

An Image Watermarking Tutorial Tool using Matlab

Kevin Heylen^a and Tim Dams^{a,b}

^aUniversity College of Antwerp, Paardenmarkt 92, Antwerpen, Belgium

^bVrije Universiteit Brussel - IBBT, Department ETRO, Pleinlaan 2, Brussel, Belgium

ABSTRACT

Digital watermarking of images, the act of hiding a message inside an image, is still a young, yet growing, research field. We have developed an environment in Matlab that allows researchers, teachers and students alike to get acquainted with the concepts of digital image watermarking techniques. This user-friendly environment is divided into two parts. First there is the educational part, which visualizes watermarking techniques and gives users the possibility to observe the results of various attacks. This part can be used as a demonstrator during lectures on image watermarking. The second part is dedicated to a benchmarking tutorial section, which allows users to easily compare watermarking techniques. A user new to the field of benchmarking can simply insert existing or newly developed watermarking algorithms, attacks and metrics into the benchmarking tutorial environment. We also included an attack section that is easy to adjust and extend with new attacks. Furthermore, we provided a report generator that will summarize all the benchmarking results in a clear, visually appealing manner. Through the use of Matlab, we were able to make a framework that is easy to modify, update and expand and also enables everyone to use the existing signal processing libraries of Matlab.

Keywords: watermarking, tutorial, educational, benchmark, Matlab, user interface

1. INTRODUCTION

Research in digital watermarking is growing, partly due to the continuously increasing availability of Internet bandwidth for home-users. However, because of the explosion of digital multimedia distribution services over the Internet (e.g. Youtube, Google Images, Napster, etc) the inherent copyright issues are also appearing. Digital Rights Management (DRM) systems are used frequently, giving producers of digital content the ability to protect their works. Protection through encryption is one solution. However, once the file is decrypted the producers have no longer control over their protected works. With digital watermarking,¹ one is able to maintain some control by embedding copyright and/or fingerprinting² information inside the multimedia. As a result, research in robust and fragile multimedia watermarking is accelerating. In this paper, we present an open-source Advanced Watermarking Toolbox (AWT) developed in Matlab for researchers, developers and anyone who is planning to work on image watermarking.

1.1 Preliminaries

A **digital watermark** w is a message that is embedded inside another signal, called the **coverwork** data c , which can be any type of data, such as images, audio, etc. Depending on the application at hand, a digital watermark can both be invisible or visible for the consumers of the coverwork data. When dealing with invisible watermarks, the message being embedded has to be as imperceptible as possible in order to hide its existence. For the remainder of this paper we will focus only on embedding and detection of invisible, robust watermarks where the coverworks are images.

1. We define the **embedding function** f_e as the following mapping:

$$f_e : I_c \times K \times W \rightarrow I_w,$$

where I_c denotes the set of coverwork images, K the set of secret keys, W the set of watermark messages and I_w the set of possible watermarked images. Hence f_e turns the coverwork $c \in I_c$ into a **watermarked work** $x \in I_w$ using a key $k \in K$ and watermark message $w \in W$.

2. During transmission of $x \in I_w$ one or more intentional and unintentional **attacks** f_a can occur:

$$f_a : I_w \times P \rightarrow \tilde{I}_w,$$

with P being the set of attack parameters used and \tilde{I}_w the set of possibly *distorted* watermarked images.

3. At the receiver side, two commonly used types of **detection functions** f_d are:

- a **non-blind** detector requires the original coverwork $c \in I_c$ in order to have a partial or full detection of a watermark message $w \in W$:

$$f_{d,non-blind} : \tilde{I}_w \times K \times I_c \rightarrow W$$

- a **blind** detector has no need of the original coverwork I_c :

$$f_{d,blind} : \tilde{I}_w \times K \rightarrow W$$

4. A **watermarking algorithm** is a couple (f_e, f_d) such that for $x = f_e(c, k, w)$ (in the blind case)

$$f_d(f_e(x, p), k) = w.$$

Comparison of watermark algorithms is based on several parameters, namely:

- **Robustness and fragility:** Robustness describes how resistant a given watermark algorithm is against both intentional or unintentional attacks. Depending on the type of application, one can also wish to have a fragile watermarking algorithm, as opposed to a robust algorithm. With fragile watermarks, one needs to detect even the smallest change made to the embedded coverwork I_w , for example to test the authenticity of the coverwork I_c .
- **Perceptibility:** Describes the amount of distortion that is created in the coverwork image after embedding a watermark. The perceptibility of invisible watermarks needs to be as low as possible.
- **Payload:** The amount of bits that can be embedded, using a given embedding function f_e and a given coverwork c . This can have a severe impact on both robustness and perceptibility: an increase in payload will result in a decrease of robustness and/or increase of perceptibility. It should also be mentioned that the dimensions of the image c need to be taken into account when describing the payload, since a larger image can contain an equal amount of bits for a smaller perceptibility penalty (or a better robustness for that matter).
- **Embedding strength:** Usually a strength-factor α is defined which describes how strongly a given watermark is embedded. As a rule of thumb, it is safe to say that an increase of α will result in an increased robustness and perceptibility.
- **Key:** As stated by Kerckhoffs' principle,³ a cryptosystem's security can only depend on a secret key; the algorithm itself can be made public. The way the secret key k is used in a watermarking algorithm varies: usually the key and the message will be used as a seed for a *pseudo-random number generator* (PRNG). The generated random stream will then be embedded. The choice of k should not have any influence whatsoever on the robustness and/or perceptibility. However, the set of keys K has to be large enough, in order to avoid brute-force attacks that try to find the secret key k .



Figure 1. Embed (left) and attack (right) interfaces.

1.2 Goal

The creation of AWT started as a bachelor thesis in which an image watermarking demonstrator was built using Matlab, the results of which were published previously.⁴ The work we present here is the result of a follow-up Master thesis. The main goal of AWT is to provide an extensive toolbox that can be used by tutors and aspiring researchers alike to get acquainted with the different concepts of digital image watermarking.

We chose to develop the environment using Matlab enabling us to use the existing libraries and toolboxes (e.g. Image processing toolbox, Report Generator toolbox). AWT consists of two major environments, an educational and benchmarking environment. The educational environment can be used as a visual aid during theoretical lectures on the basics of watermarking. Each step of the watermarking process (embedding, attack, detection, measurements) can be shown in detail and also compared when one changes one or more settings. However, since AWT is developed in Matlab, tutors can also use it in practical sessions where the students have to incorporate their own algorithms, or change existing ones.

Both environments are developed with the graphical user interface designer of Matlab and have different goals. The educational environment needs to give a clear overview of the entire watermarking process, whereas the benchmarking environment focuses on the simplicity of adding new algorithms, giving the user the ability to write attacks and easily review results with the report generator. The remainder of the paper is organized as follows. The educational environment is discussed in Section 2, including an overview of image watermarking techniques. The benchmarking environment is discussed in Section 3. In Section 4 we will present our conclusions.

2. EDUCATIONAL ENVIRONMENT

The educational environment of AWT focuses on the visualization of an entire watermarking process. It is thus the ideal companion tool for teachers, students and researchers to grasp the general principles of image watermarking, as explained in the excellent book by Cox et al.¹ As can be seen in Figure 1 (left), we designed an easy to use overview interface where the user can choose which embedding function f_e to use. In order to have a more visually appealing tool, the user can choose the secret key k and watermark w to be a binary image or a string. Both the image or string will then afterwards be converted to a binary string that will be used throughout AWT.

Using the tool, the user can review the different techniques that were developed over time. Starting with the most basic embedding scheme, Least-Significant-Bit (LSB) embedding, in which the embedder simply changes the LSB of each image pixel according to the binary watermark w . For example, in Figure 1 (left), we created a binary image with ‘copyright’ shown, which will act as the watermark w . Usually this message is created using a *pseudo-random number generator* as explained earlier. For the remainder of this chapter, we will use the LSB method to show how AWT can be used to introduce certain key properties of digital image watermarking.

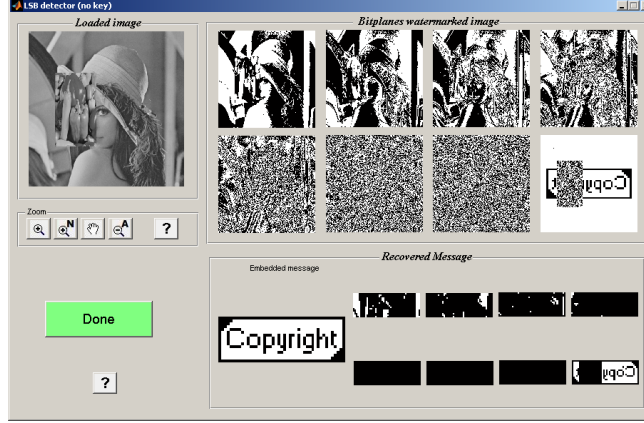


Figure 2. Decoding after flipped attack. Notice the decoding is done in each bitplane, only the LSB is partially decoded correctly(as should be) but flipped.

2.1 Measuring robustness and perceptibility

Embedding a watermark introduces a distortion on the coverwork c which we can measure using a distortion metric such as the Peak-Signal-to-Noise ratio (**PSNR**). Other metrics exist that also measure the degree of distortion between two images (e.g. weighted PSNR,⁵ CMPSNR (Color Masked PSNR),⁶ SSIM (Structure Similarity),⁷ etc.). We kindly refer to the paper by Kutter et al.⁸ for more information on different quality metrics, as well as an excellent overview on benchmarking. Since PSNR is still one of the most used metrics nowadays when comparing techniques, AWT uses this metric by default. However, AWT can easily be extended with other metrics.

Let M be the largest possible pixel value in the type of images being used, then the PSNR between the image Im_1 and Im_2 is defined as:

$$PSNR = 10 * \log_{10} \left(\frac{M^2}{\sigma^2} \right) \quad [dB]$$

where

$$\sigma^2 = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} ||Im_1(i, j) - Im_2(i, j)||^2$$

with m and n the dimensions of Im_x (note that these have to be equal in size), and $Im_x(i, j)$ the value of the pixel at coordinates (i, j) in image Im_x . The PSNR tends to infinity if both images are identical, lower PSNR values (in dB) mean larger distortion. The distortion introduced because of a watermark embedding is measured as the PSNR between the watermarked image x and the original coverwork c . LSB embedding introduces a PSNR of 48dB which is near invisible for the Human Visual System (**HVS**).

After measuring the perceptual impact of the watermark, we can test the overall robustness of the algorithm. We have chosen to measure the robustness using the **Bit Error Rate** (BER) as follows:

$$BER = \frac{\tilde{w}_{wrong}}{w_{total}}$$

with \tilde{w}_{wrong} the number of incorrectly decoded bits and w_{total} the total number of bits of the original watermark w . In other words, a BER of 0 means that each bit was correctly decoded. As expected, the BER of our LSB scheme, with no attacks, equals 0.

2.2 Attacks

Many different attacks can happen on a watermarked image x , whether intentional or not.⁸ Depending on the type of attack, the original watermark message w might become corrupted. Using AWT one can simply test the

Common Class	Available processing techniques
Compression	JPEG-like
Color Manipulation	Intensity Gamma Correction Component Adjustments
Noise	Add Noise Denoise Replace bitplanes
Geometric Distortions	Rotate Resize Remove column/row Crop Shear
Data Composition	
Multiple watermarks	
Other	Edge detection

Table 1. Attack classification in attack GUI.

robustness of a given technique when applying an attack. In the new version of the educational environment, we completely redesigned the attack interface. We sorted every attack into seven different sets, which is illustrated in Table 1. This makes the interface more intuitive and less complex to use. A user is given the possibility to apply one or more attacks on I_w and then measure the BER and distortion. In Figure 1 (right) we show the updated attack GUI.

Our LSB works flawlessly when no attacks are applied (BER=0). When any type of noise is added to x , the BER rapidly increases. With AWT one can easily show the effect of a geometric attack. If we apply a horizontal flip-attack on I_w this will result in a very high BER, however, visually the *copyright* message would be visible, but flipped as well. If we now apply an attack that adds another smaller image on top of x (data composition attack) we can see that the overlapped part results in wrong decoding. In Figure 2 we clearly see how w gets decoded if we apply a flip and data composition attack (we added a small *peppers* image). In the lower right we see the decoded message in each bitplane. Since we are testing LSB, the message is partially recovered in the LSB-plane.

2.3 Embedding in other domains, planes and color components

One can use the properties of the HVS to improve the robustness and/or perceptibility of a watermarking algorithm. For example, in AWT we can choose to embed in different parts of the image:

- Other color components: When embedding in a color image, we can choose to embed inside one or more color layers. For example, we can embed only in the Blue layer of a RGB image.
- Other transform domains: The HVS is more sensitive for certain frequencies inside an image than others. By transforming our image to the frequency domain we can embed w in certain frequencies. Or we can embed more strongly (adaptable) inside those frequencies that the HVS is less sensitive to. This is shown in the techniques labeled 'DCT' (Discrete Cosine Transform⁹) and 'DWT' (Discrete Wavelet Transform¹⁰) in AWT. Both techniques use a correlation-based detection technique (see Section 2.4).

2.4 Supported techniques

The simple LSB-technique is mostly used as a fragile watermark, due to its sensitivity to noise. *Spread-Spectrum* based embedding techniques (SS) however are more robust against noise attacks, which can be shown using the technique labeled 'Correlation' in AWT. The general idea of SS techniques is to generate two random bitpatterns

w_0 and w_1 with the *pseudo-random number generator* and the key k as secret. Both these patterns have the same dimensions as the coverwork. One of these patterns is added to the coverwork as follows:

$$x = c + \alpha \times w_i,$$

with w_i depending on the message to embed (e.g. w_1 is used if we wish to embed 1) and α being the embedding strength as explained in Section 1.1. We now have embedded one bit. However, we can simply increase the payload by dividing the coverwork into N smaller subimages. Each subimage is now embedded as before, but with the bitpatterns also resized N times. Using a correlation detector, the message can be decoded again. An improvement on this technique is *Direct Sequence Code Division Multiple Access* (DS-CDMA),¹¹ which will generate more than 2 patterns in order to increase the capacity of SS techniques. Of course, this goes at the cost of an increased perceptibility or decreased robustness. DS-CDMA can also be tested in AWT, including in the DCT and DWT domain as explained earlier.

SS techniques, though more robust, still introduce a large distortion penalty. Recently, *Quantization Index Modulation-based techniques*¹² or *dither modulation codes* were introduced. These techniques, based on Costa's 'Writing on Dirty Paper',¹³ view the watermarking model as a communication channel with side information. Instead of adding a noise-like pattern to the coverwork, QIM techniques are quantization-based. The coverwork c is quantized using two scalar quantizers Q_0 and Q_1 as follows:

$$x = \begin{cases} Q_0(c) & w = 0 \\ Q_1(c) & w = 1 \end{cases}$$

The hardest part in creating a QIM-based technique, is the construction of the quantizers. A wrong choice of quantizers Q_0 and Q_1 will increase the perceptibility or decrease the robustness against certain attacks. Only the basic QIM scheme is implemented in AWT. Future work will include more advanced quantizers, for example trellis and lattice-based quantizers.¹⁴

3. BENCHMARK TUTORIAL ENVIRONMENT

Whereas the educational environment merely provides insights in how different techniques work, the benchmarking part shows how to compare two or more techniques. When developing a watermarking algorithm, the most important step is the validation of the algorithm, which needs to occur in a uniform way, typically called *benchmarking*. Algorithms are then compared on their key parameters (as discussed in subsection 1.1) after applying a range of tests that include different images, attacks and settings.

3.1 Existing Image Watermarking Benchmark Systems

In the past, several benchmarking programs were developed to evaluate these watermarking algorithms. We now list the most well-known and frequently used benchmark systems:

- *StirMark*¹⁵ is still one of the most widely used benchmarks for image watermarking. This open source C++ program was developed in 1997 by Fabien A. P. Petitcolas, a pioneer in the watermarking world. Stirmark 3.1 was the first benchmark application and was developed for digital image watermarking. In 2000 Stirmark 4.0 was released. In this release, the attack module was expanded and also profile support was added. The relevance of the PSNR metric, which Stirmark uses, is limited when applying geometrical attacks. More information on <http://www.petitcolas.net/fabien/watermarking/stirmark>.
- *Checkmark*¹⁶ was built to improve Stirmark. This open source program, developed in Matlab, includes several more advanced attacks like JPEG2000 compression and geometric attacks. Most of the attacks in Stirmark 3.1 are also provided within Checkmark. Furthermore, it also supports additional metrics like the weighted PSNR (wPSNR)⁵ and the Watson distance,¹⁷ which is a more useful metric when working with geometric attacks. More information on <http://watermarking.unige.ch/Checkmark/index.html>.

- *Optimark*¹⁸ is a benchmarking platform providing a graphical user interface. Once the test images and algorithms for embedding and detection are provided, Optimark will guide you through the benchmarking process. In 2002 a new version of Optimark was released that includes ROC curves for false-positive detection rate. More information on <http://poseidon.csd.auth.gr/optimark/>.
- The last few years web-based applications gained tremendously in popularity. This evolution can also be noted in watermark benchmarking programs. One of these benchmarking programs is the *Watermarking Evaluation Testbed* (WET),¹⁹ which was developed by the Video and Image Processing Laboratory at the Purdue University. Another web-based benchmarking platform is *Openwatermark* (see <http://www.openwatermark.org/>), which is a project of TELE, the Communications and Remote Sensing Laboratory of the Université Catholique de Louvain and was developed in Java. *Certimark*²⁰ (see <http://www.certimark.org/>) is a more advanced web-based benchmark platform. This program provides the possibility to easily add images, algorithms and additional attacks. The results generated by the benchmark can be further processed in different analysis tools. The tool can be used as a final validation of the newly developed algorithm and it also generates certificates which proves the strength of the algorithm.

Although the potential and options of these benchmarking platforms are fully adequate, it can be a daunting task to learn to work with them. For example, some of these existing benchmarks require experienced C/C++ or Java programming skills, making them difficult to master for students not acquainted with the language at hand.

3.2 Architecture of the benchmark tutorial

The goal of our benchmarking environment is not to emulate existing benchmarks, but to provide new researchers a glimpse on the complexities of thorough benchmarking. Our benchmarking tutorial environment provides a simple script generator and easy to read reports that contain benchmarking results (using Matlab's Report Generator) enabling users to quickly compare techniques. It should be noted that AWT, though easy to use, is not an optimal benchmarking environment performance-wise. For this reason, future versions of AWT will use the Distributed Toolbox of Matlab and we will improve the efficiency of the benchmark tutorial environment.

During the setup of a benchmark in AWT, one needs to specify the types of attacks to apply, what images to use and what should be measured afterwards (e.g. robustness, perceptibility, etc.). The benchmark environment needs a benchmark script as input, which describes the tests to perform. Figure 3 (left) illustrates the main architecture.

The script can be created in two ways:

1. A wizard-like user interface built to be accessible for first time users (see Figure 4). This wizard will guide the users through four stages (input, watermark, attack and benchmark) which we will describe in section 3.2.1.
2. More advanced users can manually type the script in any text editor.

Once the script is ready, the actual benchmarking can start. This is shown in Figure 3, where we see how a benchmark loop is executed based on the information in the script. The workflow algorithm should be self-explanatory.

3.2.1 User Interface

The wizard-like user interface is developed to create an easy start for users that are new to watermark benchmarking. The wizard is divided into four modules as shown in Figure 4 and are discussed in detail hereafter. Each module, visualized in a separate window, is designed to update the benchmark script that will be used by the benchmarking algorithm.

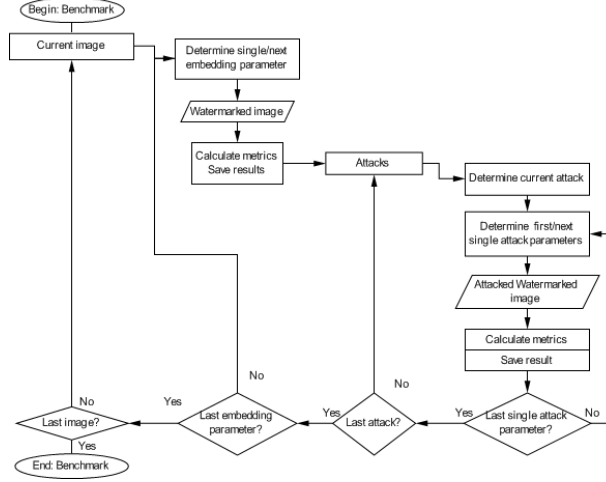


Figure 3. Benchmark algorithm workflow.

- A) *Input sections:* In the first part of the benchmarking program the user will need to specify the input parameters (such as the secret key k , described in Section 1.1). When choosing images for benchmarking, the user needs to consider that the program will only support *.tif* images. The coverworks (I_c) are made available to the watermarking algorithms in the unsigned 8-bit integer format of Matlab. We also provide the possibility to load in a dataset of images (i.e. more than one image) located in a folder. The user can easily configure parameters such as the color model in which to embed, etc.
- B) *Watermark sections:* In the watermarking step, the user first has to define whether the scheme is blind, semi-blind* or non-blind. Next, used domain, in which to embed, needs to be specified (spatial domain, frequency domain, wavelet domain, ...). Finally, the watermarking algorithm (a couple (f_e, f_d)) that will be benchmarked is selected, in which the user can specify the name, type and value of all required parameters.
- C) *Attack sections:* In this section we define the attacks that need to be applied during the benchmark on the watermarked images (I_w). Usually, a watermarking scheme is developed to be robust against a range of specified attacks. So, to verify if the algorithm is indeed sufficiently robust or to measure the robustness against other types of attacks that might occur, we made a classification of the attacks (slightly different from the educational environment) which simplifies the way to choose what attacks to apply. In each of these categories, we have predefined several sets of attacks with the minimal and maximal ranges set per attack. As a developer you can use these ranges to determine the strength and the relative weight of an attack. We have chosen to use the attack ranges defined by Certimark²¹ in the User Interface.
- D) *Benchmark sections:* The final step in the configuration of the wizard-like benchmarking process is specifying the report options. The user can choose which metrics (see Section 3.3.1) that will be used to compare the distortion in every watermarked (I_w) and attacked image (\tilde{I}_w). Furthermore the user needs to specify the report settings. The basic report that will be generated will have an XML structure, which is created using Matlab's Report Generator. However, this format is not suited for easy reviewing of the benchmark results. Therefore, after the benchmarking is finished, the report will automatically be converted into the format the user has defined. We provide three possible file formats, namely PDF, RTF and HTML, but other formats are possible with the Report Generator.

3.2.2 Script

Most advanced users and developers prefer not to run through the benchmarking wizard every time they want to benchmark or test their algorithm. For this reason we added the possibility of manually typing the script. An

*A semi-blind detector has no need of the original cover image I_c but does need some extra information that is not included in the watermarked image \tilde{I}_w .

excerpt from a possible script is shown below:

```
...
'Input parameters','',
'path_dataset','Input Images\Dataset1'
'name_embedding_layer','Red Layer'
'message_type','text'
'key_size', 64
...
```

3.3 Benchmark tutorial output

3.3.1 Benchmarking Metrics

The choice of distortion metric has an important impact on the interpretation of watermark benchmarking results. By default, we allow use of both the PSNR and SSIM as distortion metrics (see Section 2). We have chosen these two metrics since they are designed with different perspectives. The SSIM metric uses the HVS to determine the relative impact of the changes, while the PSNR measures global distortions in the image. An image that is shifted one pixel row to the left will result in a extremely low PSNR, even though an observer will view the image as barely changed. SSIM on the other hand will return a very high value (1 meaning exactly alike). Thus, one has to pay special attention to the metrics used when interpreting the benchmarking results. Note that other metrics can be added in AWT.

3.3.2 Benchmarking reports

The Matlab Report Generator is a versatile tool that allows the creation of reports of the data and workflow in AWT. Several basic report schemes are predefined (PDF, RTF and html). A default report from an AWT benchmark consists of 4 main parts:

1. **Preliminaries:** These include a simple front page and an automatically generated Table of Contents.
2. **General Summary:** This chapter will list all the benchmark specific settings such as the used watermark algorithms (and its parameters), the types of attacks (and its parameters), etc. Using this information, one is able to recreate the benchmark on another system.
3. **Perceptibility:** In this chapter we list the values of the distortion metrics, both in tables as in a graphical manner (see Figure 5). If a dataset of images was used, this information is grouped in one clear graph and table. In a future version, we will also generate mean graphs and tables in order to illustrate the mean distortion when a collection of images was used in the benchmark.
4. **Robustness:** Here we show the robustness of the detection function after having applied one or more attacks (or none). This includes the BER performance per attack, and the resulting quality after having applied an attack. Again, all results are grouped per attack and shown in easy-to-read graphs and tables.


4. CONCLUSION

We have successfully built an open watermarking tutorial toolbox (AWT) in Matlab that can be used to rapidly become acquainted with the basics of image watermarking. The educational environment can be used for first time users and teachers to quickly learn the basic concepts of different watermarking algorithms. The benchmark environment can be used either by completing a wizard or by writing a small script containing all the settings. This makes our benchmark more user-friendly than the existing benchmarks and it can also be used as a development environment as our default attacks are derived from the European guidelines,²¹ our benchmark tool can be used as a reference to compare different watermarking algorithms. Upon completing the benchmark, a detailed report is generated containing information, tables and graphs of the results. Future work on AWT will include more advanced watermarking schemes (e.g. variants on QIM,¹⁴ such as lattice QIM, etc), the ability to benchmark watermarking algorithms with error-correction codes and compatibility with Matlab's Distributed Toolbox. The most recent version of AWT can be obtained through our website <http://www.e-lab.be/MMG>.

Benchmark-Wizard

Input section

Choose image:



Settings

Embedding Component:
 Color Model:
 Component:
 Message:
 Message:
 Key:
 Size: Bit
☒ Done

Summary
 Input Path: E:\MAPFF\Heylen Kevin\Software\WWT\Input Images\Dataset11
 Embedding layer: Red Layer
 Message: Test123
 Size: 64 bit

Watermark section
Attack section
Benchmark section

Benchmark-Wizard

Input section
Watermark section

Select an algorithm

Scheme:
 Domain:
 Embed technique:
 Detect technique:
☒ Done

Parameters

Enter the total number of additional embed parameters:
 Enter the total number of additional detect parameters:
☒ Done

Summary

Scheme: Blind
 Domain: Spatial Domain
 Embed algorithm: embed_LSB.m
 Embed parameters: BlockSize:20
 Detect algorithm: detect_LSB.m
 Detect parameters: blocksize:20

Attack section
Benchmark section

Benchmark-Wizard

Input section
Watermark section
Attack section

Attack Profile

Choose an attack profile:

Compression | Color Manipulation | Enhancements | **Geometric** | Noise

☒ Rotation [-2 -0.5 0.5 2]
☒ Rotation-Crop [5 0.5 0.75 1 2 5 10 15 30 45 90 180 270]
☒ Rotation-Scale [5 0.5 0.75 1 2 5 10 15 30 45 90 180 270]
☒ Crop [5 25 5]
☒ Flip(Horizontal/Vertical) [1 2]
☒ Scale [5 25 5]
☒ Shear X [0 0.01 0.02 0.04 0.6 0.08 0.1 0.15 0.2]
☒ Shear Y [0 0.01 0.02 0.04 0.6 0.08 0.1 0.15 0.2]
☐ *Resize geometrical attacked images
 Interpolation method:

☐ Done

Benchmark section

Benchmark-Wizard

Input section
Watermark section
Attack section
Benchmark

Comparison

Choose two metrics to determine the difference between the processed images
 Metric1:
 Metric2:
☒ Done

Filetype selection

Choose the filetype which contains the benchmarking results:
☒ pdf
☐ html
☐ xtf

Notification

☒ Notify the loaded list when benchmark has finished

Enter Sender's details

Gmail Account:
 Gmail Password:

Enter Receptionists

Selected email list:

Email Properties

Subject:
 Message:
☒ Include report in attachment

Figure 4. Different stages of the wizard to generate the benchmark script: input (upper left), watermark (upper right), attack (lower left) and benchmark (lower right) sections.

Table 2.1. Watermarking results PSNR

Setting/Image	_Bitplanes_[5] _Blocksize_8	_Bitplanes_[5] _Blocksize_10	_Bitplanes_[5] _Blocksize_12	_Bitplanes_[5] _Blocksize_14	_Bitplanes_[5] _Blocksize_16
Lena	33.0766	33.0704	33.0884	33.0817	33.0813
Peppers	33.0871	33.0728	33.0729	33.0726	33.0721
house	33.0732	33.0760	33.0760	33.0703	33.0675

Table 2.2. Watermarking results SSIM

Setting/Image	_Bitplanes_[5] _Blocksize_8	_Bitplanes_[5] _Blocksize_10	_Bitplanes_[5] _Blocksize_12	_Bitplanes_[5] _Blocksize_14	_Bitplanes_[5] _Blocksize_16
Lena	0.9402	0.9403	0.9404	0.9403	0.9404
Peppers	0.9452	0.9451	0.9451	0.9450	0.9451
house	0.9572	0.9574	0.9573	0.9572	0.9573

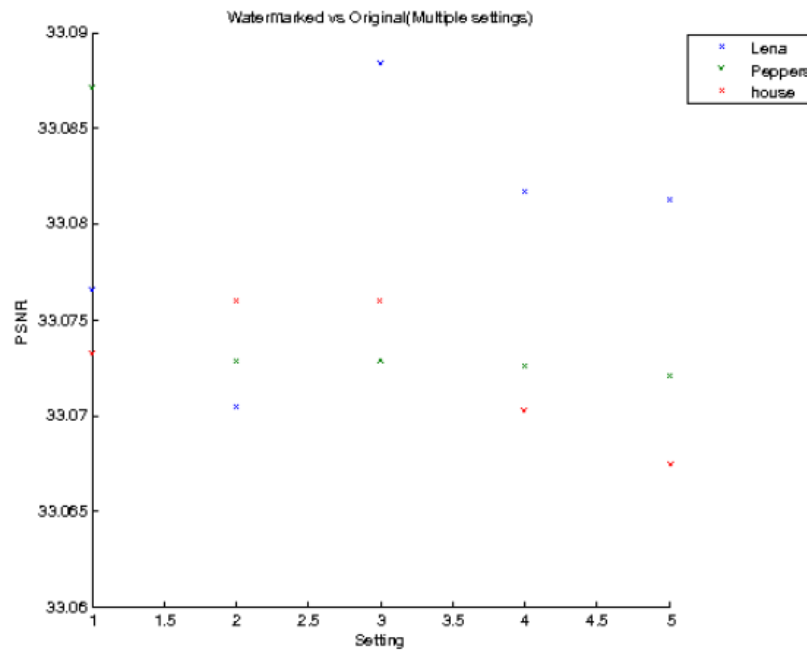


Figure 5. Automatically generated output report example.

REFERENCES

- [1] Cox, I. J., Miller, M. L., and Bloom, J. A., [*Digital watermarking*], Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2002).
- [2] Trappe, W., Wu, M., Wang, Z., and Liu, K. J. R., “Anti-collusion fingerprinting for multimedia,” *IEEE Trans. on Signal Processing* (2003).
- [3] Kerckhoffs, A., “La cryptographie militaire,” *Journal des sciences militaires* **IX**, 5–83 (January 1883).
- [4] Heylen, K., Meesters, T., Verstreppe, L., and Dams, T., “A matlab-based tutorial framework for digital watermarking,” in [*Proceedings of the Third European Conference on the Use of Modern Information and Communication Technologies (ECUMICT)*], (March 2008).
- [5] Ridzon, R., Levicky, D., and Klenovicova, Z., “Attacks on watermarks and adjusting psnr for watermarks application,” *Radioelektronika 2004* (2004).
- [6] Lambrecht, C. and Farrell, J., “Perceptual quality metric for digitally coded color images,” (1996).
- [7] Wang, Z., Bovik, A., Sheikh, H., and Simoncelli, E., “Image quality assessment: From error visibility to structural similarity,” *IEEE Transactions on Image Processing*, vol.13, no.4 (April, 2004).
- [8] Kutter, M. and Petitcolas, F. A. P., “A fair benchmark for image watermarking systems,” 226–239 (1999).
- [9] Cox, I., Kilian, J., Leighton, T., and Shamoon, T., “Secure spread spectrum watermarking for multimedia,” *IEEE Transactions on Image Processing* **6**(12), 1673–1687 (1997).
- [10] Zhu, W., Xiong, Z., and Zhang, Y. Q., “Multiresolution watermarking for images and video,” *IEEE Transactions on Circuits and Systems for Video Technology* **9**(4), 545–550 (1999).
- [11] Wang, Z., Bovik, A., Sheikh, H., and Simoncelli, E., “Image quality assessment: From error visibility to structural similarity,” (2004).
- [12] Chen and Wornell, “Quantization index modulation: A class of provably good methods for digital watermarking and information embedding (1 page),” *Proceedings of the International Symposium on Information Theory (ISIT)* (2000).
- [13] Costa, M., “Writing on dirty paper (corresp.),” *Information Theory, IEEE Transactions on* **29**(3), 439–441 (1983).
- [14] Moulin, P. and Koetter, R., “Data-hiding codes,” *Proceedings of the IEEE* (December 2005).
- [15] Petitcolas, F. A., Anderson, R. J., and Kuhn, M. G., “Attacks on copyright marking systems,” 218–238 (1998).
- [16] Meerwald, P. and Pereira, S., “Attacks, applications and evaluation of known watermarking algorithms with checkmark,” in [*Proceedings of SPIE, Electronic Imaging, Security and Watermarking of Multimedia Contents IV*], Wong, P. W. and Delp, E. J., eds. (January 2002).
- [17] Watson, A., Hu, J., and McGowan, J., “Dvq: A digital video quality metric based on human vision,” (2001).
- [18] Solachidis, V., Tefas, A., Nikolaidis, N., Tsekeridou, S., Nikolaidis, A., and Pitas, I., “A benchmarking protocol for watermarking methods,” *IEEE Int. Conf. on Image Processing, Thessaloniki, Greece* (2001).
- [19] Kim, H. C. and Delp, E. J., “A reliability engineering approach to digital watermark evaluation,” *Proceedings of the SPIE International Conference on Security and Watermarking of Multimedia Contents* (2006).
- [20] Voloshynovskiy, S., Pereira, S., Pun, T., Eggers, J., and Su, J., “Attacks on digital watermarks: Classification, estimation-based attacks and benchmarks,” *IEEE Communications Magazine (Special Issue on Digital watermarking for copyright protection: a communications perspective)* **39**(8), 118–127 (2001).
- [21] Certimark, “Common data processing and intentional attacks,” (2001).