Bi-level Document Image Compression using Layout Information

Stuart J. Inglis and Ian H. Witten

Computer Science, University of Waikato, Hamilton, New Zealand
email: {singlis, ihw}@cs.waikato.ac.nz

1 Introduction

Most bi-level images stored on computers today comprise scanned text, and their number is escalating because of the drive to archive large volumes of paper-based material electronically. These documents are stored using generic bi-level image technology, based either on classical run-length coding, such as the CCITT Group 4 method, or on modern schemes such as JBIG that predict pixels from their local image context. However, image compression methods that are tailored specifically for images known to contain printed text can provide noticeably superior performance because they effectively enlarge the context to the character level, at least for those predictions for which such a context is relevant. To deal effectively with general documents that contain text and pictures, it is necessary to detect layout and structural information from the image, and employ different compression techniques for different parts of the image. Such techniques are called *document image compression* methods.

A scheme that extracts characters from document images and uses them as the basis for compression was first proposed over twenty years ago by Ascher and Nagy (1974), and many other systems have followed (Pratt *et al.*, 1980; Johnsen *et al.*, 1983; Holt, 1988; Witten *et al.*, 1992, 1994). The best of these methods break the compression process into a series of stages and use different techniques for each. Whereas the images of the actual characters must necessarily be compressed using an image compression model, their *sequence* can be better compressed using a standard text compression scheme such as PPMC (Cleary and Witten, 1984; Bell *et al.*, 1990).

The present project extends these systems to include an automatic discrimination between text and non-text regions in an image. If a particular area of an image is known to be a photograph, for example, a more appropriate compression model can be used. In order to do so, the document must first be divided into apparently homogeneous regions or *zones*; in order to do that, it is necessary to make estimates of the inter-line and inter-character spacing. The analysis of document layout has been investigated previously by several researchers (e.g. Casey *et al.*, 1992; Nagy *et al.*, 1992; Pavlidis and Zhou, 1992; O'Gorman, 1993); but the methods often depend on particular numeric values or rules that relate to the resolution, print size and typographical conventions of the documents being analyzed, and on hand-tailored heuristics to differentiate blocks of text from drawings and halftones. Our emphasis is on completely automatic processing that does not require any manual intervention or tuning, except for the fact that it may be trained on representative examples of documents that have been segmented and classified manually.

This paper describes the document image compression process, and gives an account of how the zones in a document are located and how they are classified into text and non-text. We provide an extensive comparison of the compression achieved by this new system with JBIG (CCITT, 1993),

the standard for bi-level image compression, on one thousand document images.[1] We begin by mentioning a large publicly available corpus of scanned images that was used for the work, in order that it can be replicated. The resulting document image compression system achieves lossless compression ratios 25% greater than JBIG, and very slightly lossy ratios that are nearly three times as great as JBIG.

2 Image Corpus

Previous researchers on document image compression have obtained their results using the eight standard CCITT facsimile test images (Hunter and Robinson, 1980), or with unavailable private image collections. We have selected a large publicly-available database of 979 bi-level images, published by the University of Washington, as the image corpus for the experiments (University of Washington, 1993).

The images in the UW ENGLISH DOCUMENT IMAGE DATABASE I are scanned at 300 dpi from 160 different journals. Many have small non-zero skew angles, and in some cases text is oriented vertically instead of horizontally. Each one has been manually segmented into zones; there are a total 13,831 zones in the database or an average of 14 per image. For instance, a particular image may be divided into six text zones, one halftone zone, and one table zone. The corpus contains a total of twelve zone classes. Text zones, which are by far the most frequent (88% of all zones), represent paragraphs, bibliography sections, page numbers and document titles; other zone types include line drawings, halftones, math, and rules.

3 Document Image Compression

Document image compression works by building a dictionary of repeated patterns, usually representing characters, from an image and replacing them by pointers into the pattern dictionary. Because most English characters contain only one connected area, we define a *component* as an eight-connected set of black pixels and use the component in all stages of the system.

The first stage is to extract the components from the image. The dictionary is built from these using a template-matching procedure. As the components are processed, the patterns belonging to each dictionary entry—that is, all instances of that character—are averaged. When all components have been processed, the dictionary contains an average, supposedly representative, example of each different component, and the components in the image can be represented as indexes into the dictionary.

This kind of compression, where a dictionary is created and indexes into it are coded, can be viewed as a type of (LZ78-style) macro compression. However, during the matching process that determines whether or not two components are similar, matches are accepted even if the components are not completely identical. Thus when the image is reconstructed from the dictionary, there is noise around the edges of components. The difference between the reconstructed image and the original is known as the *halo* image. Of course, if exact matching were enforced, the dictionary would become almost as large as the number of components in the image, because the noise makes each component unique.

[1]This present version of the paper contains preliminary results for a subset of 200 images.

A simple model of the process consists of five steps:

- extract all components from the image;
- build a dictionary based on the components;
- compress the dictionary;
- compress the components using the dictionary;
- compress the halo to form a lossless image.

This model is described by Witten *et al*. (1992). In this paper we extend it to cater for zones of different types. Components are clustered into zones, and the zones classified into the types *text, non-text* and *noise*. Only the text components are used in the dictionary building stage. The halo of errors around the reconstructed characters is compressed as before. Now, however, there is a further image, called the *remainder*, which comprises the non-text regions (including noise).

The new model contains eight steps:

- extract all components from the image;
- group the components into zones;
- classify each zone into text, non-text or noise;
- build a dictionary based on the text zones;
- compress the dictionary;
- compress the text components using the dictionary;
- compress the remainder—the non-text and noise regions;
- compress the halo to form a lossless image.

We briefly describe each step. Figure 2 gives an example image and shows how it is broken down into parts.

COMPONENT EXTRACTION

Components are located by processing the image in raster-scan order until the first black pixel is found. Then a boundary tracing procedure is initiated to determine the width and height of the component containing that pixel. A standard flood-fill routine is called to extract the component from the image. As each component is extracted, several features, including the centroid and area, are determined and used later for comparing and classifying components.

DOCUMENT ZONE CLUSTERING

Components are grouped into zones by clustering them using a bottom-up technique known as the *docstrum* (O'Gorman, 1993). This involves finding the k nearest neighbors of each component, using the Euclidean distance between the components' centroids. Each component is linked with its neighbors to form a graph. In our experiments k=5 is used, because the five closest neighbors usually include two or three components in the same word, or the same line, and two components in adjacent lines.

Once the nearest neighbors are found, links in the graph between components that exceed a threshold are broken. This is a dynamic threshold, calculated directly from the docstrum, and is defined as the smaller of three times the most common component-within-line distance, and $\sqrt{2}$ times the average between-line distance. After the links whose lengths exceed this threshold have been broken, the transitive closure of the graph is formed. The resulting sub-graphs represent document zones.

ZONE CLASSIFICATION

To use the appropriate compression model, each zone's classification must be determined. Instead of developing an *ad hoc* heuristic to distinguish the zones, we use a machine learning technique to generate rules automatically for each possible type of zone. Machine learning methods learn how to classify instances into known categories. The process involves a training stage, which operates on pre-classified instances, followed by a testing stage, which uses the rules obtained during training on instances whose class is unknown. We use the well-known and robust C4.5 method (Quinlan, 1992), having determined in preliminary tests that it achieves consistently good results on this problem compared with other machine learning methods (Inglis and Witten, 1995).

Six features are calculated for each zone:

- zone density—number of components per unit area;
- component density—mean density of black pixels in the components' bounding boxes;
- aspect ratio—mean ratio of height to width of the components' bounding boxes;
- circularity—the square of the perimeter divided by the area, averaged over all components;
- separation—mode distance between a component and its nearest neighbor;
- run length—mean horizontal run length of the components' black pixels.

The 13,831 zones in the database images have been classified manually and comprise a total of 12,216 text and 1615 non-text zones. They were divided randomly in half to form training and test sets. The features for the training set, along with the manually-assigned class, text or non-text, were given to C4.5, which produced a decision tree. When this tree was used to classify the zones in the test set, 96.7% accuracy was achieved in replicating the manually-assigned classifications. (This result is averaged over 25 runs, using different random splits each time.) Figure 1 shows a simple 10-node decision tree that is generated for the simpler task of discriminating text regions from ones containing halftone. Trees for the text *vs.* non-text discrimination are three or four times as large as this.

For example, Figure 2(a) shows an image in its original form, and Figure 2(b) shows the text zones that are identified by segmenting it into zones as described above and classifying them automatically. The training data for the classifier comprises image zones that were determined manually: these are not the same as the zones generated automatically. In general, the automated procedure finds more zones, although this does depend on the distance threshold.

DICTIONARY GENERATION

The dictionary is built using only the zones that are classified as text. The dictionary-building operation relies on the ability to compare two different components to see whether they represent the same character. This comparison operation is known as *template matching*. The template matcher aligns the components by their centroids and processes them pixel by pixel. The method that we use rejects a match if the images differ at any point by a cluster of pixels of greater than a certain size; in order to detect mismatches due to the presence of a thin stroke or gap in one image but not the other, another heuristic is used (Holt, 1988).

The easiest way to build the dictionary is to store the first example of each different component as a reference example for subsequent comparisons. However, compression is adversely effected if the first component is a poor representative of its class. Consequently a running average is made of all examples of each different component during the dictionary-building process. As components are

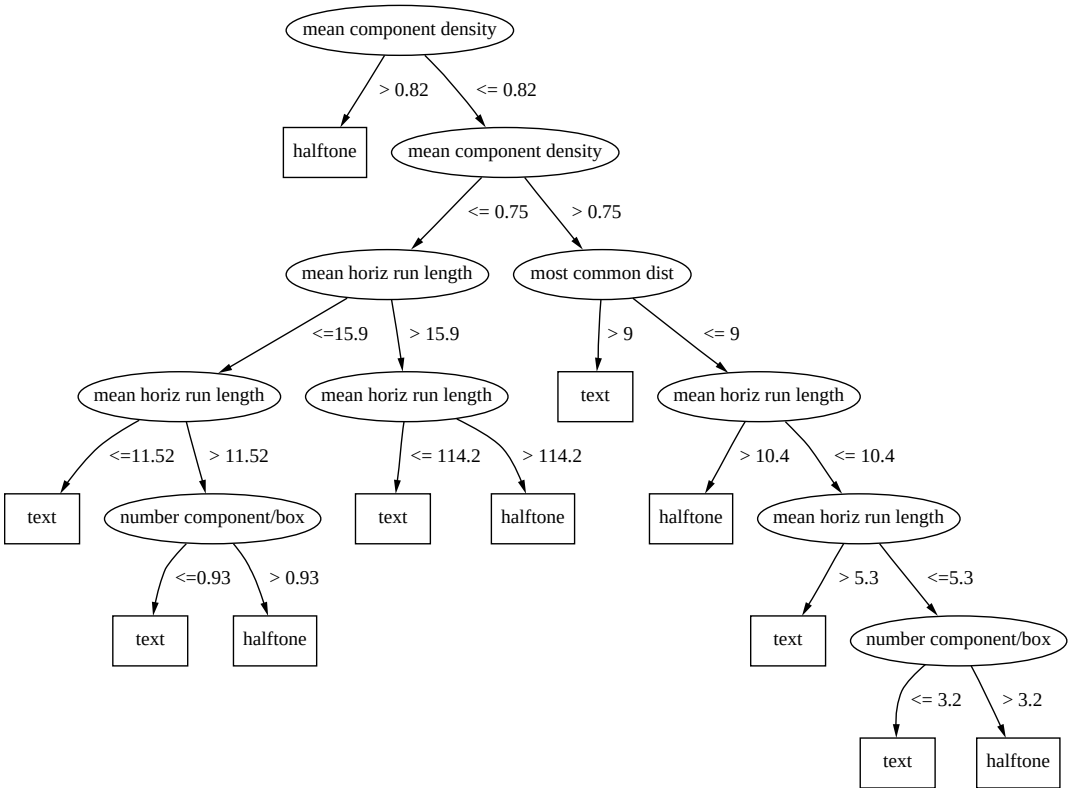

Figure 1: Decision tree discriminating text from halftones

extracted from the image, they are checked against the dictionary to determine whether a sufficiently similar component has been seen before. If not, the new component is added to the dictionary. Otherwise, if the component did match a dictionary element, that element is updated by incorporating the new component into the running average.

Because of this averaging process, it is possible that once the dictionary has been finalized a component might not match any of the elements in the dictionary. Therefore after the building process, each component is re-checked against the dictionary and added if necessary. This happens rarely: it accounts for less than 2% of dictionary entries.

DICTIONARY COMPRESSION

The dictionary elements are compressed using Moffat's (1991) two-level context method. This scheme gives the highest compression of all pixel-context based methods. The compression model is initialized at the start of compression, and the statistics are carried over from one dictionary element to the next.

TEXT COMPONENT COMPRESSION

The components in each text zone are processed sequentially. First, their indexes into the dictionary are compressed using PPMC (Bell *et al.*, 1990). Then the relative position of each component with respect to its predecessor in the zone is calculated. These offsets are compressed using a three-tier model, using the first model if possible, the second one if the statistics required by the first are not available, and the third as a last resort. The first model contains counts of offset

values with respect to a dictionary element; that is, the offsets are character-specific. The second accumulates all offset information for every dictionary element; a generic model for all characters. Finally, if neither of these first two models can be used, the offset is gamma encoded (Elias, 1975).

At this stage the dictionary has been compressed, the component order is known, and the component positions in the image are known. Thus the recipient can form a *reconstructed* image using the average components from the dictionary.

REMAINDER COMPRESSION

The *remainder image* consists of the non-text and noise components. An example is shown in Figure 2(d). As the remainder is independent of the previous stages, the other images are of no help in coding it. Therefore it is compressed using the two-level context method. If the system is processing an image in lossy mode, this is the end of the compressed stream.

HALO COMPRESSION

As there is only one example of each component in the dictionary, pixel errors occur around the borders of characters when the reconstructed image is compared with the original. These errors constitute the halo image, and Figure 2(c) shows an example—which is the XOR difference between Figure 2(a) and 2(b). It is possible from the halo image to make out the outlines of characters, and in some places actually to read them. Because the reconstructed image has already been coded, it makes sense to encode the halo with respect to it. In fact, since the entire reconstructed image is available, a context that is centered on the pixel to be coded can be used. This has been coined "clairvoyant" compression (Witten *et al.*, 1992). In practice, it turns out that coding the original image using a clairvoyant context in the reconstructed image leads to higher compression than if the halo is coded using the same context, because the halo is complicated by its exclusive-or nature. The compressed stream is now lossless.

4 Experiments

In this section, the document image compression scheme described above is compared with two pixel-context models: JBIG[2] and Moffat's (1991) two-level compression scheme.

The JBIG implementation gave an overall compression ratio of 17.5:1 over the corpus of images, while the two-level scheme achieved 19.1:1. The document image compression scheme was run in two different modes, with and without inclusion of the halo. The lossless mode, with the halo, achieved 22.1:1 and the lossy mode 49.4:1. These results are summarized in Table 1. It should be noted that the lossy mode guarantees a very good approximation to the original image: it is impossible for a block of more than four incorrect pixels to occur. Arguably, the lossy image is actually cleaner than the original because all examples of each letter are exactly the same.

The times taken to compress and decompress, measured on a SUN SparcCenter 1000, are shown in Table 2.[3] The document compression scheme is asymmetrical in that dictionary building and

[2]We have used an implementation of JBIG written by Markus Kuhn (1995), because it is the most complete of the two publicly available implementations.

[3]The times quoted for the document compression scheme are somewhat pessimistic because no serious attempt has been made to optimize the code.

Scheme	Total size	Compression ratio
JBIG	11 841 564	17.5
Two-level	10 854 655	19.1
Lossless	9 394 914	22.1
Lossy	4 202 827	49.4

Table 1: Compression figures for the different schemes

Scheme	Compression time (min:sec)	Decompression time (min:sec)
JBIG	0:22	0:23
Two-level	1:00	1:04
Lossless	5:00	2:00
Lossy	3:50	0:20

Table 2: Compression time per image for each scheme

Header	Dictionary	Symbols	Offsets	Halo	Remainder	Footer
0.1%	5.8%	5%	4%	60%	25%	0.1%

Table 3: Relative component sizes for document image compression

zone classification are not required when decompressing. This contrasts with JBIG and the two-level scheme, for which compression and decompression take about the same time. Lossless compression in the document-based scheme is fifteen times slower than JBIG, decompression is six times slower. In lossy mode, compression is ten times slower and decompression is slightly faster. Table 2 shows the full results for all schemes.

Due to the asymmetric nature of the document image compression system, it is most applicable to situations where lossy decompression is time-critical, such as on-line searches, or when compression time is unimportant, such as in CD-ROM mastering.

Table 3 shows the contribution of the different kinds of information in lossless document image compression to the total compressed size, for a typical example image. The halo is over half the file size, which explains why the lossy compression ratio is more than twice as great the lossless one.

5 Conclusion

We have extended a document image compression system to include layout and zone classification information, and made it robust and suitable for general-purpose use by removing the necessity for manual intervention or tuning to particular typographical conventions. The resulting compression scheme yields a 25% improvement over JBIG. Using the system in lossy mode where the halo around the characters is not encoded, the compression ratio becomes nearly three times that for JBIG. The new scheme is asymmetric in that decompression is much faster than compression, but both are considerably slower than JBIG—except for lossy decompression, which is slightly faster.

References

Ascher, R.N. and Nagy, G. (1974) “A means for achieving a high degree of compaction on scan-digitized printed text.” *IEEE Transactions of Computers*, vol. 23, no. 11, pp. 1174–1179, November.

Bell, T.C., Cleary, J.G. and Witten, I.H. (1990) “Text compression”, Englewood Cliffs, NJ, USA: Prentice Hall

Casey, R. Ferguson, D., Mohiuddin, K. and Walach, E. (1992) “Intelligent forms processing system.” *Machine Vision and Applications*, vol. 5, pp. 143–155.

CCITT (1993) “Progressive bi-level image compression." *Recommendation* T.82; also appears as ISO/IEC standard 11544.

Cleary, J.G. and Witten, I.H. (1984) “Data compression using adaptive coding and partial string matching,” *IEEE Trans. on Communications*, COM-32, pp. 396–402.

Elias, P. (1975) “Universal codeword sets and representations of the integers.” *IEEE Trans Information Theory*, vol. IT-21, pp. 194–203.

Holt, M.J. (1988) “A fast binary template matching algorithm for document image data compression." in *Pattern Recognition*, J. Kittler Ed. Berlin, Germany: Spring Verlag.

Hunter, R. and Robinson, A.H. (1980) “International digital facsimile coding standard.” *Proc IEEE*, vol. 68, no. 7, pp. 854–867.

Inglis, S. and Witten, I.H. (1995) “Document zone classification using machine learning.” *Proc. Digital Image Computing: Techniques and Applications*, Brisbane, Australia, December.

Johnsen, O., Segen, J. and Cash, G.L. (1983) “Coding of two-level pictures by pattern matching and substitution." *Bell System Tech. Jour.*, vol. 62, no. 8, pp. 2513–2545, May.

Kuhn, M. (1995) URL: http://wwwcip.informatik.uni-erlangen.de/user/mskuhn

Moffat, A. (1991) “Two level context based compression of binary images." in *Proc. IEEE Data Compression Conference*, Los Alamitos, CA, USA: IEEE Computer Society Press, pp. 328–391, April.

Nagy, G., Seth, S. and Viswanathan, M. (1992) “A prototype document image analysis system for technical journals.” *IEEE Computer*, pp. 10–22; July.

O’Gorman, L. (1993) “The document spectrum for page layout analysis." *IEEE Trans Pattern Analysis and Machine Intelligence*, vol. PAMI-15, no. 11, pp. 1162–1173, November.

Pavlidis, T. and Zhou, J. (1992) “Page segmentation and classification.” *CVGIP: Graphical Models and Image Processing*, vol. 54, no. 6, pp. 484–496.

Pratt, W.K., Capitant, P.J., Chen, W.H., Hamilton, E.R. and Wallis, R.H. (1980) “Combined symbol matching facsimile data compression system." *Proc. IEEE*, vol. 68, no. 7, pp. 786–796, July.

Quinlan, J.R. (1992) C4.5: *Programs for Machine Learning*, Morgan Kaufmann.

University of Washington English Document Image Database I, (1993) Seattle, WA, USA.

Witten, I.H., Bell, T.C., Harrison, M.H., James, M.L. and Moffat. A. (1992) “Textual image compression.” *Proc. DCC*, edited by J.A. Storer and M. Cohn, pp. 42–51. IEEE Computer Society Press, Los Alamitos, CA.

Witten, I.H., Bell, T.C., Emberson, H., Inglis, S. and Moffat, A. (1994) “Textual image compression: Two-stage lossy/lossless encoding of textual images." *Proc. IEEE*, vol. 82, no. 6, pp. 878–888, June.

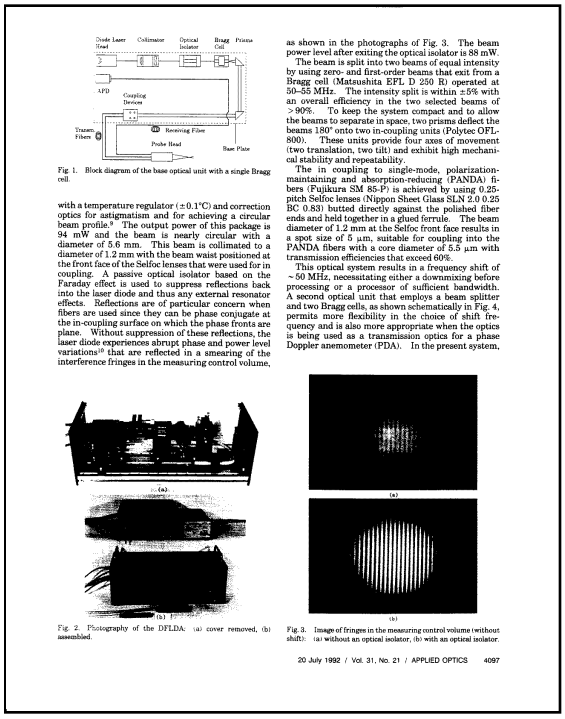

Figure 2(a): Original Image

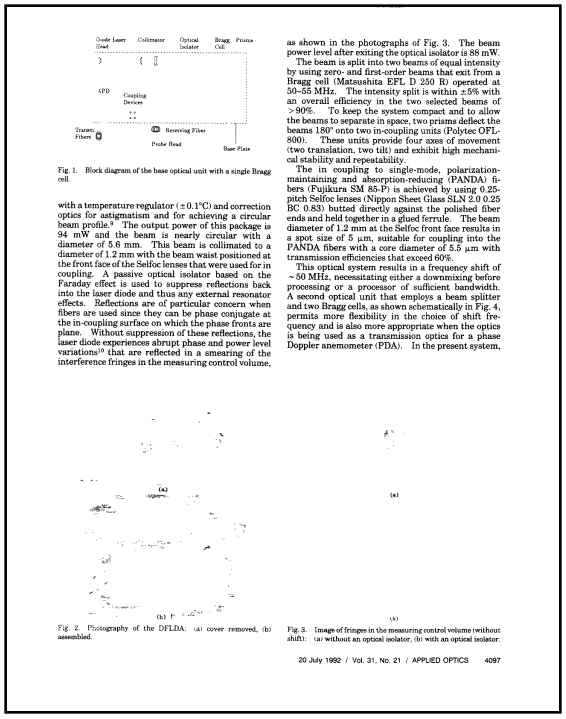

Figure 2(b): Text components

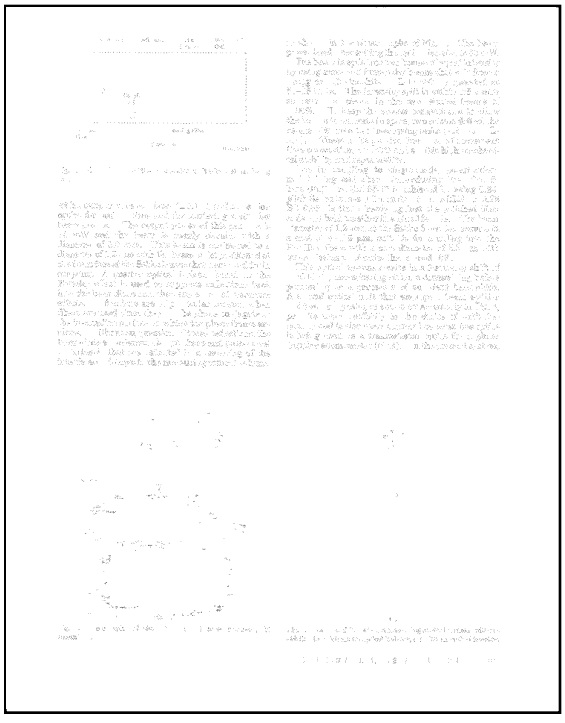

Figure 2(c): Halo image

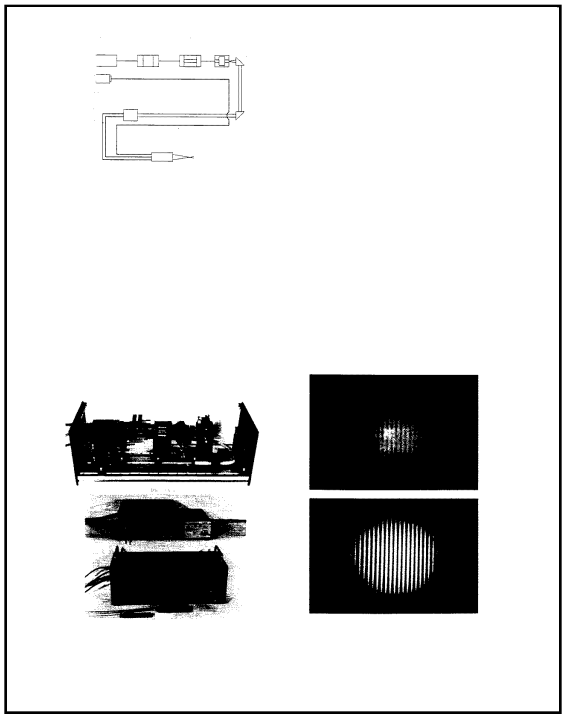

Figure 2(d): Remainder image