorization, the object Graphic Database contains only wires. This is the starting point for higher level object recognition.



(a) (b) (c)

Fig. 8. Results of 2-D conversion: (a) original image, (b) recognized graphics (bars, texts, and leaders), and (c) detected hatching lines.



(a) (b) (c)

Fig. 9. Results of 2-D conversion (a) original image, (b) recognized graphics (bars, texts, and a dash-dotted line), and (c) detected hatching lines.

We carry out the recognition of objects class by class. After each class is recognized, the database contains more high-level objects, which can be used for recognizing yet higher level classes. The recognition order is difficult to determine. Several objects may coexist closely and may constraint the existence of others. They may also be components of the same higher level object, with one's existence strengthening the evidence for the existence of others. This mutual dependence of objects is reflected in the syntax of the engineering drawings. For example, in a dimension-set, which is a high level object, one or two leaders and text strengthen the evidence for the existence of each other. As another example, for arc segmentation, there may be more clues for center and endpoint detection if centerlines and tangent lines respectively are detected.

While we have shown that it may be harmful to be committed to a fixed, predefined recognition sequence, current CAD conversion systems lack this type of considerations, so recognition of one class of objects depends mostly on prior detection of certain classes of other objects. The underlying assumption is that objects used to recognize higher level objects are themselves correctly recognized. Unfortunately, this is not always the case, as objects are frequently misdetected or not detected at all. In these cases, no object or a wrong object, will be recognized. The modest automatic recognition rates are, to a large extent, a direct consequence of a fixed predefined class recognition sequence.

Currently, the order in which graphic objects are segmented and recognized is not very critical, since the algorithms are relatively independent of each other. However, the integration of these recognition procedures, which is expected to increase the total recognition performance, is supported by a dynamic control mechanism, whose search decision at each time point is based on the combination, location and orientation of the graphic objects detected at that time point. This is done by dynamically triggering the recognition of one or more classes of graphic objects while recognizing a certain graphic object class. For example, in leader detection we trigger an arrowhead detection process. Bar or arc detection may be triggered during dashed line detection. Before the candidate is tested for being a dash of the dashed line, a bar or an arc is first constructed, using this candidate as the first component. We then try to extend it to the direction opposite to that of the current dashed line in order to try to obtain a longer dash.

Due to the mutual object dependence, MDUS is designed to integrate a host of recognition processes into one procedure in a flexible order that represents the most likely dependence sequence. While recognizing a certain class of objects, the previously recognized objects are used as evidence, the strength of which is directly proportional to the probability of the object being a component of the hypothesized object. Key components, such as short thick bars for arrowheads, polylines for arcs and arrowheads for leaders, are used as the basic evidence, which is assigned high probability.

The recognition sequence can be modified by the system if some anticipated object has not been found where it is expected. For example, existence of a leader, a textbox and a wire indicates that this wire is the tail of the second leader. The second arrowhead is thus missing to complete the dimension set, and it should be sought for more rigorously in the neighborhood where it is expected to be found.

IV. System Functionality and Performance Evaluation

Currently implemented modules enable MDUS to convert paper engineering drawings, which are scanned and stored as raster files, into a combination of three recognition levels:

1) wire fragments;
2) graphic primitives;
3) higher level graphic objects.

The conversion results can be stored in IGES and/or DXF file formats and exported to any CAD systems that has a preprocessor for one of those formats. The following figures demonstrate current capabilities of MDUS applied to portions of real life drawings.

Fig. 7 shows the results of sparse pixel vectorization, arc segmentation, text segmentation, dash-dotted line detection and leader detection from a clean real life drawing. Figs. 8 and 9 show additional functions of the system, including text segmentation, leader detection, and hatching line detection. The light lines in Figs. 8(c) and 9(c) within four groups are detected hatching lines.

The sparse pixel vectorization is both time efficient and shape preserving. Typical drawings of size 1000×1000 pixels and medium complexity require about 3–5 s on an SGI Indy workstation. For clean and simple drawings, the pixel detection rate (defined as the ratio of the number of pixels which are originally black and also covered by the detected vectors and the total number of the original black pixels) is about 95%, while the pixel false alarm rate (defined as the ratio of the number of pixels which are originally white but covered by the detected vectors and the total number of pixels covered by the detected vectors) is less than 10%. For noisy drawings, the pixel detection rate is around 90% and the false alarm rate is up to 20%.

The text segmentation rate obtained on a sample of noisy, real life drawings is 94% [7]. The hatching line detection from Figs. 8 and 9 is 75% without any false alarm, as can be seen from the figures. Our dashed line detection algorithm won first place at the Dashed Line Detection Contest held during the First International Workshop on Graphic Recognition at the Pennsylvania State University, University Park, August 1995, achieving 100% recognition rate with just one false alarm [3], [9].

V. Summary and Future Work

We have described the object class hierarchy, data structures, algorithms, current implementation status and performance of the MDUS. The ultimate goal of MDUS is to automate as much as possible of the routine manual labor associated with automated CAD conversion, while leaving to the human operator decisions concerning ambiguous situations that the machine cannot reliably resolve on its own.

Even though the present implementation covers only the lexical level of graphic object recognition, it can already serve as a basic CAD conversion system with acceptable performance for many of the tasks in terms of both recognition rate and time efficiency. Near future developments include syntactic analysis to obtain geometry/annotation separation and 2-D view understanding. In parallel, we are also investigating the domain of 3-D reconstruction from dimensioned 2-D views, such that when complete 2-D understanding is obtained, it can be directly used in the 3-D understanding module.

Acknowledgment

The authors wish to thank the anonymous referees for their valuable comments.

References

[1] D. Dori, Y. Liang, J. Dowell, and I. Chai, "Sparse pixel recognition of primitives in engineering drawings," *Mach. Vision Applicat.*, vol. 6, pp. 69–82, 1993.
[2] D. Dori, "Vector-based arc segmentation in the machine drawing understanding system environment," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17, pp. 1057–1068, Nov. 1995.
[3] B. Kong, I. T. Phillips, R. M. Haralick, A. Prasad, and R. Kasturi, "A benchmark: Performance evaluation of dashed-line detection algorithms," *Graphics Recognition—Methods and Application, Lecture Notes in Computer Science*, R. Kasturi and K. Tombre, Eds. Berlin, Germany: Springer, 1986, vol. 1072, pp. 270–285.
[4] W. Liu, D. Dori, Z. Tang, and L. Tang, "Object recognition in engineering drawings using planar indexing," in *Proc. Int. Workshop Graphics Recognition (GREC'95)*, Penn. State Univ., University Park, 1995, pp. 53–61.
[5] ftp://ftp.technion.ac.il/pub/supported/ie/dori/ MDUS/sgimdus.gz and sunmdus.gz
[6] D. Dori and W. Liu, "The sparse pixel vectorization algorithm and its performance evaluation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 21, pp. 202–215, Mar. 1999.
[7] D. Dori and W. Liu, "Vector-based segmentation of text connected to graphics in engineering drawings," in *SSPR96*, 1996, pp. 322–331.
[8] D. Dori and W. Liu, "Stepwise recovery of arc segmentation in complex line environments," *IJDAR*, vol. 1, no. 1, pp. 62–71, 1998.
[9] D. Dori, W. Liu, and M. Peleg, "How to win a dashed line detection contest," *Graphics Recognition—Methods and Application, Lecture Notes in Computer Science*, R. Kasturi and K. Tombre, Eds. Berlin, Germany: Springer, vol. 1072, pp. 286–300, 1996.
[10] W. Liu and D. Dori, "Recognition of hatching lines in engineering drawings," in *Proc. 13th Israeli Symp. AI, CV, and NN*, Tel Aviv University, 1997.
[11] P. Baffes, *NETS—Neural Network Simulator*, Software Technology Branch NASA, Johnson Space Center, ftp://ftp.technion.ac.il /pub/unsupported/dos/simtel/neurlnet/nasanets.zip