Abstract

This report introduces two semi-fragile watermarking algorithms in details, including how they are deigned and implemented using Matlab. The first algorithm relies on embedding the watermark, which is a pseudo-random sequence of 0s and 1s, into the DCT domain of a greyscale image; this algorithm supports image restoration. The second algorithm, similar to the first one, still operates in the DCT domain but it only modifies coefficients instead of replacing them. The report discusses the characteristics and performance of the algorithms and suggests possible ways of optimising them at the end.

Keywords: Semi-fragile watermarking, DCT embedding, Localisation, Restoration

Table of Contents

1. Introduction

Digital watermark is traditionally used for copy right protection or ownership identification. The watermark must be robust enough to survive as many attacks as possible in order to make sure that the watermark is still recognisable after the work is modified; in addition, the watermark cannot decrease the quality of the work significantly. An example of such watermarking system can be found in [3]

For integrity authentication, however, the sensitivity of the watermark is a major concern instead of its robustness, meaning that the watermark should reflex modifications to the work, so that people can decide whether or not the work is still trustable.

Two types of watermarks can achieve such task, which are fragile watermark and semi-fragile watermark. The former is supposed to be extremely sensitive, so that any changes to the work will destroy it; but sometimes, certain changes are acceptable or even compulsory. For example, images published on the web are likely to be compressed first, lossless compression and certain degrees of lossy compression should not be recognised as malicious modification. In this case, semi-fragile watermark is more suitable to be used as it can be designed to ignore harmless changes and reflex the malicious ones only.

The two semi-fragile watermark algorithms presented in this report are used to authenticate 8 bit greyscale images only. They accept certain degrees of JPEG compression but reject any other modifications to the images. The watermarks are embedded in the DCT domain in both algorithms, but one of them embeds the watermark bits by replacing DCT coefficients to some constants based on a threshold, while the other one modifies the coefficients to make sure that the least significant bits are consistent with the watermark bits.

One of the watermarking algorithms also features image restoration capability. Such feature will enable an image to recover itself if some parts of the image are tampered with or even chopped. Having said that, the restored parts suffer from a rather low quality because the restoration is achieved by using information from a highly compressed version of the image, many details of the image are thrown away in order to gain high compression rate.

The fundamental principles and implementations of the two watermarking algorithms are introduced in details in section 2.

Embedding watermarks in the images will inevitably change the quality of the image. The 3rd section in this report discusses the impacts and seeks improvements by adjusting some key parameters and variables in the algorithm.

2. Design and Implementation

Algorithm One

The first algorithm proposed in this report takes a grayscale image and a key as inputs, and produce a watermarked version of the image. To achieve this target in Matlab, the algorithm takes the following steps:

1. Divide the image into 8 by 8 blocks and apply DCT transform to the image in each of them.
2. Generate watermark data for the image. Unlike the watermark used in copyright protection or ownership identification, which is usually a meaningful, human-readable logo, the watermark used here is a sequence of computer generated 0s and 1s. A key is needed for the random number generator, so that whenever the same key and a length are given, the watermark generated by the generator is always the same. The size of the watermark is determined by the size of the image, more precisely, by how many 8 by 8 blocks an image has. By default, the algorithm embed 8 watermark bits into each block, so the size of the watermark is 8 * number of blocks.

```
wmk = randint(1, wmksize, [0, 1], key);
```

Code Fragment 1: Watermark Construction

3. Embed the watermark into the DCT domain. Due to the nature of DCT coefficients, not all of them can be used to embed watermark bits, for example, if the coefficients that represent low frequency are used, the quality of the image will be significantly lowered; on the other hand, embedding the watermark bits into the high frequency coefficients will make the watermark too fragile. Hence, the ideal region is the area between low and high frequency, which is indicated by the shade in figure 1:

					6	7	8
				13	14	15	16
			20	21	22	23	
		27	28	29	30		
	34	35	36	37			
41	42	43	44				
49	50	51					
57	58						

Figure 1: Suitable Coefficients for Embedding Watermark Bits

The numbers in the table above is the index of coefficients. To increase the security of the watermark, it is better to embed the watermark bits within different coefficients in each block. The shortcut to randomize the locations is to randomize the index of the coefficients, then use the first 8 indexes to locate the actual coefficients. Pseudorandom number generator is used here again to make sure that the programme can find those coefficients and extract the watermark bits for authentication.

```
blkcoefidx = randintrlv(blkcoefidx, key);
embedcoef = blkarray(blkcoefidx(1:8));
```

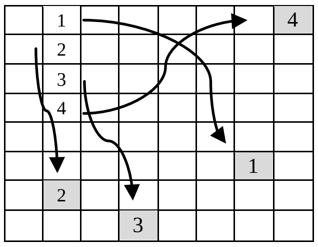
Code Fragment 2: Coefficients Randomization

The idea of embedding watermark bits into coefficients is to compare the coefficients with a threshold and change the original coefficients to some constant value if necessary. The programme set the threshold and the constant to 25 and 20 respectively, therefore, if the watermark bit is 1, and the original coefficient is greater than 25 (the threshold), then change this coefficient to 20 (the constant); similarly, if the watermark bit is 0 and the original coefficient is less than -25, then change this coefficient to -20.

```
if (wmk(wi) == 1) && (embedcoef(j) < threshold)
            embedcoef(j) = constant;
elseif (wmk(wi) == 0) && (embedcoef(j) > -threshold)
            embedcoef(j) = -constant;
end
```

Code Fragment 3: Watermark Embedding

4. The image will be converted back to spacial domain by applying inverse DCT transform in all 8 by 8 blocks.
5. Embed the highly compressed image into the least significant bits of the image itself. This step is specially required for image restoration, which recovers the image if some parts of it are modified. Embedding data into LSB is fairly easy, but it is rather tricky to compress the image. A compression scheme described in J. Fridrich and M. Goljan's paper is implemented here. The basic idea is to use 64 bits in total to represent 11 DCT coefficients in low frequency (including the DC value) and discard the rest of coefficients. This is very similar to JPEG compression, except that the quantized coefficients are further compressed, by using fewer bits to store the value. The LSB embedding process is different from the one described in the paper. In the original scheme, the highly compressed image is divided into blocks and "embedded into the LSB of the block $B + p$, where p is a vector of length approximately 1/3 of the image size with a randomly chosen direction"; while in the algorithm proposed in this paper, p is a vector of length at least 1/3 of the image size with a relatively fixed direction, depending on a key. The difference can be illustrated by the following two figures, where figure 2(a) shows the original scheme and figure 2(b) shows the modified scheme:



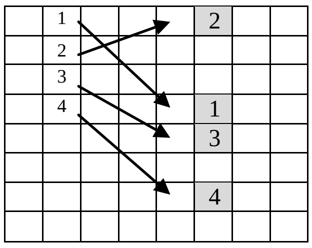


Figure 2(a) **Figure 2(b)**

Figure 2: LSB Embedding Comparison

When a watermarked image is produced, it will be saved in the same directory where the original image is located. The path and PSNR will be displayed on the user interface.

To authenticate a watermarked image, the algorithm takes a test image and a key as inputs and produces an authentication image. To achieve this target, the algorithm takes the following steps:

1. Divide the image into 8 by 8 blocks and apply DCT transform in each of them.
2. Generate watermark data in exactly the same way as in embedding process.
3. Retrieve the watermark data from each block. This step requires two procedures. First of all, locate the 8 coefficients that contain the segment of watermark data. This is done by providing the same key that is used for watermark embedding to the pseudorandom number generator. Secondly, retrieve the watermark bits by comparing the value of a specific coefficient with a constant. Similar to the constant, whose value is 20, in embedding process, the constant used here is assigned a value of 20 initially, but the coefficient does not have to match it exactly, if the coefficient is greater than 90% the constant, then record 1, otherwise record 0, because the algorithm allows certain modifications.

```
if embedcoef(j) > constant * 0.9
        localwmk(j) = 1;
elseif embedcoef(j) < -constant * 0.9
        localwmk(j) = 0;
end
```

Code Fragment 4: Watermark Retrieval

4. Compare the newly generated watermark data with the retrieved watermark. The comparison takes place immediately after all watermark bits are retrieved from one block. A Matlab function “corr2” is called to calculate how similar the two watermarks are and the result is compared with a threshold. If the result is greater than the threshold, then algorithm will believe that the block is authenticated, otherwise it will replace the entire block with a pure black block to show that it is tampered.
5. (Optional) Restore the tampered area in the image. The user can decide whether or not to restore the tampered area, if there is any. Image restoration

replies on the highly compressed image that is stored in LSB of the image, if the data in LSB is changed as well, the restoration process will result in errors.

Algorithm Two

The second algorithm proposed here is an enhanced version of the first one, it tries to improve the first algorithms in two aspects: increase reliability of the watermark and reduce modifications to the host image. This algorithm is based on the system proposed by J. Cox et al in [1] but is slightly modified to speed up execution.

The need for increasing reliability of the watermark is based on a potential design flaw in the first algorithm. In the first algorithm, watermark bits are embedded into the middle frequency of DCT coefficients only, so that the watermark can be semi-fragile and visually invisible at the same time; however, the high and low frequency are not protected by the watermark, if any modifications take place in these two areas, the watermark will not be able to detect them.

Making changes to the high frequency coefficients might be okay but making changes in low frequency coefficients will severely affects the quality of the image. To protect the coefficients in low frequency, the algorithm computes some signature data from them and combines the signature with the watermark to produce the enhanced watermark data. Assume that 8 enhanced watermark bits are going to be embedded in the ith 8 by 8 block, to compute the signature bits, the algorithm selects 8 low frequency coefficients and compare them with the coefficients in the same places in the (i + 1)th block. If the first coefficient in the current block is greater or equal to the one in the next block, record 1, otherwise record 0, then compare the second, the third and so on until 8 signature bits are computed. This procedure can be illustrated by an example (only 8 coefficients are shown here, they have been rounded into integers):

The ith Block

672	37	5					
-44	-29	1					
1	5						

The (i + 1)th Block

628	21	-33					
140	24	-8					
-11	21						

	Comparison	**Signature bit**
1	672 >= 628?	1
2	37 >= 21?	1
3	5 >= -33?	1
4	-44 >= 140?	0
5	-29 >= 24?	0
6	1 >= -8?	1
7	1 >= -11?	1
8	5 >= 21?	0

Signature Bits for the ith Block: 1, 1, 1, 0, 0, 1, 1, 0

Figure 3: Computing Signature Bits

Note that there is no “i + 1” block for the last block, so the last block will be compared with the first block to produce the final 8 signature bits. When all signature bits are computed, they will be combined with the watermark bits by using XOR operation to form the enhanced watermark data.

To embed 1 watermark bit, the algorithm firstly selects 3 coefficients, quantize them with a strength factor alpha, round them up to integers, extract LSBs and perform XOR to get 1 bit of data. The bit computed from the coefficients is compared with the watermark bit, if they are equal, no modification is needed; otherwise the LSB of the smallest coefficients will be flipped by reducing its value by one. This procedure will be repeated by 8 times if 8 watermark bits need to be embedded, hence, 24 coefficients are required in total, they are selected randomly from mid-frequency but not all of them need to be modified.

```
qba = randintrlv(floor(blkarray(blkcoefidx) ./ (alpha * qf) + 0.5), key);

for j = 1:3:24
        a = abs(qba(j));
        b = abs(qba(j+1));
        c = abs(qba(j+2));
        lsb = xor(xor(bitget(a, 1), bitget(b, 1)), bitget(c, 1));
        if lsb ~= wmk(wi)
           switch min(min(a, b), c)
               case a; qba(j) = qba(j) - 1;
               case b; qba(j+1) = qba(j+1) - 1;
               case c; qba(j+2) = qba(j+1) - 1;
           end
        end
        wi = wi + 1;
end
```

Code Fragment 5: Watermark Embedding in Algorithm Two

Two points are worth mentioning here. First of all, the quantization is done by using a standard quantization table for JPEG compression; secondly, the strength factor alpha is introduced to control how much JPEG compression the image should take. The efforts made here is to make sure that the watermark is able to survive certain degrees of JPEG compression but is still sensitive to reflex other modifications.

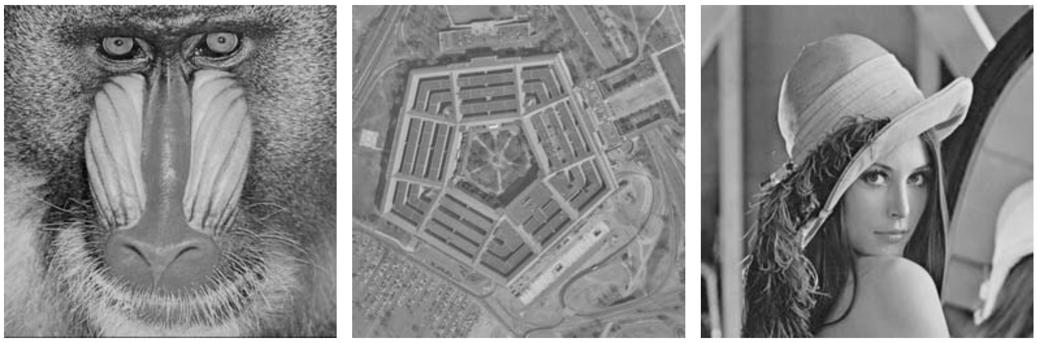
Watermark retrieval process is almost exactly the same as it is embedded, except that the algorithm only records the bit extracted from LSBs of the coefficients instead of modifying them accordingly. Also, similar to algorithm one, the extracted watermark will be compared with the original watermark and any block that fails to pass the authentication threshold will be highlighted.

GUI

The GUI is constructed using GUIDE in Matlab in order to provide good usability to the user. It is worth mentioning that the images displayed on the interface are not their actual size.

3. Analysis

Three images are chosen to be analyzed. Image baboon is an image with textures and lots of details; image Pentagon contains distinguishing lines and fewer details compare with image baboon; and image Lena has the least details but has the smoothest surfaces. All images are grayscale, with a size of 512 by 512 pixels.



Baboon Pentagon Lena

Figure 4: Sample Images for Analysis

Peak Signal-to-Noise Ratio (PSNR) is used in most of the analysis cases in this chapter. It is a term used to measure the quality of the watermarked images. A good watermarked image should have a PSNR of 30-50 dB compare with its original version.

	Baboon	Pentagon	Lena
Algorithm One	27.7230	29.9454	30.3844
Algorithm Two	34.4376	36.6370	40.0531

Figure 5: PSNR Table for Sample Images

The above table shows PSNR of the watermarked images produced by the two algorithms. Algorithm two obviously generates better watermarked images because it only reduces the value of DCT coefficients by one where necessary, while the first algorithm does not take the original value of coefficients into account. However, both algorithms modify coefficients in mid-frequency; the quality of the image is lowered inevitably.

In addition, the structure of the image could also affect its watermarked version. The image with lots of details, i.e. baboon, means that more DCT coefficients in mid-frequency or even high frequency are used to describe the image, changing these values will cause greater difference between the original and watermarked images; this may explain why PSNR for baboon is always lower than the other two. Interestingly, artifacts caused by the watermark can be well hidden into the texture; therefore, they are sometimes visually unperceivable.

Any changes made to the watermarked image should be detected by both algorithms, but two errors may occur. One of them is False Positive, meaning that some untampered areas may be recognized as tampered; the other one, False Negative, is the opposite, it ignores the tampered areas and reports they are untampered. The errors could be a result of rounding error, which occurs during DCT transition. Another reason could be the rigid threshold setting, because if the correlation between the original and retrieved watermarks is less than the threshold, then the specific part of the image will be flagged as tampered, while the threshold is set based on statistic data, it may not be able to accurately authenticate certain images that have some unique characters. Overall, the Average Detection Rate P_d is calculated as follow:

$$P_d = 100 – (P_{fp} + P_{fn}) / (1 + N)$$

Where P_{fp} is percentage of false positive, P_{fn} is percentage of false negative; N is the number of pixels in tampered areas. The table below shows the average detection rate when 10%, 20% and 30% of the watermarked image, produced by algorithm one, are modified via copy and paste attack.

	Baboon	Pentagon	Lena
No Modification	***99.9637*** FP: 0.0732 FN: 0.0000	***99.9878*** FP: 0.0244 FN: 0.0000	***100.0000*** FP: 0.0000 FN: 0.0000
10% Modification	***95.8773*** FP: 0.0821 FN: 8.1633	***95.3378*** FP:0.0274 FN: 9.2971	***96.5986*** FP: 0.0000 FN: 6.8027
20% Modification	***96.8029*** FP: 0.0922 FN: 6.3020	***96.5958*** FP: 0.0307 FN: 6.7776	***96.3139*** FP: 0.0000 FN: 7.3722
30% Modification	***96.8757*** FP: 0.1045 FN: 6.2041	***96.9622*** FP: 0.0348 FN: 6.0408	***97.2245*** FP: 0.0000 FN: 5.5510

Figure 6: Average Detection Rate for Algorithm One
(FP: False Positive, FN: False Negative)

By using the default threshold, false negative rate is always greater than false positive rate. The average detection rate is maintained in a reasonable range. When JPEG compression is applied, false positive is increased significantly while false negative rate slightly drops. The following table is constructed using images with 20% modification:

	Baboon	Pentagon	Lena
QF = 100	***96.7588*** FP: 0.0615 FN: 6.4209	***96.2986*** FP: 0.0307 FN: 7.3722	***95.9393*** FP: 0.0000 FN: 8.3234
QF = 90	***89.6535*** FP: 15.6989 FN: 4.9941	***87.5963*** FP: 17.9109 FN: 6.8966	***87.7305*** FP: 17.8802 FN: 6.6587
QF = 70	***82.9681*** FP: 28.3564 FN: 5.7075	***80.6426*** FP: 33.3641 FN: 5.3508	***79.1230*** FP: 34.5008 FN: 7.2533
QF = 50	***60.1391*** FP: 75.0845 FN: 4.6373	***59.9388*** FP: 76.4363 FN: 3.6861	***58.9330*** FP: 77.9724 FN: 4.1617

Figure 7: Average Detection Rate for Algorithm One After JPEG Compression
(FP: False Positive, FN: False Negative)

Apparently, JPEG compression results in modifications in DCT domain, many DCT coefficients, including the ones that are embedded with watermark bits, are changed. Coefficients that are close to high frequency are more likely to be compressed or to be discard during compression; when quality factor decreases, more and more coefficients will fail to keep their original values. Watermark bits extracted from those coefficients are extremely likely to be different from the original ones, so that many blocks are flagged as being tampered.

It is possible to improve the performance of the algorithm by adjusting some of the key parameters. Some modifications are introduced to algorithm one in order to lower its impact on the host image and increase its average detection rate.

Experiment 1: reduce the number of watermark bits embedded in each 8 by 8 block from 8 to 4.

	Baboon	Pentagon	Lena
PSNR	30.4589	32.5119	33.5699
ADR for No Modification	***99.6216*** FP: 0.7568 FN: 0.0000	***99.8047*** FP: 0.3906 FN: 0.0000	***99.8413*** FP: 0.3174 FN: 0.0000
ADR for 20% Modification	***84.2017*** FP: 1.7512 FN: 29.8454	***90.3934*** FP: 0.3072 FN: 18.9061	***88.2530*** FP: 0.3072 FN: 23.1867
ADR for 20% Modification and JPEG Compression (QF = 70)	***69.0808*** FP: 35.0845 FN: 26.7539	***69.4205*** FP: 37.9724 FN: 23.1867	***78.0425*** FP: 19.5392 FN: 24.3757

Figure 8: Results for Experiment 1
(ADR: Average Detection Rate, FP: False Positive, FN: False Negative)

This experiment is attempting to introduce fewer artifacts to the host image by using less watermark bits per block. The improved PSNR shows that such modification surely helps. The consequence, on the other hand, is that the average detection rate drops, because 4 watermark bits do not seem to be sufficient to cover an 8 by 8 block. In addition, one bit of disagreement among 4 bits means a 75% similarity between the original and the retrieved watermark; if 8 bits are used, the percentage could be increased to 87.5.

Experiment 2: reduce the number of coefficients used for embedding watermark bits. In this case, 6 coefficients that are close to the low frequency are discarded. Figure 9 illustrates this modification.

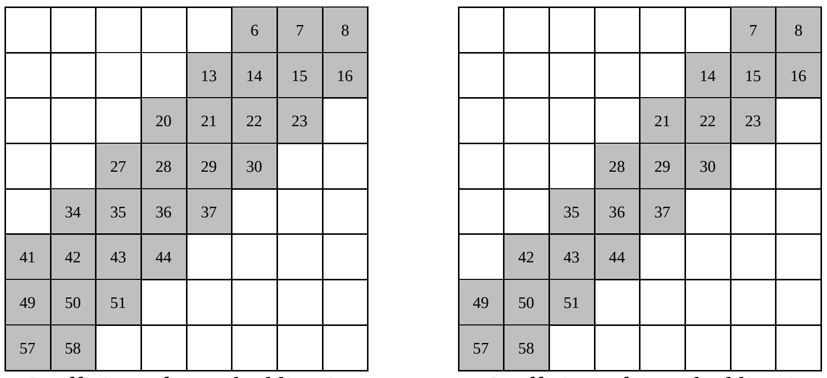


Figure 9: Coefficient Usage Comparison

	Baboon	Pentagon	Lena
PSNR	28.2750	29.8454	30.6408
ADR for No Modification	***99.9146*** FP: 0.1709 FN: 0.0000	***100.0000*** FP: 0.0000 FN: 0.0000	***99.9878*** FP: 0.0244 FN: 0.0000
ADR for 20% Modification	***97.0168*** FP: 1.5668 FN: 4.3995	***98.5137*** FP: 0.0000 FN: 2.9729	***96.8043*** FP: 0.9217 FN: 5.4697
ADR for 20% Modification and JPEG Compression (QF = 70)	***85.8340*** FP: 23.9324 FN: 4.3995	***83.4041*** FP: 27.1275 FN: 6.0642	***83.5384*** FP: 27.0968 FN: 5.8264

Figure 10: Results for Experiment 2
(ADR: Average Detection Rate, FP: False Positive, FN: False Negative)

Similar to the purpose of experiment 1, the modification is made to reduce noise in the watermarked image, by embedding watermark bits in coefficients closer to high frequency. PSNR shows that the quality of the watermarked images is slightly increased. There does not seem to be any side effects, On the contrary, the average detection rate is even a little bit higher. The quality of the watermarked images might be the explanation, the better the quality is, the fewer interferences the watermark will suffer from; nevertheless, the algorithm still needs enough watermark bits to authenticate each block, 8 bits seem to be alright but 4 is a bit too few.

Experiment 3: reduce the constant from 20 to 10 and the threshold from 25 to 15 in the embedding procedure.

	Baboon	Pentagon	Lena
PSNR	29.9912	33.7064	35.1470
ADR for No Modification	***98.4497*** FP: 3.1006 FN: 0.0000	***98.6084*** FP: 2.7832 FN: 0.0000	***98.4253*** FP: 3.1494 FN: 0.0000
ADR for 20% Modification	***94.0519*** FP: 4.7619 FN: 7.1344	***93.8067*** FP: 4.3011 FN: 8.0856	***95.4233*** FP: 4.5161 FN: 4.6373
ADR for 20% Modification and JPEG Compression (QF = 70)	***57.9172*** FP: 80.1229 FN: 4.0428	***56.7665*** FP: 83.2565 FN: 3.2105	***55.4514*** FP: 85.5300 FN: 3.5672

Figure 11: Results for Experiment 3
(ADR: Average Detection Rate, FP: False Positive, FN: False Negative)

Setting the constant and the threshold to smaller numbers improves the quality of the watermarked image, more significantly than previous experiments. As a result, the watermarked image becomes more sensitive to changes. In this experiment, no image can achieve a perfect detection rate when no modification is applied; and the average detection rate reaches record low with JPEG compression.

Experiment 4: change the watermark extraction algorithm presented in code fragment 4 to:

```
if embedcoef(j) > 0
            localwmk(j) = 1;
else
            localwmk(j) = 0;
end
```

Code Fragment 6: Altered Watermark extraction algorithm

	Baboon	Pentagon	Lena
ADR for No Modification	***100.0000*** FP: 0.0000 FN: 0.0000	***100.0000*** FP: 0.0000 FN: 0.0000	***100.0000*** FP: 0.0000 FN: 0.0000
ADR for 20% Modification	***96.6012*** FP: 0.6144 FN: 6.1831	***97.4435*** FP: 0.0000 FN: 5.1130	***96.7127*** FP: 0.1536 FN: 6.4209
ADR for 20% Modification and JPEG Compression (QF = 70)	***92.6480*** FP: 7.9263 FN: 6.7776	***92.9527*** FP: 8.3871 FN: 5.7075	***90.9922*** FP: 9.2166 FN: 8.7990

Figure 11: Results for Experiment 4
(ADR: Average Detection Rate, FP: False Positive, FN: False Negative)

By changing the watermark extraction algorithm, watermarked images finally get rid of false positive errors when no modification is applied; the change made to the algorithm offers greater flexibility, so that reasonable JPEG compression does not seem to be a problem here, either. This is not the case when quality facto is lower than 60, though, where over 80% of the image fails to pass the authentication process. Such modification could be a desired improvement to the algorithm, if the user does not care about JPEG compression, otherwise the user should stick to the default setting because the compression can be clearly reflected.

Experiment 5: change the authentication threshold τ from 0.65 to 0.9 or 0.5.

	Baboon	Pentagon	Lena
τ = 0.5, ADR for 20% Modification	***94.6239*** FP: 0.6452 FN: 10.1070	***98.1544*** FP: 1.0753 FN: 2.6159	***98.0509*** FP: 1.0445 FN: 2.8537
τ = 0.5, ADR for 20% Modification and JPEG Compression (QF = 70)	***87.3775*** FP: 14.9002 FN: 10.3448	***83.3145*** FP: 19.8157 FN: 13.5553	***84.1850*** FP: 20.2151 FN: 11.4150
τ = 0.9, ADR for 20% Modification	***97.1713*** FP: 3.0415 Fn: 2.6159	***97.7584*** FP: 2.5806 FN: 1.9025	***97.4754*** FP: 2.6728 FN: 2.3781
τ = 0.9, ADR for 20% Modification and JPEG Compression (QF = 70)	***63.4346*** FP: 70.7527 FN: 2.7381	***62.6080*** FP: 74.0707 FN: 0.7134	***62.3909*** FP: 74.6237 FN: 0.5945

Figure 12: Results for Experiment 5
(τ: Threshold, ADR: Average Detection Rate, FP: False Positive, FN: False Negative)

If the retrieved watermark exactly matches the original one, function “corr2” will return 1. Other common values produced by this function are close to 0.7, 0.5 and 0.1, negative values are fairly common if the image is largely tampered. This experiment shows that setting the threshold to different values changes the sensitivity of the watermark, the higher the threshold is, the more sensitive (fragile) the watermark gets.

Now let’s examine algorithm two. Compare with algorithm one, the second algorithm is not a completely new algorithm. As stated previously, it is an enhanced version of algorithm one, and the major improvements include being able to produce high quality watermarked image (this is proved by data in Figure 5) and offering protection for DCT coefficients in high/low frequency. Its performance can be summarized by the following table:

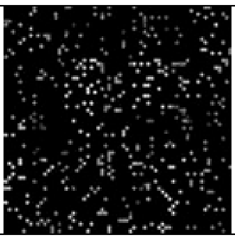
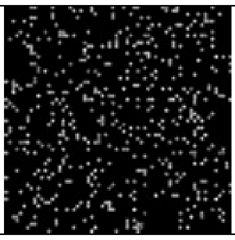
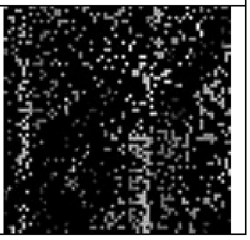
	Baboon	Pentagon	Lena
No Modification	***97.6563*** FP: 4.6875 FN: 0.0000	***98.6206*** FP: 2.7588 FN: 0.0000	***99.1455*** FP: 1.7090 FN: 0.0000
10% Modification	***84.9195*** FP: 20.4104 FN: 9.7506	***85.8866*** FP: 16.6621 FN: 11.5646	***86.5706*** FP: 15.2941 FN: 11.5646
20% Modification	***79.9795*** FP: 29.3395 FN: 10.7015	***80.1024*** FP: 29.0937 FN: 10.7015	***80.8849*** FP: 25.7450 FN: 12.4851
30% Modification	***74.3433*** FP: 39.1501 FN: 12.1633	***76.3226*** FP: 37.7220 FN: 9.6327	***78.7543*** FP: 32.5322 FN: 9.9592
20% Modification with JPEG Compression (QF = 90)			

Figure 13: Average Detection Rate for Algorithm Two
(FP: False Positive, FN: False Negative)

Algorithm two seems to have a relatively low average detection rate. This is the result of simplifying the algorithm described by J. Cox et al; the error correction part from the original literature, which guarantees the success of bit flipping, is not implemented here, when DCT coefficients are transformed back to spacial domain, rounding errors may kill off the watermark bits.

An obvious solution to this problem is to add error correction mechanism to the programme, but doing so will significantly increase the running time because such mechanism involves executing a dozen of loops. Matlab is not optimised for dealing with loops, so it may be better to implement it in other languages such as C or Java.

When JPEG compression is applied, even with a high quality factor, more than 85% of the image is marked as tampered. This is caused by the diversity of quantization tables. The algorithm uses the standard quantization table, so if the JPEG compression is done using the same table, everything will be fine. The fact is, however, Matlab uses different quantization tables, if the JPEG image is written by the "imwrite" function, it will be impossible to authenticate it.

The last topic in this chapter is image restoration. In an image, a tampered area can be recovered by extracting its highly compressed version from the image. In order to save space, the image itself is compressed in an extreme way, for example, the highly compressed version of the three sample images are shown below with the PSNR:

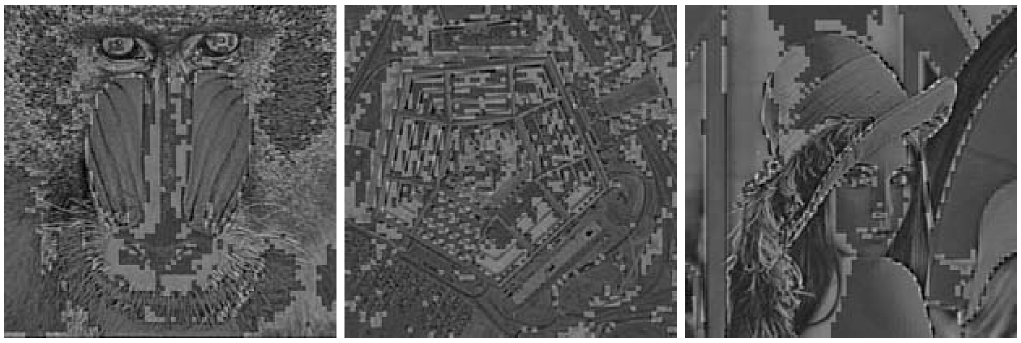

Baboon PSNR 12.1280 | Pentagon PSNR: 11.5259 | Lena PSNR: 12.7947

Figure 14: Compressed Image for Restoration

The following figure shows the recovery capacity in three scenarios. In group (a), a part of the image is copied and pasted onto some other part; in group (b), a part of the image is chopped off and in group (c), a fairly large part of the image, roughly a third of the image is chopped off, which is close to the maximum area that can be recovered.

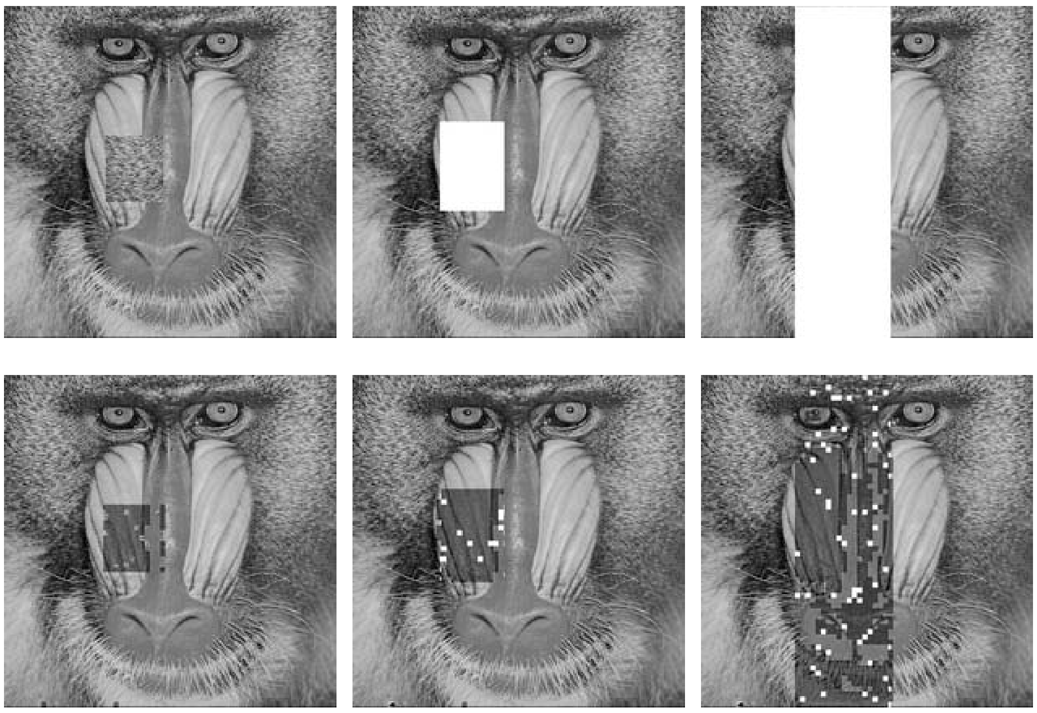

Group (a) | Group (b) | Group (c)

Figure 15: Image Restoration Capacity

Images presented in three groups all suffer from false negative errors, but most of the area is successfully recovered, even when a third of the image is chopped off. The image recovery scheme introduce in [2] would not be able to achieve this because it randomly distributes the recovery data; while the scheme is modified in here to guarantee that up to 30% of the image can be recovered.

4. Conclusion

This report examines two approaches of implementing semi-fragile watermarks. Algorithm one embeds watermark bits by replacing the original DCT coefficients with a constant; while algorithm two modifies coefficients according it their original values. At this point, the second algorithm is more intelligent than the first one, it is able to produce watermarked images with relatively high quality; however, watermarked images produced by algorithm two are quite sensitive compare with algorithm one, in other words, the watermark embedded by algorithm two seems to be fragile, rather than semi-fragile; as discussed in chapter 3, it is partially a result of using different quantization tables, another reason might be the result of over-simplifying. Generally speaking, algorithm two is more advanced; it has a better framework but its proper implementation is rather heavy-weight; algorithm one is straightforward, to gain a better performance, it needs to be carefully tuned, several ways of adjusting algorithm one has been discussed in chapter 3.

An image restoration technique is implemented and briefly reviewed. As the test images show, the mechanism works but it could do better, as the quality of the recovered image is disappointingly low. Many image restoration schemes that are available in literatures seems to be some kind of deblurring or noise removing schemes, they has nothing to do with recover tampered images, such as [5]. The key issue here is not how to embed or retrieve the compressed image; the most important question is how the image can be efficiently compressed. Some compression schemes can be found in [4], lossless compression does not seem to be an option here; and it is hard to find the balance between the quality of the image and the redundancy the can be removed in lossy compression. A very interesting restoration scheme is [6], it is able to recover the image without embedding any data in advance. If there is more time, this scheme should definitely be studied.

5. Contribution

I implemented the two algorithms and the graphic user interface.

6. Reference

[1] J. Cox, M.L. Miller, J.A. Bloom, *Digital Watermarking*, Morgan Kaufinann Publisher, 2002

[2] Jiri Fridrich, Miroslav Goljan, *Images with Self-Correcting Capabilities*, 1999

[3] S Samuel and WT Penzhom, *Digital Watermark for Copyright Protection*, 2004

[4] Cebrail Taskin, Serdar Kursat Sarikoz, *An Overview of Image Compression Approaches*, The Third International Conference on Digital Telecommunications, 2008

[5] Miki Haseyama, Megumi Takezawa, Keiko Kondo and Hide0 Kitajima, *An Image Restoration Method Using IFS*, 2000

[6] Wen Li, David Zhang, Zhiyong Liu, and Xiangzhen Qiao, *Fast Block-Based Image Restoration Employing the Improved Best Neighborhood Matching Approach*, 2005