Implementation of: "slant transform watermarking for digital images" in matlab

Sammy Pfeiffer
09/06/13

Index

1. Introduction

This project is about the implementation of the paper “*Slant Transform Watermarking for Digital Images*” by Anthony T.S. Ho, Xunzhan Zhu and Jun Shen. Which can be found at: http://www.surrey.ac.uk/computing/files/pdf/papers/Anthony_Ho/5_Ho_2003.pdf

As we can read in it's abstract it's about:

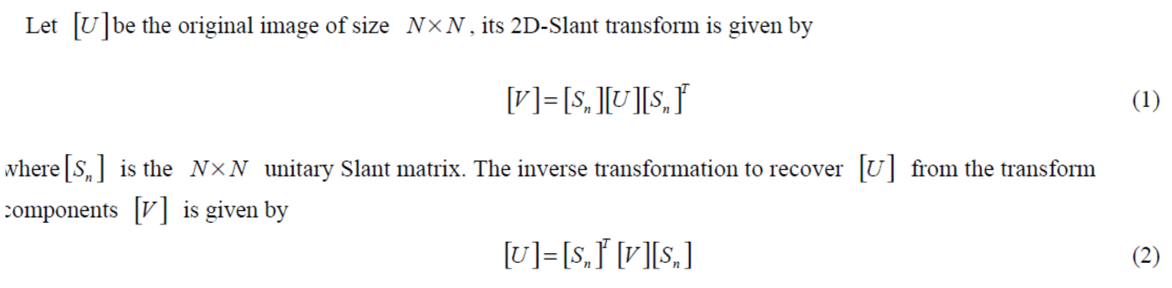
“In this paper, we propose a digital watermarking algorithm based on the Slant transform for the copyright protection of images.”

The implementation has been done in MATLAB R2012b and can be found at https://github.com/awesomebytes/slant_paper_implementation

2. Implementation

First of all we need to create Slant matrices. This is a rather painful process very error prone, after trying to implement it by myself for a couple of days I found out I always had some error comparing my obtained matrices with some matrices I could find already computed. Luckily after some depth searching I found out the work done by I. W. Selesnick publicly available at http://eeweb.poly.edu/iselesni/slantlet/index.html which contains functions to compute the Slant matrices for the Slant transform in MATLAB.

This makes the computing of the paper 2D-Slant transform pretty easy:


Let $[U]$ be the original image of size $N \times N$, its 2D-Slant transform is given by

$$[V] = [S_n][U][S_n]^T \quad (1)$$

where $[S_n]$ is the $N \times N$ unitary Slant matrix. The inverse transformation to recover $[U]$ from the transform components $[V]$ is given by

$$[U] = [S_n]^T [V][S_n] \quad (2)$$

1: Paper Slant transform

In the code we can find that computing it looks like:

```
N_watermark=size(W);
L_watermark = log2(N_watermark(1)); % getting the size of the image to compute
the slant matrix
S_n_watermark = sltmtx(L_watermark); % compute slant matrix
S_nT_watermark = S_n_watermark'; % compute inverse slant matrix
U_watermark = W_double;_% just to use same terminology than the paper
V_watermark = S_n_watermark * U_watermark * S_nT_watermark; % (1)
originalU_watermark = S_nT_watermark * V_watermark * S_n_watermark; % (2)
```

Now, having the ability of computing the Slant transforms, we can start with the implementation of the method. The first step we find is:

“The watermark image, W (x, y), is first transformed into a set of Slant transform coefficients by equation (1). A Slant transform matrix of order 64 is applied to this image, and then a 64×64 Slant transform coefficients matrix is obtained. The DC component is stored in the secret key file and the AC components are used for watermark embedding.”

We only need to add to the previous code the storing of the secret keyfile:

```
% DC component is the secret key file
secret_key = V_watermark(1,1);
% AC components are used for watermark embedding
```

If we move on we find:

“Let the original image be I (x, y) . Similar to the algorithm used in [11,12], it is decomposed into a set of non-overlapped 8x8 sub-blocks.”

Which in code looks like:

```
% the original image I
% decompose in non-overlapped 8x8 sub-blocks
I_sub_8x8 = mat2cell(I_double,  [zeros(1,64) + 8], [ zeros(1,64) + 8]); % 64x64
of 8x8 blocks
```

Let's continue:

“An m-sequence random number generator is used to select a certain number of sub-blocks for watermark embedding, whose initial seed is also kept in the secret key file. In every selected sub-block, sixteen middle and high frequency coefficients are used for later modulation.”

My interpretation of the m-sequence random number generator is just to generate the sub-blocks that we will use for watermarking later on. The seed is stored.

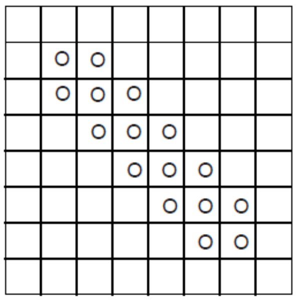
```
% we need a seed for random which will be secret too
rng(secret_rng_seed); % supersecret seed
% get some random numbers here, quantity of them "m" in the paper
m_subblocks = (64*64) / 16; % and we insert the coefficients by 16 in each
subblock
m_subblocks = m_subblocks - 1; % substract last subblock that will have only 15
coefficients
% thats 256 by the way

% m-sequence random number generator for selecting a certain number of
% sub-blocks for watermark embedding
subblocks_for_water_embedding_row = randi([1 64], 1, m_subblocks);
subblocks_for_water_embedding_column = randi([1 64], 1, m_subblocks);

% in every subblock we select 16 values
% they follow scheme in paper (diagonal)
embedding_locations = [2 2; 3 2; 3 2; 3 3; 3 4; 4 3; 4 5; 4 6; 5 4; 5 5; 5 6; 6
5; 6 6; 6 7; 7 6; 7 7];
```

The embedding locations are chosen like is described in the paper.

“The high frequency components are relatively vulnerable to compression operations, while the low frequency components must be retained for visual quality of the watermarked image. Therefore, most existing watermarking schemes choose to embed the watermark into the middle frequency band. In our scheme, embedding locations as shown in Figure 3 are adopted, which are observed, through our experiments, to provide a best tradeoff between robustness and data integrity.”


2: Embedding Locations

Let's get into the watermarking:

*“Let the watermark Slant transform coefficients denoted by m_i. The AC coefficients of Slant transformed original image sub-blocks, before and after inserting watermark are denoted by x_i and *x_i , respectively, and i∈(0,n], where n is the number of the watermarked coefficients to be inserted into every sub-block, which is set to 16 in our experiment.*

The embedding formula is given as follows

***x_i = α m_i** (7)*

where α is the watermark strength factor that controls the tradeoff between visual quality of the watermarked image and robustness of the watermarking scheme.

*After embedding, the original coefficient x_i is replaced by *x_i and a new 8×8 matrix of Slant transform coefficients of image sub-block is obtained. The inverse Slant transform is then applied to the 8×8 matrix using equation (2) to obtain the luminance matrix of the watermarked image sub-block. After performing the watermark insertion for all the selected sub-blocks of the original image, a watermarked image, I ' (x, y) , is obtained.“*

This is how it looks in code:

```
m_i=2; % m_i = 1 will be the DC coefficient so dont use it
S_n = sltmtx(log2(8)); % compute slant matrix of size 3, the size of the block
S_nT = S_n'; % compute inverse slant matrix
alpha = 0.1; % alpha of the paper is not described... 0.6 is a lot it seems, 0.1
is not so noticeable (but it is very much)
% watermark those locations for every subblock chosen to be watermarked
for current_subblock_to_watermark=1:m_subblocks
    U_subblock =
cell2mat(I_sub_8x8(subblocks_for_water_embedding_row(current_subblock_to_waterma
rk), subblocks_for_water_embedding_column(current_subblock_to_watermark)));
    V_subblock = S_n * U_subblock * S_nT; % (1) Slant transformed
    for i=1:16  % for every location
        location = embedding_locations(i,:);
        coef_to_insert = V_watermark((uint8(m_i)/64) +1, mod(m_i,64) +1); % this
gets the next coefficient from the 64x64 watermark
        m_i = m_i+1;
        V_subblock(location(1),location(2)) = alpha * coef_to_insert;
    end
    watermarkedU_subblock = S_nT * V_subblock * S_n; % (2) recovered with the
modified coefficients

I_sub_8x8_watermarked{subblocks_for_water_embedding_row(current_subblock_to_wate
rmark), subblocks_for_water_embedding_column(current_subblock_to_watermark)} =
watermarkedU_subblock;
end

I_watermarked_double = cell2mat(I_sub_8x8_watermarked);
```

Once having the image watermarked the paper continues:

*“In watermark detection, the positions of the sub-blocks with watermark embedded are computed using the seed of the m-sequence and initial state number that is stored in the key file. All the selected sub-blocks are Slant transformed. Let these coefficients denoted by *'x_i and the retrieved watermark Slant transform coefficients by 'm_i' , and i∈ (0,n], where n is the number of the watermarked coefficients to be inserted in every sub-block. The watermark extraction formula is given by*
*m_i' = *'x_i / α ”*

In code this means to do something similar to the inverse of what we did before:

```
detected_watermark_slant = zeros(64,64);
detected_watermark_slant(1,1) = secret_key;
m_i=2; % m_i = 1 is the DC coefficient
S_n = sltmtx(log2(8)); % compute slant matrix of size 3, the size of the block
S_nT = S_n'; % compute inverse slant matrix
% extract watermark coefficients from the blocks
for current_subblock_to_detect=1:m_subblocks
    U_subblock =
cell2mat(watermarked_image_sub_8x8(subblocks_for_water_embedding_row(current_sub
block_to_detect),
subblocks_for_water_embedding_column(current_subblock_to_detect)));
    V_subblock = S_n * U_subblock * S_nT; % (1) Slant transformed
    for i=1:16  % for every location
        location = embedding_locations(i,:);
        detected_watermark_slant((uint8(m_i)/64) +1, mod(m_i,64) +1) =
V_subblock(location(1),location(2)) / alpha; % store each coef
        m_i = m_i+1;
    end
end

S_n = sltmtx(log2(64)); % compute slant matrix of size 6, the size of the
watermark image
S_nT = S_n'; % compute inverse slant matrix
detected_watermark_double = S_nT * detected_watermark_slant * S_n; % We recover
the watermark
detected_watermark = uint8(detected_watermark_double);
```

3. Results

Results using lena image:

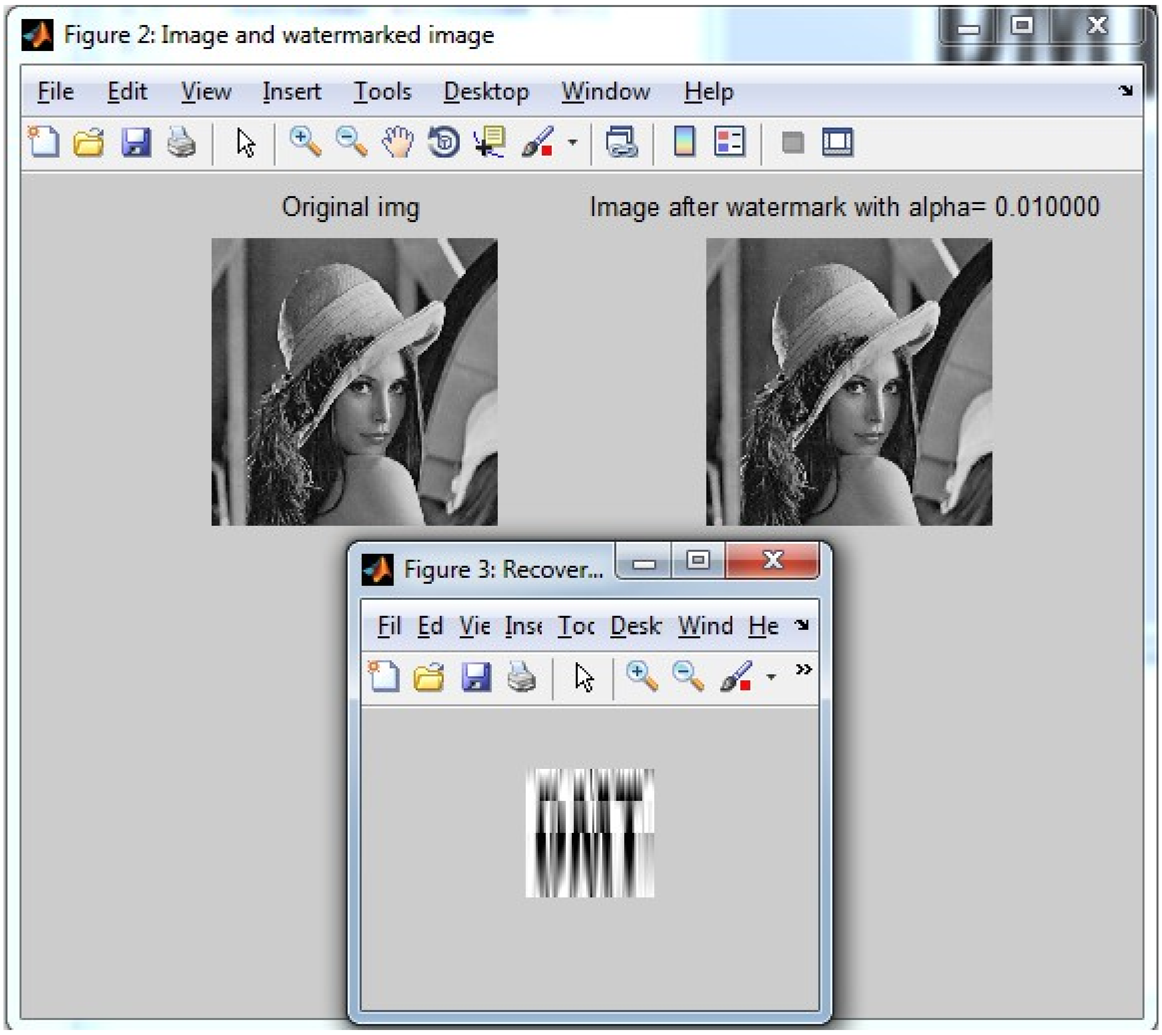
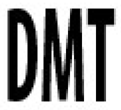


Figure 2: Image and watermarked image
File Edit View Insert Tools Desktop Window Help
Original img
Image after watermark with alpha= 0.010000
Figure 3: Recover...
Fil Ed Vie Inse Toc Desk Wind He

3: results with lena

Using the watermark looking like:



Results using baboon:

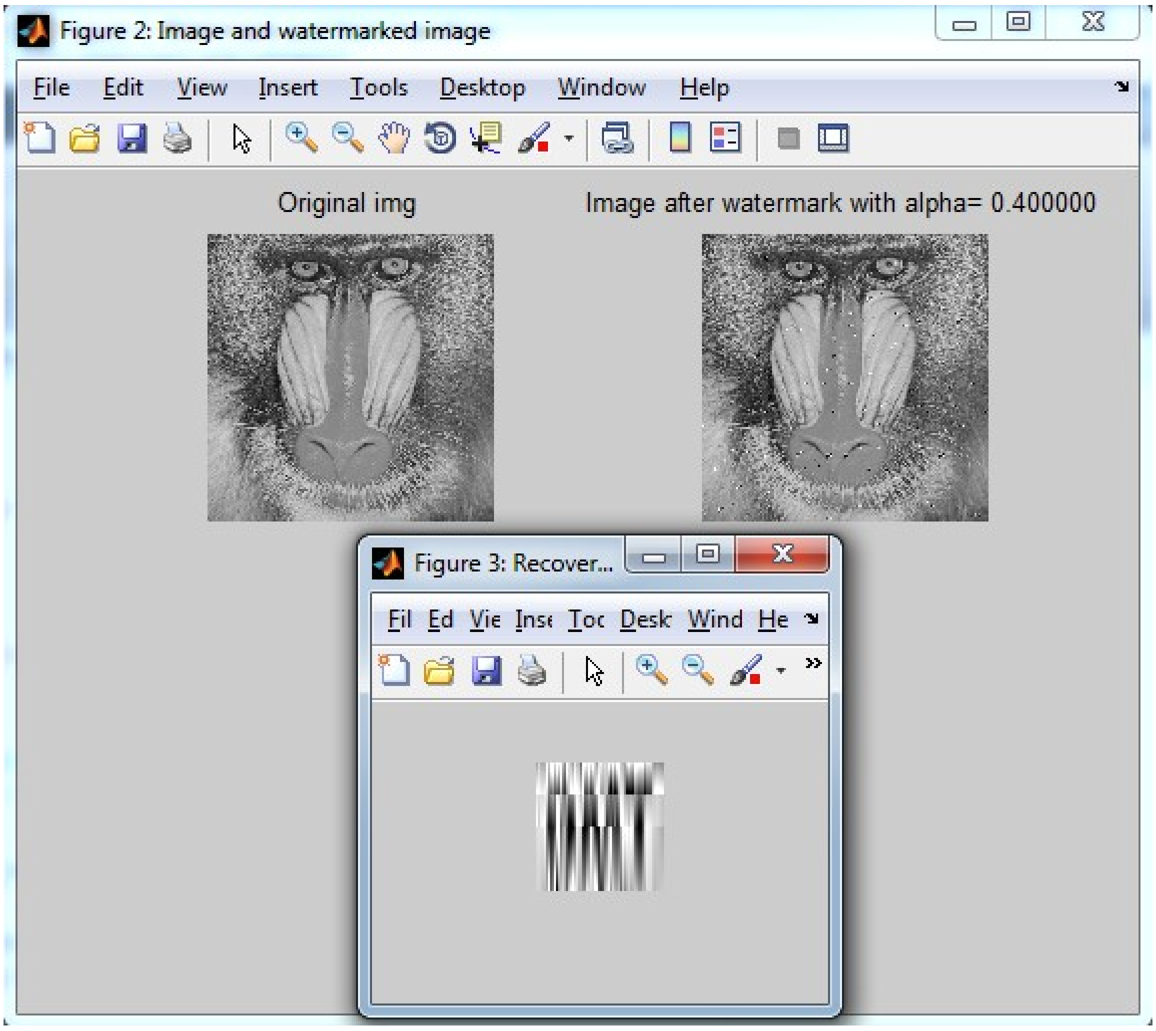


Figure 2: Image and watermarked image
File Edit View Insert Tools Desktop Window Help
Original img
Image after watermark with alpha= 0.400000
Figure 3: Recover...
Fil Ed Vie Ins Too Desk Wind He

4: results with baboon

4. Conclusions

I got to implement my interpretation of the algorithm and it seems like it works but it has some drawbacks:

1) Only works with square images of base 2 size.

2) In my experience modifies the image in a way it's visible.

3) The paper isn't specific about what to do with the “m” variable and the “ α″ variable

4) The paper is not very well written. That may bring to a lot of confusion (as I had) and maybe missinterpretations.

5) The implementation is kind of hardcoded for 512x512 images to watermark and a watermark of 64x64.

I tried to modify the watermarked image by compressing it again in jpeg and resizing it (and then back to 512x512) and the watermark stayed. I must say I didn't really think it was going to work.

The experience of implementing the paper was rather painful for the way it's written but quite rewarding to see it working.

5. How to use

There is a test file called test_insert_watermark_and_extract.m which contains a execution of the implementation:

```
% watermark to insert
W = imread('watermark_image.jpg');

% original image
%I = imread('lena.jpg');
I = rgb2gray (imread('baboon.jpg'));

rng_seed = 105;
alpha = 0.4; % the lower the better...
[watermarked_img, secret_key] = insert_watermark(I, W, rng_seed, alpha);

extracted_watermark = extract_watermark(watermarked_img, secret_key, rng_seed,
alpha);
```

The implementation is divided in the function insert_watermark and the function extract_watermark.

There is also a file called watermark_.m which was my developing file.

Also in the directory you can find the images lena and baboon to test.

And also you can find the Slant matrices from order 2 to order 13 in a zip file as I got frustrated searching for some of them to compare with the ones I was generating.

6. Attachments

Code insert watermark function

```
function [watermarked_image, secret_key] = insert_watermark(image_to_watermark,
watermark_to_insert, secret_rng_seed, alpha)
%
% Insert the watermark_to_insert
% into image_to_watermark
% using the secret_rng_seed and alpha provided
% and store the secret data
% to recover the watermark later
%
% returns the watermarked image
% and the secret_key
%

% watermark image
W = watermark_to_insert;
W_double = double(W);

% original image
I = image_to_watermark;
I_double = double(I);

N=size(I);
if N(1) ~= N(2) % image must be squared... that sucks
   error('Error: not a square image')
end

N_watermark=size(W);
% watermark image slant stuff
L_watermark = log2(N_watermark(1)); % getting the size of the image to compute
the slant matrix
S_n_watermark = sltmtx(L_watermark); % compute slant matrix
S_nT_watermark = S_n_watermark'; % compute inverse slant matrix
U_watermark = W_double; % just to use same terminology than the paper
V_watermark = S_n_watermark * U_watermark * S_nT_watermark; % (1)
originalU_watermark = S_nT_watermark * V_watermark * S_n_watermark; % (2)

% DC component is the secret key file
secret_key = V_watermark(1,1);
% AC components are used for watermark embedding

figure('Name','Watermark and Slant transform')
subplot(2,2,1),imshow(W);
title('Original watermark');
subplot(2,2,2),imshow(uint8(originalU_watermark));
title('Watermark after slant applying');

% in V we can find the transform.
% from (0,0) (top left) we find the lower freqs (DC component)
% and in the bottom right we find the high freqs (AC components)

% we need to do the V = S * U * St for the watermark image
% 64x64
% from there we get the DC component (0,0) and save it in the "secret file"

% the original image I
% decompose in non-overlapped 8x8 sub-blocks
I_sub_8x8 = mat2cell(I_double,  [zeros(1,64) + 8], [ zeros(1,64) + 8]); % 64x64
of 8x8 blocks
I_sub_8x8_watermarked = I_sub_8x8; % we make a copy to put the watermark on

% we need a seed for random which will be secret too
rng(secret_rng_seed); % supersecret seed
% get some random numbers here, quantity of them "m" in the paper
%m = (64*64); % (64*64 watermark coefficients - 1 (DC coefficient))
m_subblocks = (64*64) / 16; % and we insert the coefficients by 16 in each
subblock
m_subblocks = m_subblocks - 1; % substract last subblock that will have only 15
coefficients
% thats 256 by the way

% m-sequence random number generator for selecting a certain number of
% sub-blocks for watermark embedding
subblocks_for_water_embedding_row = randi([1 64], 1, m_subblocks);
subblocks_for_water_embedding_column = randi([1 64], 1, m_subblocks);

% in every subblock we select 16 values
% they follow scheme in paper (diagonal)
embedding_locations = [2 2; 3 2; 3 2; 3 3; 3 4; 4 3; 4 5; 4 6; 5 4; 5 5; 5 6; 6
5; 6 6; 6 7; 7 6; 7 7];


m_i=2; % m_i = 1 will be the DC coefficient so dont use it
S_n = sltmtx(log2(8)); % compute slant matrix of size 3, the size of the block
S_nT = S_n'; % compute inverse slant matrix
%alpha = 0.1; % alpha of the paper is not described... 0.6 is a lot it seems,
0.1 is not so noticeable (but it is very much)
% watermark those locations for every subblock chosen to be watermarked
for current_subblock_to_watermark=1:m_subblocks
    U_subblock =
cell2mat(I_sub_8x8(subblocks_for_water_embedding_row(current_subblock_to_waterma
rk), subblocks_for_water_embedding_column(current_subblock_to_watermark)));
    V_subblock = S_n * U_subblock * S_nT; % (1) Slant transformed
    for i=1:16  % for every location
        location = embedding_locations(i,:);
        coef_to_insert = V_watermark((uint8(m_i)/64) +1, mod(m_i,64) +1); % this
gets the next coefficient from the 64x64 watermark
        m_i = m_i+1;
        V_subblock(location(1),location(2)) = alpha * coef_to_insert;
    end
    watermarkedU_subblock = S_nT * V_subblock * S_n; % (2) recovered with the
modified coefficients

I_sub_8x8_watermarked{subblocks_for_water_embedding_row(current_subblock_to_wate
rmark), subblocks_for_water_embedding_column(current_subblock_to_watermark)} =
watermarkedU_subblock;
end

I_watermarked_double = cell2mat(I_sub_8x8_watermarked);
I_watermarked = uint8(I_watermarked_double);
figure('Name','Image and watermarked image')
```

```
subplot(2,2,1),imshow(I);
title('Original img');
subplot(2,2,2),imshow(I_watermarked);
title(sprintf('Image after watermark with alpha= %f', alpha));

imwrite(I_watermarked, 'watermarked_image.jpg','jpg');
% return the watermarked image
watermarked_image = I_watermarked
```

Code extract watermark function

```
function watermark_extracted = extract_watermark(watermarked_image, secret_key,
secret_rng_seed, alpha)
%
% Extract the watermark of a watermarked image
% given the necessary data
% secret key (the DC component)
% the random number generator seed
% the alpha used for embedding
%
% returns the extracted watermark
%

watermarked_image_double = double(watermarked_image);
watermarked_image_sub_8x8 =  mat2cell(watermarked_image_double,  [zeros(1,64) +
8], [ zeros(1,64) + 8]); % 64x64 of 8x8 blocks for the 512x512 img

% set random number generator seed
rng(secret_rng_seed);
m = (64*64); % (64*64 watermark coefficients - 1 (DC coefficient)) will be taken
into account when iterating
m_subblocks = (64*64) / 16; % and we insert the coefficients by 16 in each
subblock
m_subblocks = m_subblocks - 1; % substract last subblock that will have only 15
coefficients
% generate the subblocks that were watermarked
subblocks_for_water_embedding_row = randi([1 64], 1, m_subblocks);
subblocks_for_water_embedding_column = randi([1 64], 1, m_subblocks);

% in every subblock we select 16 values
% they follow scheme in paper (diagonal)
embedding_locations = [2 2; 3 2; 3 2; 3 3; 3 4; 4 3; 4 5; 4 6; 5 4; 5 5; 5 6; 6
5; 6 6; 6 7; 7 6; 7 7];


detected_watermark_slant = zeros(64,64);
detected_watermark_slant(1,1) = secret_key;
m_i=2; % m_i = 1 is the DC coefficient
S_n = sltmtx(log2(8)); % compute slant matrix of size 3, the size of the block
S_nT = S_n'; % compute inverse slant matrix
% extract watermark coefficients from the blocks
for current_subblock_to_detect=1:m_subblocks
    U_subblock =
cell2mat(watermarked_image_sub_8x8(subblocks_for_water_embedding_row(current_sub
block_to_detect),
subblocks_for_water_embedding_column(current_subblock_to_detect)));
    V_subblock = S_n * U_subblock * S_nT; % (1) Slant transformed
    for i=1:16  % for every location
        location = embedding_locations(i,:);
        detected_watermark_slant((uint8(m_i)/64) +1, mod(m_i,64) +1) =
V_subblock(location(1),location(2)) / alpha; % store each coef
        m_i = m_i+1;
    end
end

S_n = sltmtx(log2(64)); % compute slant matrix of size 6, the size of the
```

```
watermark image
S_nT = S_n'; % compute inverse slant matrix
detected_watermark_double = S_nT * detected_watermark_slant * S_n; % We recover
the watermark
detected_watermark = uint8(detected_watermark_double);

figure('Name','Recovered watermark');
imshow(detected_watermark);

% store in disk the watermark found
imwrite(detected_watermark, 'detected_watermark.jpg','jpg');

% return the detected watermark in imshow-able format
watermark_extracted = detected_watermark;
```