University of Surrey

Department of Computing

A new multi-set modulation technique for increasing hiding capacity of Binary Watermark for Print & Scan Processes

C. Culnane, H. Treharne, A.T.S. Ho

May 2006

Computing Sciences Report

CS-06-02

A new multi-set modulation technique for increasing hiding capacity of Binary Watermark for Print & Scan Processes

C. Culnane, H. Treharne, A.T.S. Ho

Department of Computing, School of Electronic and Physical Sciences,
University of Surrey, Guildford, Surrey, GU2 7XH
csm1cc@surrey.ac.uk

Abstract. In this paper we propose a multi-set modulation technique to increase the hiding capacity within a binary document image. As part of this technique we propose an Automatic Threshold Calculation and Threshold Buffering, Shifted Space Distribution and Letter Space Compensation technique. The Automatic Threshold Calculation is used to distinguish word spaces from letter spaces. The Threshold Buffering is used to reduce the chance of misinterpretation of spaces during the detection phase, following printing and scanning. The Shifted Space Distribution and Letter Space Compensation techniques robustly embed a watermark into the binary document image. The Automatic Threshold Calculation has been shown to be successful in identifying word spaces for different types of fonts and font sizes. The combination of the Shifted Space Distribution, Letter Space Compensation and Threshold Buffering techniques have been shown to create a watermark that is robust to printing and scanning.

1 Introduction

Several approaches have been proposed which examine the digital watermarking of binary documents. Low and Maxemchuk proposed a method of line and word shifting in [1]. Low et al. [2] compared two methods of line and word shifting for binary text documents whilst Wu et al. [3] examined the full range of binary documents. Ho et al. proposed a method of pixel flipping in [4]. Koch and Zhao [5] proposed a method of embedding data into a binary image based on the percentage of pixels that were white in a block of an image. However, the watermark was not robust to print and scan operations. The potential for digital watermarks is well documented [6], but text documents, more so than other documents, are often printed and transferred from the digital realm to the analog, by way of a printer. The analog copy can then be transferred back into the digital realm with the use of a scanner. There is a need for a watermark that is robust to these print and scan operations. A print and scan resilient system is proposed in [7], but this was for images and not binary document images. One of the notable methods for binary document images was proposed by Zou

& Shi [8] in which they identified a way of embedding bits in the lines of text using inter-word space modulation. The fundamental concept of their work is to divide the spaces, between words in a line, into two sets. The total space in these sets is adjusted to create a detectable difference. The difference between the two sets indicates whether a '0' or a '1' is embedded. In [8] they state that their approach is robust to printing the watermarked document, photocopying it ten times and then scanning it. Their approach is limited to providing 1 bit of embedding space per line of text.

Creating a watermarking scheme robust to printing and scanning presents a number of different challenges. Traditionally, in binary digital watermarking, a watermark is generated by flipping pixels to change the image. This is not a suitable method to use in a print and scan system. When a document is printed and then scanned the pixel locations change, so recovering such a watermark is extremely difficult. Furthermore, during the print and scan process noise and distortion is introduced, and the robustness of the watermark must be demonstrated in the presence of these difficulties.

In this paper we present an approach which improves upon the method presented by Zou and Shi [8]. Our contribution is to increase the capacity of the system by introducing a Multi-set Modulated Word Space technique whereby the capacity is not fixed at 1 bit per line, but is dynamically calculated based on the content of that line. In order to maintain a good level of robustness a method of Automatic Threshold Calculation is proposed. This is complemented by a Threshold Buffering technique. A Shifted Space Distribution method is proposed for distributing space between sets, whilst a Letter Space Compensation technique is used to ensure the line stays the same overall length. The benefit of the proposed approach is to gain a greater capacity whilst still maintaining robustness to print and scan. The greater capacity of the watermark could potentially be used for authenticating binary images and documents.

2 Multi-set Modulated Word Space

In this section we present our new Multi-set Modulated Word Space technique. As part of this technique we propose the Automatic Threshold Calculation, Threshold Buffering, Shifted Space Distribution and Letter Space Compensation techniques to improve the hiding capacity and handling of different fonts.

2.1 Concept

The basic concept takes the principle idea in [8] and expands it further, by having multiple pairs of sets in one line. Word spaces can be defined as the white space between adjacent words. Letter spaces can be defined as the white space between adjacent letters in a word. The line is divided according to how many word spaces should be in each set, so the capacity of a line depends on its content. Word and letters spaces are distinguished with the use of an Automatic Threshold Calculation. Automatic Threshold Calculation is adequate for basic

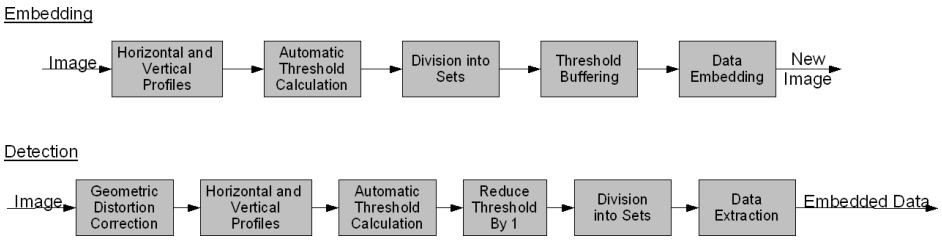

Fig. 1. Flow Diagram of System

embedding and detection. However, to handle the distortions caused by Print and Scan operations a buffer is created around the *threshold*. This is to avoid misinterpretation of letter and word spaces during detection. Figure 1 shows an overview of the watermark embedding and detecting system. Boxes that are common in both use the same algorithms in both processes.

2.2 Horizontal and Vertical Profiles

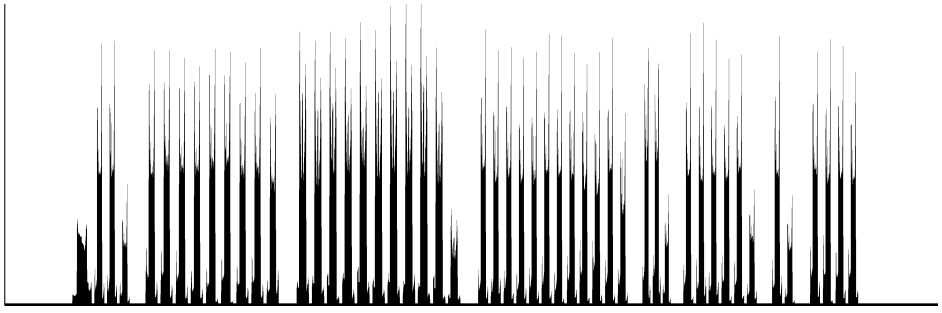

Fig. 2. Horizontal Profile

The standard way of splitting a document into lines of text, and those lines into letters is to use horizontal and vertical profiles, as seen in [8] and earlier described in [1]. They are graphs of the pixels present in the horizontal or vertical plane. The profiles are calculated by counting the number of black pixels present in the relevant plane.

Figure 2 shows a horizontal profile, which is used to distinguish the separate lines of text within the document. Zero values in the profile distinguish the individual lines. The position and size of lines can be found by analysing the zero values and the groups of sequential positive values.

For each line found using the horizontal profile, a vertical profile is calculated, as can be seen in Figure 3. The same process of counting black pixels is used. This time the groups of black pixels represent letters or words, whilst the zero values are the spaces. Once all the spaces have been found a *threshold* is calculated.

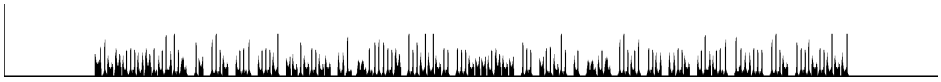

Fig. 3. Vertical Profile

2.3 Automatic Threshold Calculation

The Automatic Threshold Calculation algorithm aims to correctly distinguish between word and letter spaces, as defined above.

Initial attempts to use a static *threshold* were not successful. The static *threshold* makes the watermark more perceptible. This is due to letter spaces being incorrectly classified as word spaces. If this happens, the embedding process could increase the space between letters in a word. Too much space between letters could be noticeable to the human eye. It is therefore important to correctly classify the spaces found in a line.

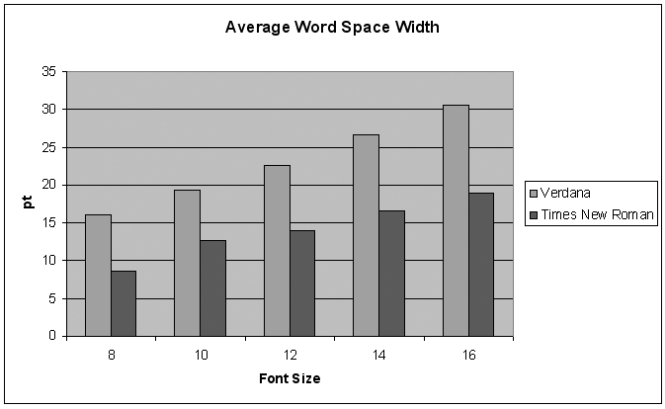

Fig. 4. Average Font Size

Figure 4 shows a graph of the comparison of average word space width for the Times New Roman and Verdana fonts. It is clear that there is a significant difference between the average word space width for the two fonts at the same point size. The difference is also not static and the variation increases the larger the font size. A similar effect was seen when comparing other fonts. This clearly

	Width (px)
S_1	13
S_2	13
S_3	15
S_4	13
S_5	15
S_6	14
A_1	3

Table 1. Table of Space Widths

shows the requirement for some form of Automatic Threshold Calculation, which can vary according to the content of a particular line.

The Automatic Threshold Calculation is based on the standard deviation of the spaces that are present in a line. Initially, the algorithm used a combination of the mean and the standard deviation. However, the mean was sensitive to the changes that could occur during the print and scan operation and this causes the *threshold* to vary and thus the composition of the sets to change. Hence, it could lead to errors in the calculation of the total space in a set and thus introduce detection errors. The standard deviation is, however, less sensitive to the changes and is therefore more stable. The Automatic Threshold Calculation is performed after profiling the document prior to embedding and detection.

2.4 Embedding

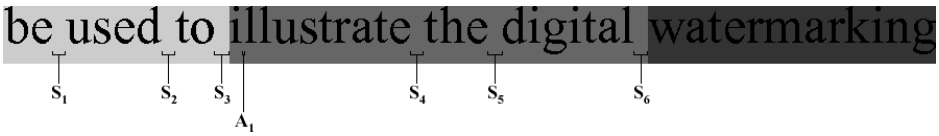

Fig. 5. Set Division

Division into Sets Figure 5 shows an extract from a line, approximately a third of the line is shown. The word spaces have been identified as S_1 through to S_6. Table 1 shows the pixels width of each of the annotated spaces. The significance of the shading is explained below. A_1 is a letter space that will be of interest after the embedding.

The *threshold* determines what will be considered a word space and what will be a letter space. The word spaces can then be divided into multiple pairs of sets. A default value of three spaces in each set was chosen. This was chosen because typically lines contain in the region of twelve spaces. As a result, a value of four did not increase the capacity enough, whilst a value of two did not provide the required level of robustness.

Only pairs of sets are created, with any spare word spaces being assigned to a spare group, situated on the right hand side of the line. This is signified by the darkest shading in Figure 5. Each pair of sets provides one bit of possible data embedding. But since we have multiple sets in each line we can embed more than 1 bit per line. Figure 5 shows the last two sets for a line, the complete line contained four sets. Let ϕ refer to a set whilst the subscript indicates the set number.

$$\phi_3 = S_1 + S_2 + S_3 = 41$$

$$\phi_4 = S_4 + S_5 + S_6 = 42$$

Where A_1 is a letter space that is part of ϕ_4. It is not included in the calculation of total set space, but is associated with ϕ_4.

Threshold Buffering Initial tests have shown that the minor changes occurring between letter and word spaces during printing and scanning results in spaces being interpreted differently during embedding and detection. To avoid this, and to increase the robustness, a system of creating a buffer around the *threshold* was added.

The goal of this system is to create word spaces that are greater than or equal to the sum of *threshold* and *thresholdBuffer*, and to ensure that letter spaces are strictly less than or equal to the *threshold* minus the *thresholdBuffer*. By expanding the word spaces and contracting the letter spaces the distinction between letter and word spaces becomes clearer. For each set we determine all the word spaces that need expanding. We also determine all the letter spaces that require contracting. Any spare space obtained by contracting the letter spaces is distributed evenly to the word spaces requiring expansion. At this point if all word spaces are greater than or equal to the sum identified above an appropriate buffering has been achieved. (Note that any excess spare space is distributed evenly amongst all word spaces in a set.) If more space is needed to expand the word spaces, further reductions are made to the letter spaces. (Note that no letter spaces are reduced below 1px.) Only letter spaces from within a set are used to create the buffer. Letter spaces from other sets and the spare set are not used. This is to reduce the inter-set dependency and because there is no guarantee that a spare set will exist. In situations where there is not enough spare letter space in a set, the maximum buffer possible, given the available spare space, is created.

Data Embedding The process of data embedding is based on the concepts presented in [8]. Data is embedded on a line by line basis within the document by making a detectable difference in the total size of the two sets. The detectable difference that should be present between two sets is referred to as the embedding strength. This is achieved by making changes to the spaces of the individual sets and then we achieve the desired change in total set size, thus creating a detectable difference. Using Figure 5 as an example, recall that $\phi_3 = 41$ and $\phi_4 = 42$. The parameters for the line in Figure 5 are as follows:

– *threshold* = 10
– *thresholdBuffer* = 3
– Embedding Strength (ϵ) = 6

To embed a '1' ϕ_3 must be at least ϵ greater than ϕ_4, conversely to embed a '0', ϕ_4 must be at least ϵ greater than ϕ_3. Clearly this is not the case in the example above and so ϕ_3 and ϕ_4 must be adjusted accordingly as follows:

$$\text{embed '1'} : \phi'_3 - \phi'_4 = \epsilon$$

$$\text{embed '0'} : \phi'_3 - \phi'_4 = -\epsilon$$

where ϕ'_3 and ϕ'_4 indicates the adjusted sets, which are the original sets augmented with half of the embedding strength, assuming that ϕ_3 and ϕ_4 are equal.

It is often the case that the two sets are not equal, as in Figure 5, and so some extra work is needed before we can augment the sets. Since ϕ_4 is one pixel bigger than ϕ_3 this difference must first be eradicated. The difference between them is divided by two and one set is made bigger whilst the other is made smaller by this amount. In this case the difference is only one pixel, which cannot be evenly split between the two. When this occurs the set to be enlarged is assigned the extra pixel (ϕ_3 in this case). It is important that the total length of the line is maintained, in order to reduce perceptibility. As a result we have:

$$\phi'_3 = \phi_3 + 1 + 3$$

$$\phi'_4 = \phi_4 - 1 - 3$$

Shifted Space Distribution In the above we identified the required changes to the spaces in a set. The Shifted Space Distribution technique makes the necessary changes but is careful to observe the threshold buffering and ensure the minimum perceptibility of the watermark. The process of increasing a space width is easy, since there is no critical limit on increasing a space's width. However, the process of reducing a space width must be carefully controlled. Word spaces must never be reduced below the sum of the *threshold* and *thresholdBuffer*, in this case 13. This is to ensure spaces do not change from word spaces to letter spaces, which could cause errors in the detection of the watermark. This limit on the level of reduction potentially causes a problem of not being able to satisfy the required changes. For example by reducing word spaces S_5 and S_6 in ϕ_4 to 13 pixels each we achieve an overall reduction of three pixels for this set. However, a reduction of four pixels is required in order to make the required change.

Letter Space Compensation Having made the maximum reduction to the word spaces in a set, and if we have not established the required changes the Letter Space Compensation technique identifies letter spaces from that set which can be reduced. This allows us to use the amount of reduction from the letter spaces to increase the other set. In our example this means identifying two pixels in ϕ_4 (A_1 from Figure 6) to be transferred to ϕ_3. You cannot simply add pixels

to ϕ_3 and leave ϕ_4 as it is because we must maintain the length of the line. The rule of not reducing letter spaces below 1 pixel is used, as in Threshold Buffering, so that the threshold calculation is not affected during detection. The Letter Space Compensation technique ensures that the embedding strength is maintained across a pair of sets.

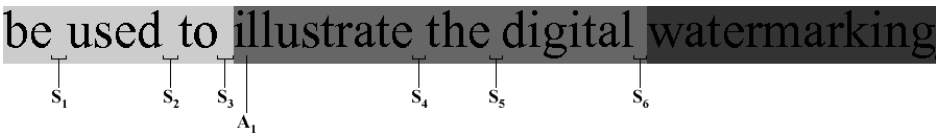

Fig. 6. Embedded

	Width (px)	Original Width (px)
S_1	15	13
S_2	15	13
S_3	16	15
S_4	13	13
S_5	13	15
S_6	13	14
A_1	1	3

Table 2. Table of New Space Widths Compared with Original

Figure 6 shows the annotated image with the data embedded. Table 2 shows the new space widths and the original widths. Note that the word spaces S_4, S_5 and S_6 have not been reduced below 13. Also note that the letter space, A_1 has been reduced from three to one to allow the transfer of the remainder to the operation to increase the size of ϕ_3. Figure 6 provides the following set sizes:

$$\phi_3 = 15 + 15 + 16 = 46$$

$$\phi_4 = 13 + 13 + 13 = 39$$

$$\text{where} \quad \epsilon = 46 - 39 = 7$$

The embedding strength (ϵ) is finally calculated to be seven, one more than required. This is due to the rounding operation when splitting a one pixel difference between the two sets.

2.5 Detection

Horizontal and vertical profiles are again created, as in the embedding process in Section 2.4. However, the main concerns are not Threshold Buffering or Shifted Space Distribution but detecting the watermark accurately with increased capacity and dealing with geometric distortion caused by printing and scanning.

Geometric Distortion We used a method for correcting geometric distortion based on the use of the horizontal profile and the calculation of the total amount of white space. It is assumed that there are only two points where the total amount of white space is at a maximum:

1. when the document is straight (0°)
2. when the document is rotated (180°)

Assuming that the document will be distorted by less than 1°, in either direction, a horizontal profile can be generated for each rotated value: -1°, 0°, and +1°. From this horizontal profile the total amount of white space is calculated by summing the parts of the profile that are at zero. Finding the maximum total white space value of the profiles determines the amount of rotation required to correct the distortion. The range can be widened if the image appears to be suffering from a greater degree of rotation. Obviously, rotations performed on high resolution images are computationally intensive. For example, it was found that an A4 page scanned at 600x600 pixels requires one gigabyte of memory to be rotated. However, the principle of the process holds true for smaller images. In our experiments, images are scaled to one half of their original size before they are rotated and profiled. Once the required amount of rotation is found the original is rotated by that amount.

During the initial experimentation described in Section 3.2, it was discovered that the amount of rotation an image suffered during the Print and Scan process could be as little as 0.25°. This small rotation had a detrimental effect on lines with a small font size. As a result a smaller rotation increment was needed, that gradually increased the precision of the rotation. For example, if the maximum amount of white space was found at 1°, the values from 0.0° to 2.0° would be tested at increments of 0.1°. If the maximum amount of white space was found at 0.2°, the values between 0.10° and 0.30° would be tested at increments of 0.01°.

The process of rotating the document to correct rotation can cause a degradation in the quality of the lettering. We noticed that noise, in the form of white pixels, is added to some letters. To counter this a Gaussian blur is applied to the image after the rotation has been completed. This is to mitigate against the chance of letters being split apart due to the distortion correction. If this occurs the horizontal and vertical profiles may incorrectly divide the document.

Division into Sets The *threshold* is reduced by one during the detection process, to allow for changes in spaces due to noise and distortion. This is possible due to the Threshold Buffering conducted during the embedding process. The value of one was chosen because it allows small distortions in word spaces with the minimum risk of misinterpreting letter spaces. The Threshold Buffering cannot be assumed to have created a buffer of three on every occasion, therefore a reduction of two or three would be more likely to cause misinterpretations. Other than this reduction the same process of dividing the sets is undertaken and creating multiple sets is the same as the embedding process.

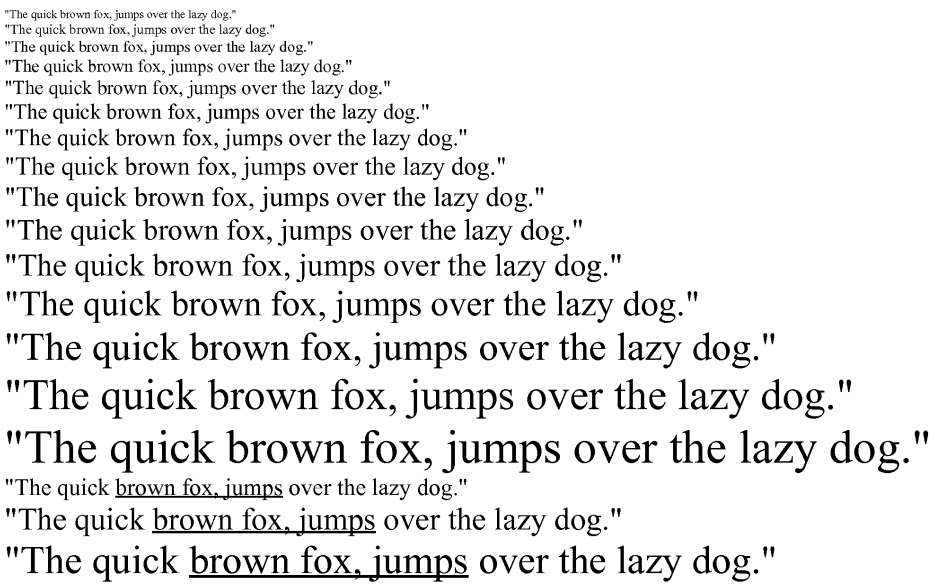

Fig. 7. Extract from a Sample Test Document for Times New Roman

Data Extraction The data is extracted by analysing the total word space differences between each set in each pair of sets in each line of a document. A '1' bit is present if set A is bigger than set B whilst a '0' bit is present if set B is bigger than set A.

3 Experimentation

This section evaluates the effectiveness of the Automatic Threshold Calculation, Threshold Buffering and Shifted Space Distribution techniques.

3.1 Automatic Threshold Calculation

The Automatic Threshold Calculation was tested using a sample document containing various font types, sizes and styles. This sample document was created in the following fonts: Arial, Arial Narrow, Comic Sans, Courier New, MS Sans Serif, Script, Tahoma, Times New Roman, and Verdana. Figure 7 shows an extract from a sample test document. One such document was created for all of the above fonts. Each line in the document represents a particular test, with each test being conducted on multiple font sizes. The following tests are contained within the document: font size, partial underlining, partial bold styling, two different fonts in a line, no spaces in a line, two different font sizes in a line and a line of single letter words. The partial underlining test aimed to test the effect of having part of the line underlined. It should be noted that the Automatic Threshold Calculation did not divide the underlined section, it viewed it as a

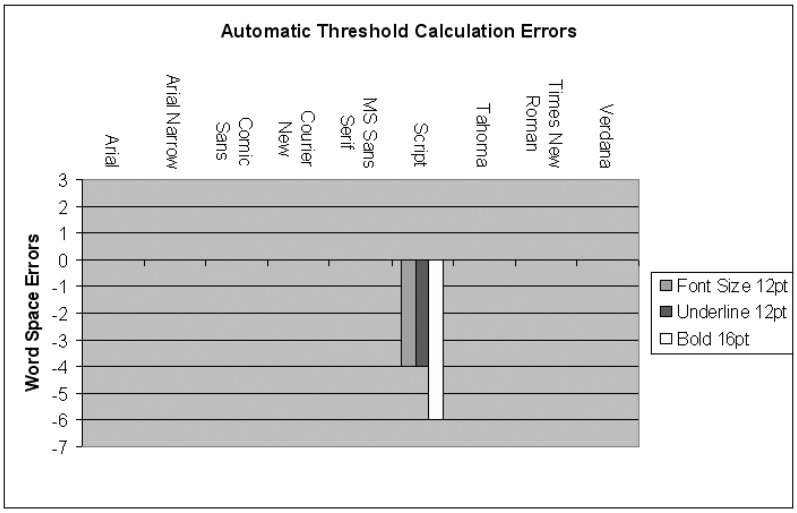

Fig. 8. Threshold Errors for Font Size, Underline, and Bold Test

single long word. Tests on no spaces and single letter words in a line were chosen as extreme tests and were not expected to be successful.

Each document was processed using the Automatic Threshold Calculation and the number of word spaces detected was recorded. The documents were also manually analysed to record a benchmark of the correct number of word spaces that should be detected. From these we could analyse the number of errors. A negative comparison indicates that too few word spaces were detected, whilst a positive value indicates too many word spaces were detected. The positive and negative errors are reflected on the y axis as illustrated in Figures 8 and 9, respectively.

Figure 8 shows an extract for the results for font size, underlining and bold styling. We focus on three font types because they reflected the overall results for all font sizing, underlining and bold styling. All the fonts, except Script, handled the styling and sizes of fonts correctly. The Script font is a handwriting style font that joins some letters together. As a result, there are fewer letter spaces. This makes it difficult to distinguish between letter and word spaces. The algorithm is designed so that it is better to detect fewer word spaces at the cost of capacity than to incorrectly detect letter spaces as word spaces at the cost of robustness.

Overall the Automatic Threshold Calculation dealt very well with the challenges of font size and different fonts. We have attempted some extreme tests which stress the algorithm and identify ways in which the algorithm can be improved under these conditions. In most cases it also successfully handled multiple font types in a line and partial styling. Arial and Times New Roman both had single errors, but the rest, other than Script, were successful in handling the multiple font types and partial styling.

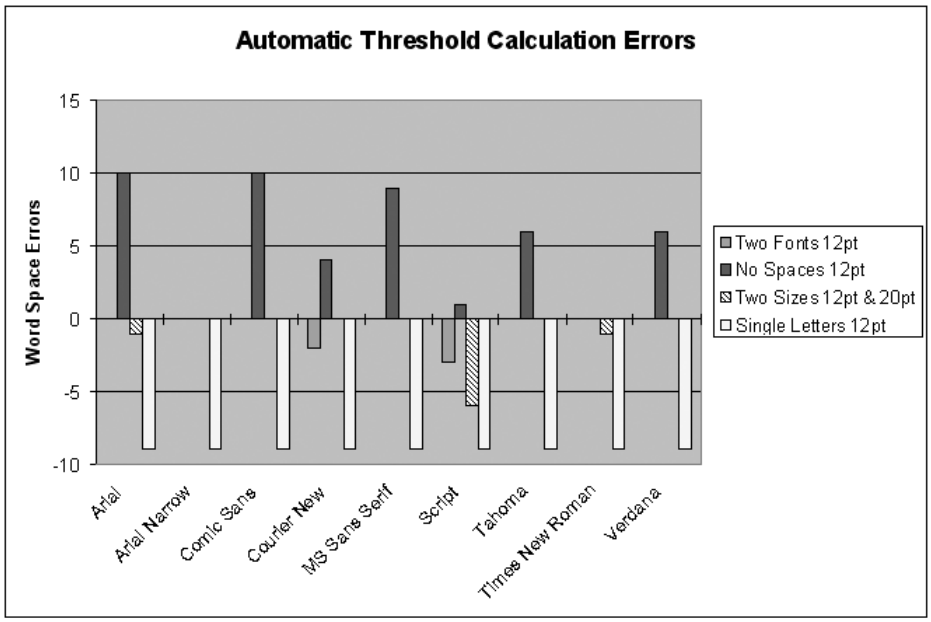

Fig. 9. Threshold Errors for Complex Tests

Figure 9 shows an extract of the results in which a line is made up of two fonts, two font sizes, no spaces and single letter words. Again these particular results were chosen because they were representative of their class. Most of the standard document fonts, with two different fonts in a line, were successfully handled, with the exception of Courier New. The Script font again had problems in all areas.

It is interesting to note that, all the fonts, except Arial Narrow and Times New Roman had errors with the no spaces test. In practice the algorithm uses the rule that if the standard deviation of all the spaces is below 1.5 then the *threshold* is set to be the maximum space width, giving rise to no word spaces being found. In practice it is unlikely that such a sentence would appear without a single space.

All the fonts failed the single letter test. The regularity of the spaces means that the standard deviation is always low, resulting in the use of the above rule and thus the inability to recognise word spaces. This results in an undue reduction in capacity. It is not a major concern that the Automatic Threshold Calculation failed this test, since it is highly unlikely that a line would contain just single letters separated by spaces.

3.2 Watermarking

Test Setup The test document was an A4 page of text. A copy was created for each of the fonts tested in section 3.1, all having a font size of 12pt. Each was saved as a PDF file and converted to a PNG file at 600dpi. The watermarked documents were printed on a HP PSC 2110 in Normal mode. The printed documents were scanned on the same HP PSC 2110 at various resolutions in Black

& White mode. The automatic straighten and colour adjustment were switched off.

Watermark ID	Watermark
A	WATERMARKED!
B	THE EXAMPLE!
C	ZZZZZZZZZZZZ

Table 3. Watermarks and ID's

Experimentation Table 3 contains the watermarks and their respective ID's. Each watermark was embedded in a copy of each document in each font, giving a total of 27 documents. The data was embedded as a binary string, using 8-bits per character. Where a document did not have the capacity to fit the entire watermark, the maximum to the nearest letter was embedded.

		Bit Error Rate								
		150dpi			**300dpi**			**600dpi**		
Font	**Capacity (bits)**	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
Arial	88	9	13	19	5	5	4	1	0	1
Arial Narrow	99	12	14	57	2	0	2	2	3	1
Comic Sans	69	27	19	28	1	21	1	0	0	0
Courier New	52	2	3	4	2	2	0	2	0	2
MS Sans Serif	90	16	10	16	1	3	5	0	1	0
Script	72	29	39	31	38	35	35	30	35	35
Tahoma	82	6	6	6	0	3	3	2	1	1
Times New Roman	90	45	10	28	1	13	34	0	3	1
Verdana	68	22	20	22	0	0	7	0	0	0

Table 4. Print & Scan Results

Table 4 shows the bit error rate (BER) results from the documents being scanned at 150, 300, and 600 dpi. These results are illustrated in Figure 10. The BER was calculated by comparing the binary string detected with the one embedded. There were occurrences of fewer sets being detected during the detection phase from the embedding phase. In these situations, if it was clear where the set had been lost, the detected string was padded with the inverse bit of the original. This would result in a lost set being counted as a bit error, without having a significant impact on all the remaining bits in the string, that would otherwise have been shifted one place to the left. The results show that it is possible to achieve a zero bit error result. They also demonstrate that in most cases 150dpi is not a high enough resolution by which to scan the document. The best results were obtained at 600dpi. The Script font did not perform well, although this was to be expected since it performed badly in the Automatic Threshold Calculation tests. The Verdana font performed well with five out of the six test above 150dpi resulting in no bit errors. Courier New provided the most consistent results, and

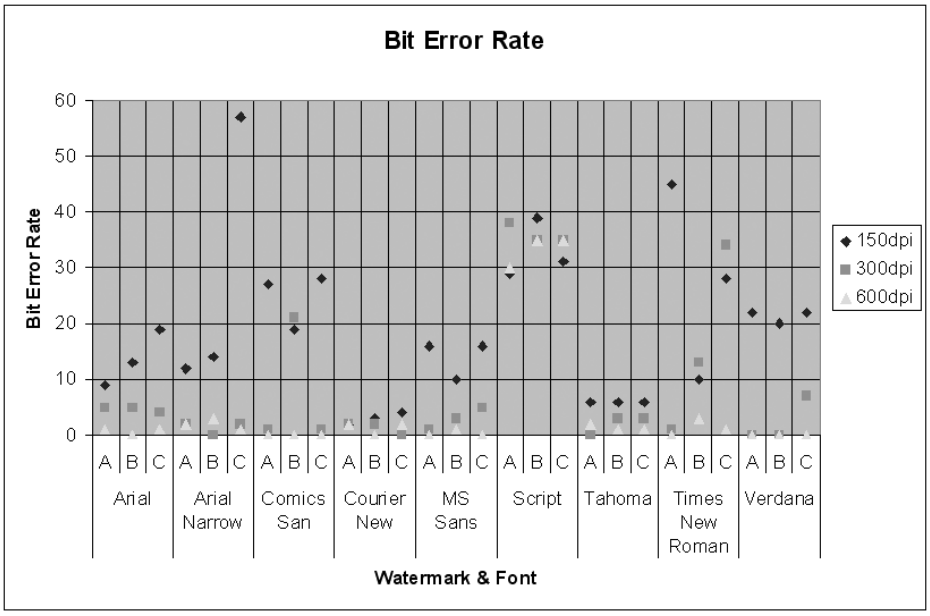

Fig. 10. Bit Error Rates

interestingly coped well at 150dpi. The Tahoma font also performed better than most at 150dpi and consistently over the other resolutions.

Overall the results demonstrate that the principle of the system works and with some further improvements it may be possible achieve even more zero bit errors.

4 Conclusion

The proposed method has been shown to provide a greater capacity whilst still being robust to print and scan. The Automatic Threshold Calculation has shown to be useful in handling multiple fonts and different font sizes. Both the Tahoma and Comic Sans fonts correctly classified spaces with zero errors, except in the extreme tests. The Shifted Space Distribution has produced results which appear imperceptible to the human eye. The Letter Space Compensation technique has improved the robustness of the watermark. Without this technique we would not have been able to maintain the embedding strength. At 600dpi both the Comic Sans and Verdana font were able to detect the watermark with a zero BER.

The capacity in [8] is restricted to embedding one bit per line. The capacity in our approach is dependent on the content of each line and can vary from one line to the next. For example, a line in a smaller sized font will have more word spaces, and therefore will have a higher capacity. The capacity can be varied by adjusting how many word spaces should be present in each set. The fewer the number of word spaces the greater the capacity, but the watermark is less robust.

5 Future Work

Our future plans involve improving the current algorithm and dealing with noise which can result from scanning documents.

5.1 Multi-Set Modulated Word Space

There are a number of possible improvements to the method for embedding data and the use of the space. The Threshold Buffering technique requires further work to identify a bound below the *threshold* value in the extreme cases identified in Section 3.1. The use of the space in the spare group may also provide a way of increasing the robustness of the watermark and eliminating the problems of losing sets between embedding and detection.

5.2 Noise Removal

The current noise removal procedure is done manually. The method described by Zou and Shi in [8] removed isolated black pixels. This was not implemented because the noise we saw was greater than single black pixels. Our initial experiments confirm that it may be possible to remove noise from around the outside of the text using horizontal and vertical profiling. Further research is needed to remove noise from between words or lines.

References

1. S. H. Low, N. F. Maxemchuk, and A. P. Lapone. Document identification for copyright protection using centroid detection. *IEEE Transactions on Communication*, 46(3):372–383, 1998.
2. S. H. Low and N. F. Maxemchuk. Performance comparison of two text marking methods. *IEEE Journal on Special Areas in Communications*, 16(4):561–572, 1998.
3. M. Wu, E. Tang, and B. Liu. Data hiding in digital binary images. In *International Conference on Multimedia and Expositions*, volume 1, pages 393–396, Jul 31 - Aug 2 2000.
4. A.T.S. Ho, N. B. Puhan, A. Makur, P. Marziliano, and Y. L. Guan. Imperceptible data embedding in sharply-contrasted binary images. In *ICARCV*, volume 2, pages 958 – 963, Dec 2004.
5. J. Zhao and E. Koch. Embedding robust labels into images for copyright protection. In *International Congress on Intellectual Property Rights for Specialised Information, Knowledge and New Technologies*, Vienna, Austria, 21–25 1995.
6. M. L. Miller I. J. Cox and J. A. Bloom. *Digital watermarking : principles and practice*. Morgan Kaufmann, 2001.
7. A.T.S. Ho and F. Shu. A print-and-scan resilient digital watermark for card authentication. In *ICICS-PCM*, volume 2, pages 1149 – 1152, Singapore, Dec 2003.
8. D. Zou and Y. Q. Shi. Formatted text document data hiding robust to printing, copying and scanning. In *IEEE International Symposium on Circuits and Systems (ISCAS05)*, Kobe, Japan, May 2005.