DIGITAL WATERMARKING

project documentation

Wroclaw, 19.01.2021

Multimedia and computer visualization	
supervisor: **Marek Woda Ph.D.**	
project members	
Jakub Macek	235585
Łukasz Mastalerz	235315
Krystian Wojakiewicz	235552

Contents

1 Introduction

This document is a final documentation of academic project created for Multimedia and Computer Visualization course taken at Wroclaw University of Science and Technology. Main subject of the project is related to digital watermarking methods for 2D images.

2 Motivation, business justification

Project was created to expand knowledge in the field of image processing. Application is intended to protect copyright and allow people to mark the source of original image. Moreover, project contains performance analysis for implemented digital watermarking techniques.

3 State-of-the-art

3.1 Competitive solutions on the market

There are lots of commercial applications that support different digital watermarking techniques to mark copyright on sound or graphic files. Most popular tools:

- iWatermark Pro
- uMark
- Arclab Watermark Studio

3.2 What was achieved so far on the market

Nowadays, there are a lot of applications of digital watermarking technology. From the first research at the end of last century, different algorithms and techniques are applied in many fields:

- TV audience monitoring - audio signal of TV channels is watermarked. It helps TV companies to measure market share.
- content protection - for example, sound in movies is watermarked and verified by Blu Ray players
- tracking movies from illegal sources - it is possible to detect, where and when pirate illegally recorded a movie

4 Project scope and goals

There are two main subjects in the project:

- building a desktop application that uses selected techniques to embed digital watermark into loaded image or detect digital watermark presence
- performance analysis of implemented digital watermarking methods and testing their robustness to different operations on image

Project work has been divided into two phases that are presented below.

1. Phase I
    - desktop application creation - GUI implementation with mocked data
    - implementation of Least Significant Bit digital watermarking technique
    - implementation of Discrete Cosine Transform based digital watermarking technique
2. Phase II
    - connecting all implemented parts in application
    - test robustness of implemented methods, performance analysis
    - project documentation creation

5 Watermarking methods

There are two digital watermarking techniques implemented in the project.

5.1 Least significant bit

Digital image can be simply described as a set of pixels. Pixel is the smallest unit of an image and represents the values of red, green and blue colors brightness in particular place.
Color values kept in each pixel are in the range from 0 to 255 and represented in **binary** from - coded as 8-bit values. Secret message needs to be converted to binary from. It can be used with ASCII table. For example, "secret" string can be converted to:

```
01110011 01100101 01100011 01110010 01100101 01110100
```

binary code. **Least Significant Bit** technique iterates over pixel values and replaces least significant bit with next numbers of binary code obtained from secret message.

LSB technique implementation used in project also uses special delimiter that is appended to secret message that should be hidden. Role of such delimiter is to inform that message is already encoded, it's also helpful to decode the secret message from watermarked image.

5.2 Discrete cosine transform

The discrete cosine transform (DCT) method of watermarking a digital image is an example of a frequency domain watermark. It involves calculating the DCT of an image and extracting a given number of the most significant DCT coefficients. This ensures, that the watermark is embedded in the most important "parts" of the image, which makes it harder to remove or weaken the watermark. The fact, that the additional data (the watermark itself) does not correspond to a spatial area and is thus much harder to detect with the naked eye. After we have calculated the DCT and extracted the coefficients, we can embed the watermark. We do this by adding a pseudo-random, normally distributed vector to the original coefficients. Below is the formula used for calculating the new coefficients:

$$new\_coeff = original\_coeff * (1 + random\_vector * \alpha)$$

To decode the the watermark, we must store the changed coefficients (their indices) and reverse the formula. To detect the watermark, we use the similarity between the vectors of DCT coefficients of the original image and the one we are testing. The similarity is represented by the following formula:

$$\frac{V \cdot W}{\|V\|}$$

The larger the value of this fraction, the more similar the two vectors are. We can validate, whether the image has been watermarked by comparing this value to some set standard.

6 Technology stack

Main programming language used in project is **python3**. Desktop application is based on pyQT framework. OpenCV python library is used to work with images.
Project repository was hosted on Github. TravisCI tool was used for code analysis. Trello web application was used for planning activities and progress tracking.

7 Application setup

To run an application, python3 virtualenv creation is recommended. In virtual environment, simple command

```
$ pip install -r requirements.txt
```

is needed to download and install all required libraries.

8 Quickstart

After you have activated the python virtualenv and installed all libraries, the easiest way to run application is to write in console

```
python main.py
```

It will run the GUI module for the app, from where you have access to all watermark methods.

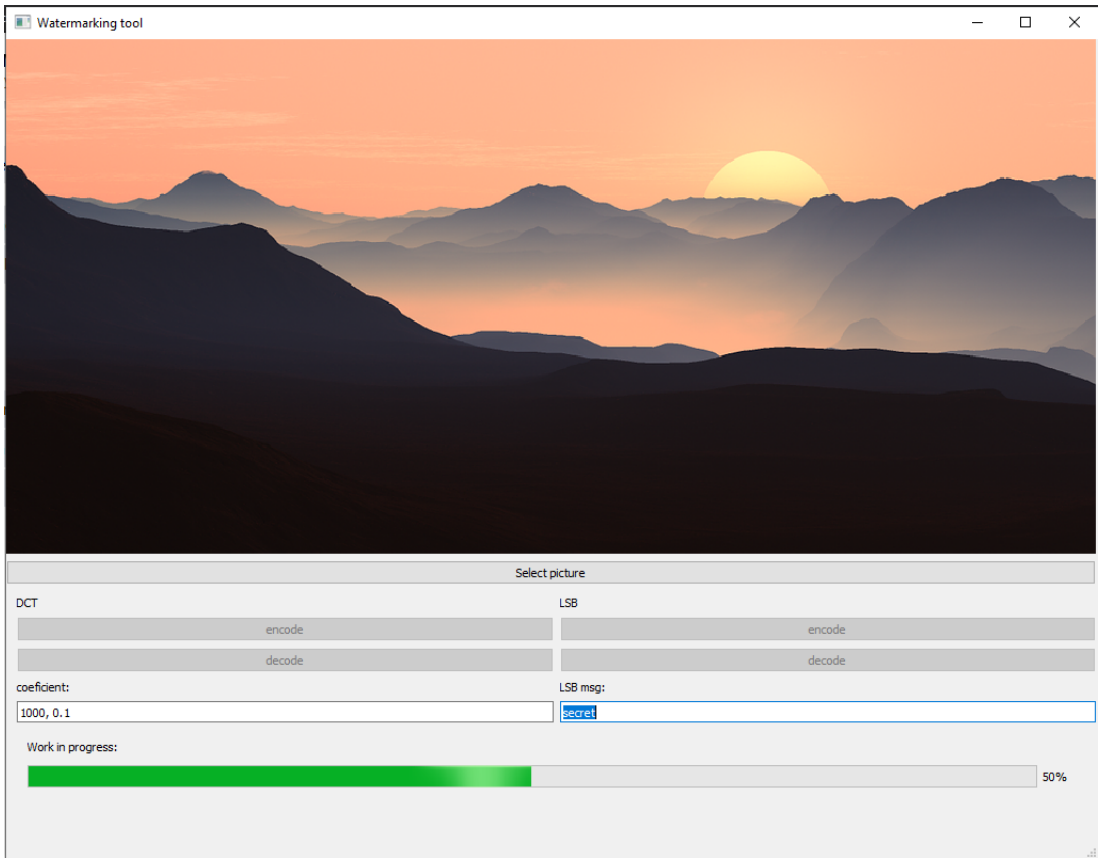

Figure 1: Running application

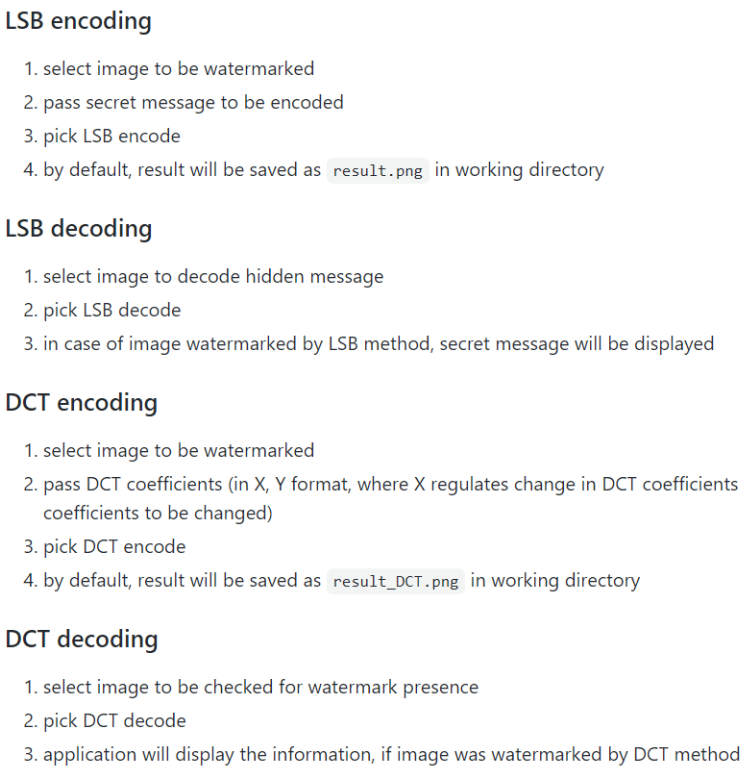

Figure 2: Possible options

9 Performance tests for watermarked images

To check the watermarks resistance on changes on the images we decided to modify it in a few way:

- Rotation
- Resize
- Normalization
- Histograms equalization
- Adding noise
- JPG conversion

And after each modification we were checking if watermark still can be found in the image.

9.1 Performance tests for LSB method

All tests for LSB method can be found in */test/performance_analysisLSB.py* in project directory.

9.1.1 No modification

The first test was just simple checking if without any modification, the watermark is in the image. The reason of it was to check if the watermarked file used for tests isn't corrupted in any way.

```
no_modification :  PASS
```

9.1.2 Rotation

This test case show us that the LSB method isn't invulnerable to images rotation, but after the image is rotated back to the original orientation the watermark can be found.

```
after_rotation_90 :  FAIL
after_rotation_180 :  FAIL
after_rotation_90_and_back_to_normal :  PASS
after_rotation_180_and_back_to_normal :  PASS
```

9.1.3 Resize

LSB method isn't resistant to a resize operation. Even resizing the picture back to the original size doesn't help like it did in case of rotation. Probably a bits where the watermark is hidden are removed during resize operation.

```
resize_to_150 :  FAIL
resize_to_50 :  FAIL
resize_to_50_and_back :  FAIL
```

9.1.4 Normalization

After normalization operation the watermark still can be found in the image.

```
normalization :  PASS
```

9.1.5 Histograms equalization

After using the histogram equalization the watermark isn't available in the picture.

```
histograms_equalization :  FAIL
```

9.1.6 Adding noise

After adding Gauss noise to the picture, the watermark is disappearing, no matter what parameters were used to generate the the noise. But with salt and peeper the story is different, because the watermark detection is random. In our opinion it's depend of where the distortion was added in the picture. If watermark bits weren't affected, it still can be found.

```
gauss_noise :  FAIL
salt_and_peeper :  FAIL / PASS
```

9.1.7 JPG conversion

The JPG conversion is completely changing binary form of the image, so it was expected that after that operation the watermark will disappear. But to be sure made this test. Also converting the image back to PNG doesn't restore the watermark in the picture.

```
jpg_conversion :  FAIL
jpg_conversion_and_back_to_png :  FAIL
```

9.2 Performance tests for DCT method

All tests for the DCT method can be found in */test/dct/PerformanceDCT.py* in the project directory. The results for a particular test case are determined based on the similarity value. The original images showed similarity values of around 30, so we decided that anything below 10 is not satisfactory and shall be failed.

9.2.1 No modification

The first test we performed was to check, whether the watermark has been correctly embedded in the image and can be detected by our algorithm.

```
no_modification :  [[31.09772195]] :  PASS
```

9.2.2 Rotation

This tests verifies whether the watermark is vulnerable to simple rotations of the tested image. We tested two scenarios: rotate by 90 degrees, rotate by 90 degrees and back. This test showed that the rotation of the image matrix is easily able to break the watermark.

```
after_rotation_90 :  [[-0.4793732]] :  FAIL
after_rotation_90_and_back_to_normal :  [[31.09245197]] :  PASS
```

9.2.3 Resize

In this case we scaled the image down to 10% of the original value and back to their original size four times. After the first scaling the watermark was still easily detectable, but after four such scalings, the watermark was no longer passing our criteria, although the image itself was no longer valuable at this point.

```
   single_scaling :  [[29.13056862]] :  PASS
after_multiple_scalings :  [[6.91130751]] :  FAIL
```

9.2.4 Normalization

After normalization of the test image, the watermark can still easily be found in the image.

```
normalization :  [[25.91971214]] :  PASS
```

9.2.5 Histograms equalization

After using the histogram equalization the watermark is still barely detectable in the image, but the margin is not very large. Unfortunately for the attacker, this method can impact the quality of the original image pretty significantly.

```
histograms_equalization :  [[14.7498582]] :  PASS
```

9.2.6 Adding noise

After adding Gaussian noise to the picture doesn't affect the DCT watermark in any relevant way. The reason for this might be that the Gaussian noise is spatial in its nature, hence it's not as "visible" in the frequency domain. The same conclusions could be drawn from the salt and pepper test case.

```
gauss_noise :  [[31.51980348]] :  PASS
salt_and_pepper :  [[30.86336659]] :  PASS
```

9.2.7 JPG conversion

The JPG conversion is a lossy compression, meaning that some of the original information of the image is lost during the process of compression. Interestingly, the JPG standard also uses the DCT to extract and process the DCT coefficients. The important part is that, JPG does the exact opposite of what our watermark does, it searches for the **least** significant coefficients and removes the data corresponding to them. This causes a large drop in file size, but since for the most part, it doesn't affect the most significant coefficients, the watermark is unharmed.

```
jpg_conversion :  [[31.63033716]] :  PASS
jpg_conversion_and_back_to_png :  [[31.63033716]] :  PASS
```

10 Summary

Firstly, project helped us to better understand steganography problem and possible methods for hiding data in the images. Through tests and analysis it can be said that it's not so easy to implement digital watermarking technique which will be resistant to different operations on images. For commercial programs, we think that the key is to provide a stable tool that will keep the watermarked data in case of image rotations, scaling etc. It can be a quite interesting issue to dive into and learn, what are the possible ways to improve the algorithms and make it better.