Program Homework 1

Implementation of the Spread-Spectrum Watermarking System

A. Informed Watermark System

a. Implementation:

I first transform the picture from RGB domain to the YCBCR domain. After that, I do DCT on the luminance layer of that transformed picture. Then, I put the watermark with length N into the highest frequency blocks of that DCT matrix instead of the largest magnitude coefficients.

To fulfill that, I use the concept of ZIGZAG algorithm to get the highest frequency blocks. At the end, I retransform the picture from YCBCR domain to the RGB domain, and export that picture.

If I want to extract the watermark, I first subtract the suspect picture with the origin one in DCT domain. Since the watermark is generated by Gaussian random generator, I will normalize the results with the largest value. Then, I can get the wanted value of similarity by do DOT calculation.

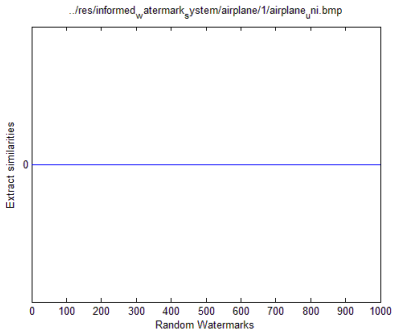
b. Performance:

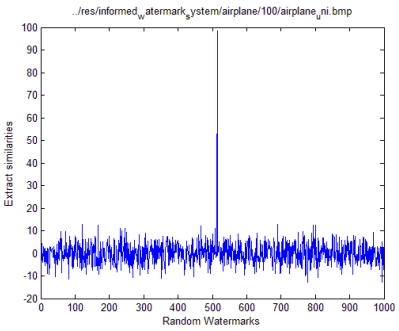
Since I put the watermark on the highest frequency blocks of that picture, the appearance of picture remains well even the alpha is near 100. However, the performance depends highly on the value of alpha.

When the alpha value is too small, such as 0.1 and 1, sign() algorithm make it very difficult to differentiate the watermarks. As you can easily find out, the performance means nearly nothing in the graph.

If you increase alpha to about 10 or 100, the watermark performance will sharply increase.

Thus, I will neglect alpha with 0.1, 1 and only focus on alpha with 10, 100 afterwards in this report.





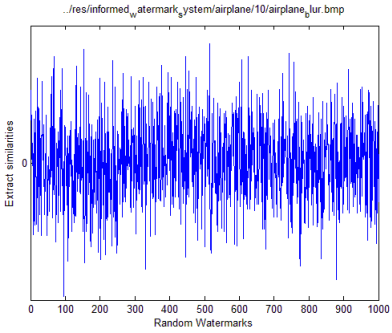
c. Attacks – robustness:

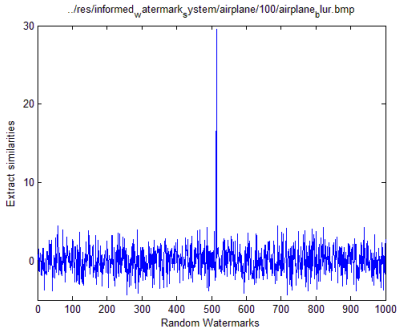
I figure out a few attack methods, such as blurring, collusion, cropping, dithering, JPEG compression, multiple-watermarking, rotating and scaling. I will demonstrate the implementation and performance against these attacks as following, and for simplicity, I only introduce the picture “airplane.bmp”.

1. Blurring:

   I use fspecial(), and imfilter() in MATLAB to fulfill this attack.

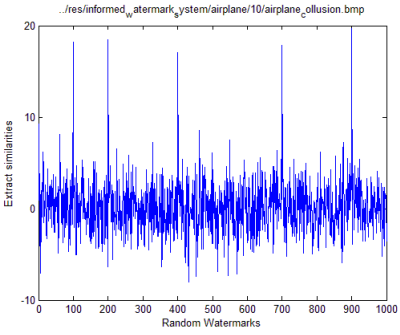
   The performance of my embed-algorithm with alpha 100 can remain similarity at about 0.3, however, it decreases sharply when alpha is low.

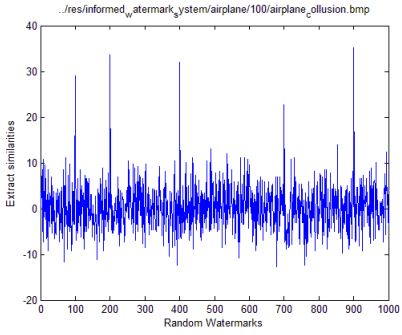




2. Collusion:

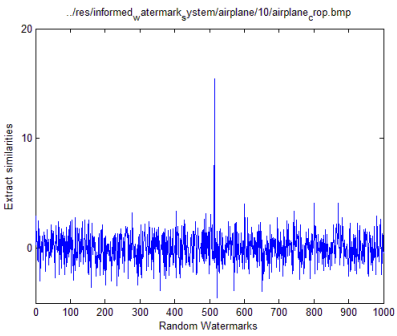
   I embed the average of five watermarks into the picture. The performance remains well even alpha decrease.

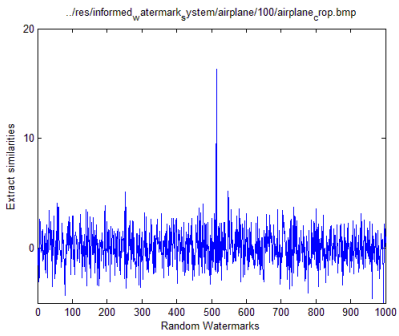




3. Cropping:

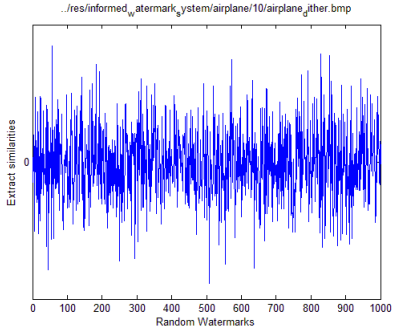
   I crop the center of watermarked image with 1/4 size as original image. I get poor performance if I calculate similarity with the corresponding part of original image. If I recover the cropped image by cover it on the original image then do calculation, its performance increases a lot.

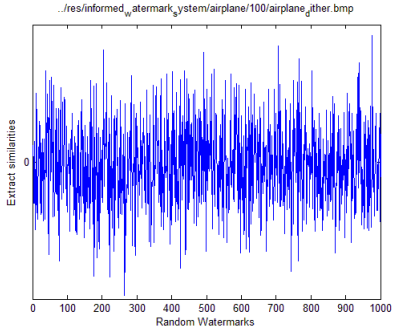




4. Dither:

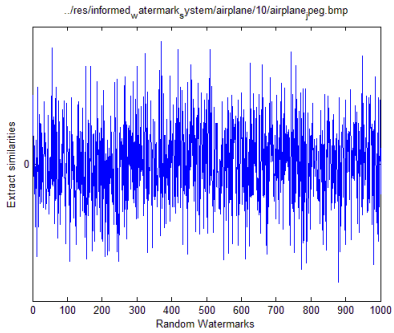
   I initially get poor watermarked image in these attacks. Finally, I find out that it is caused by the biasing of the return value from dither() in MATLAB. Nevertheless, the performance remains poor after fixing bugs.

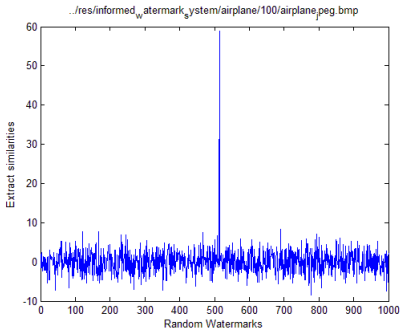




5. JPEG – compression:

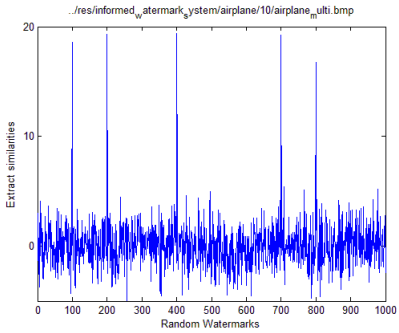
   My algorithm do well against this attack only when alpha is big enough.

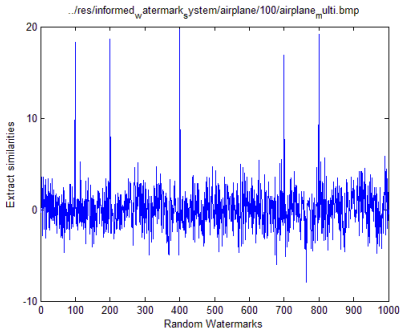




6. Multi-watermark:

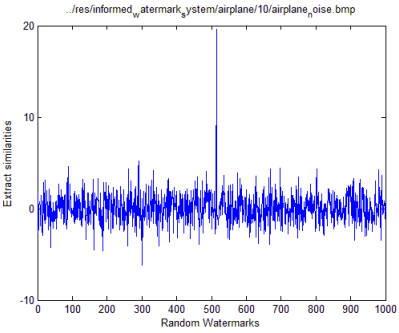
   My algorithm can deal with this attack well.

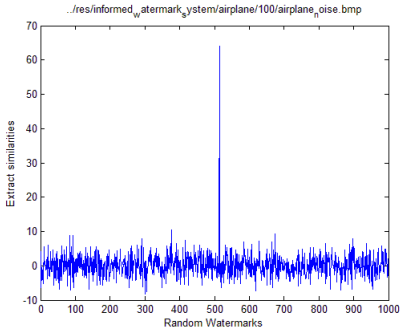




7. Noise:

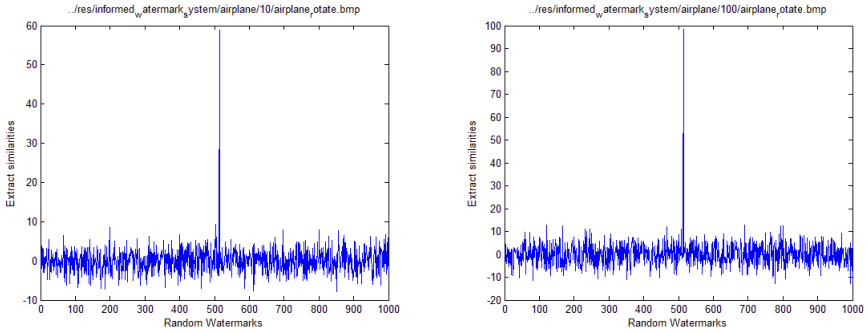
   My algorithm can deal with this attack well.





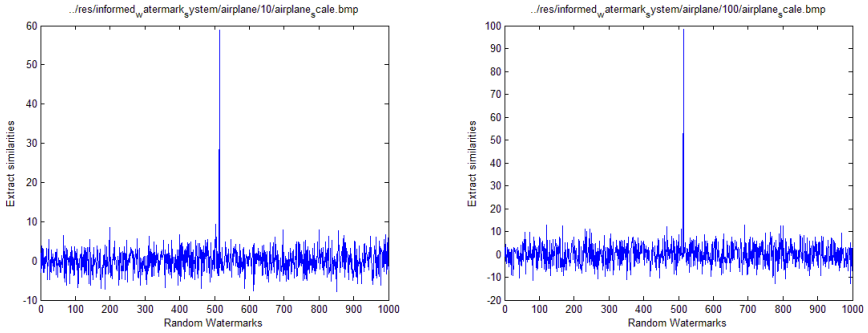
8. Rotation:

   My algorithm can deal with this attack well.



9. Scaling:

   My algorithm can deal with this attack well.



d. Challenges:

The above performance clearly indicates that hiding watermarks on the N-highest frequency is not a good idea against various attacks. It performs poorly especially against blurring, dithering, and JPEG compression.
A better way is to hide the information in the higher magnitude coefficients of the DCT matrix.

B. Multiple Watermark System

a. Implementation:

This can be done easily by just embed multiple watermarks. One can claim the ownership of one multimedia if and only if he owns a higher order of watermark sequence, for example, a tuple of correct watermarks.

b. Performance:

Using multiple watermarks can enhance the capacity of the hiding message, and thus increase the overall robustness. This can be proved by easy calculation in math.

c. Attacks – robustness:

Just similar as the above experiment.

d. Challenges:

Also similar to the above-said.

C. Blind Watermark System

a. Implementation:

Similar to the informed watermark system. I only do a little change at the extraction part. When using the informed watermark system, I will let every element in the extracted watermark sequence divide the largest value among them. This procedure can eliminate the influence created by alpha.

For blind system, it is much more difficult to do so. The solution that comes to my mind is as following. First, we may neglect the influence created by $v \cdot x$, since they are in the various domain. Second, we calculate the similarity $x' \cdot x \times \alpha$ for all watermarks, and let them divide the largest value among them. The reason for this procedure is that if x' and x are similar, the calculation result of this is approximately $N \times \alpha$, where N is the length of watermark. So, after the division, only the real watermark can remains about 1, and others remain nearly 0.

This method will fail if all the watermarks are true, or if all the watermarks are false. In practice, we may assume this situation will not happen easily.

b. Performance:

The performance is about the same as informed one.

c. Attacks – robustness:

The robustness is about the same as informed one.

d. Challenges:

The challenges is about the same as informed one.

D. Conclusion:

Frankly speaking, I do not follow exactly the specification stated on the ppt. I misunderstand the highest magnitude coefficients as highest frequency and thus make the watermark robustness poor. I will improve the performance by using another algorithm next time.

MATLAB is really a powerful tool for matrix calculation and image processing; its algorithm implementations for FFT and DCT are far faster.

When I need to do attacks with wide image domain knowledge, I will google, check the functionality and usage of specific functions, then just use it. I make a lot of functions for various attacks and use autorun() to do all the jobs for me.

E. PSNR:

a. airplain.bmp:

0.1 (51.3721); 1 (51.3721); 10(48.7273); 100(33.0658);

b. baboon:

0.1 (52.1231); 1 (52.1231); 10(49.4868); 100(33.8213);

c. fruits:

0.1 (52.1979); 1 (52.1979); 10(49.5559); 100(33.9418);

d. peppers:

0.1 (51.5443); 1 (51.5443); 10(48.9231); 100(33.3056);

F. Data Storage:

Origin images are put under “data/” folder.
Documentations are put under “doc/” folder.
Experiment results are put under “res/” folder, which includes raw data and images for all the pictures with alpha 0.1, 1, 10, 100 and one marked image in both informed and blind watermark system.
Detailed code implementations can be traced in “src/” folder.
Not-important temporary files will be stored in “tmp/” folder.
Random generated watermarks are put in “wm/” folder.

G. Execution instruction:

a. Change the working folder to be “src/”.
b. Modify “src/testAll.m” to specify to-do attacks if needed.
c. Modify each attack in “src/testXXX.m” if needed.
(XXX means the name of attack)
d. Insert “autorun()” for do the overall calculation.