Digital Image Watermarking Resistant to Translation, Rotation, and Scaling

Scott McCloskey

MS Project
Department of Computer Science
Rochester Institute of Technology

Committee: Stanislaw Radziszowski,
Roger Gaborski,
Edith Hemaspaandra

April 27, 2002

Abstract

Digital image watermarking has been touted for some time now as a solution to a wide range of problems from copyright enforcement to media bridging. A number of these application areas, however, require that the image's watermark be resistant to common image processing attacks such as scaling, rotation, and cropping. This is one area where current watermarking technologies often fall short of expectations, providing a significant barrier to their use in commercial systems. It was the purpose of this project to experiment with one proposed solution to this shortcoming, and attempt to make improvements upon that solution.

These experiments focused on three areas of potential improvement: more reliable extraction of the watermark, better robustness toward JPEG compression, and better visual masking to improve the appearance of the watermarked image. Significant improvements were made, though the algorithm's failure to handle JPEG compressed images precludes its use in a commercial setting.

1 Introduction

It's no news to anyone reading this that the meteoric rise of the Internet in recent years has significantly increased the availability of all forms of media. Nowhere is this more obvious than in the realm of commercial musical recordings, where Napster, Morpheus, Gnutella and other internet-based distribution systems have made incredible volumes of musical content widely available, all without the consent of the music's originators. This situation is not unique to the music industry, as photographers, filmmakers, and other content-originators all struggle with the same problem. The solution, and even the need for a solution, to this problem of copyright infringement is still a topic that is open to great debate, but there is no debating the fact that some interesting and novel technologies have been developed along the way.

Much of this novel technology has resulted from endeavors to embed messages or identifiers in digital still images, generally intended to allow photographers and visual artists to enforce copyrights covering their works. Digital image watermarking (and the related field of steganography) was briefly a matter of general public interest when, following the terrorist attacks of September 11, 2001, USA Today recirculated a report that Osama bin Laden's network of terrorists were likely passing secret messages over the Internet using, among other things, image watermarking [5]. Notwithstanding these nefarious applications, research continues to improve current watermarking technologies. While many different methods have been developed to embed and extract information from images, many of these methods fall apart in the presence of unanticipated alterations of the watermarked image.

It is the purpose of this project to experiment with one method, outlined by Kutter in [6] and [7], that attempts to overcome this weakness. In particular this method attempts to make the embedded watermark immune to affine transformations, which includes scaling, rotation, shearing, and translation. The ability of this algorithm to successfully extract the watermark from images attacked in this manner will be analyzed and compared to the performance of algorithms employed in commercial watermarking software.

We begin by discussing the history of watermarking and the related field of steganography. From there we discuss the different types of watermarking by focusing on both the intended applications as well as the underlying technology. This is followed by a detailed discussion of the theory behind the watermarking algorithm, including details of implementation that go unmentioned in [7]. The test methodology is presented, followed by the results and analysis of the algorithm's performance. Finally, we experiment with modifications to the algorithm intended to either improve performance or reduce the visual impact of the watermarking process.

2 Background

2.1 Steganography and Watermarking

Digital image watermarking is generally considered to be a derivative of Steganography, the ages-old practice of covert communication. Steganography is believed to have been first used in ancient Greece as recorded in the *Histories* of Herodotous. Indeed, the etymology of the word 'steganography' traces back to the Greek phrase whose literal translation is "covered writing" [4, page 2]. This is a rather apt description of the field's first use as employed by Histæus, then ruler of Miletus, who wanted to incite revolt in nearby Persia. In order to ensure it's safe passage, Histæus tattooed his inciteful message on the shaven head of a trusted slave. When the slave's hair grew, covering the message, he was able to pass into Persia without incident. Once the slave arrived at his destination his head was re-shaven and the message was revealed.

Further examples, both in fiction and in history, can be found in more modern times. In *Mother Night*, author Kurt Vonnegut writes about a fictional World War II spy, Howard W. Campbell, Jr., who passes information from operatives in Germany to intelligence agents in the United States by steganographic means. Campbell is the narrator of a propaganda program on a German radio broadcast, where he delivers regular denouncements of the United States for its actions against Germany. All the while, unbeknownst to the German government, Campbell broadcasts a sub-text of intelligence information by the placement of pauses, coughs, and mispronunciations in the overt message.

These two examples attempt to illustrate that, while steganography and watermarking both attempt to convey a secret message within an obvious one, they differ in a number of important dimensions. While messages passed by steganographic means are covert they are not necessarily secure in the cryptographic sense. Clearly, had Persian sentries been aware of Histaeus's method they might have shaved the head of all those entering their territory, thereby foiling the intended revolt. In the study of watermarks, on the other hand, researchers employ the cryptographic notion of Kerkhoff's Principle - the assumption that the adversary knows the method by which the watermark is embedded. Thus, in addition to the watermarking method employed, the adversary must also know a secret key, used in the embedding process, in order to extract the hidden message.

2.2 The Basics of Watermarking

All watermarking systems have, generically, two main steps: embedding and extraction. The embedding process takes as input a message (the data to be embedded in the image), a carrier signal (the digital image into which the watermark will be embedded), and a key, similar to the keys used in cryptographic systems. The output of the embedding process is a new digital image which contains the watermark.

The reverse process, extraction, is not the same for all watermarking systems.

This process, at a minimum, takes the watermarked image and the key as inputs. Depending on the specific method employed, the extraction may additionally take as input the original (unmarked) image and/or the message that is thought to be embedded in the image. The output of the extraction process varies, as well. Some systems extract the message and return it as an output, whereas others (most often those that take the watermark as an input to the extraction) will return a measure of confidence that the specified mark is found in the image. The type of extraction used is generally motivated by the intended application; several of these will be discussed later.

In addition to the generic embedding and extraction steps, watermarking systems have a number of characteristics that can be generically defined. The three most important of these characteristics are the watermark's robustness, payload, and level of perceptibility.

Robustness is simply the notion of the watermark's resistance to different types of attacks - any alteration of the watermarked image. Robustness is often achieved in watermarking systems by redundantly embedding the same information in different parts of the image. In general, a more robust watermark is preferred to one that is less so, but that is not always the case.

The payload of a watermark is the amount of information (measured typically in bits) that can be conveyed in a single image. Payloads of watermarks can vary from one bit of information to several bytes; commercial software packages generally have payloads in the range of 64 to 100 bits. Obviously, larger payloads are preferable in general, as longer watermarks make possible the embedding of more useful information such as an identification number, copyright statement, etc.

The perceptibility of the watermark is simply the degree to which a viewer of the watermarked image can notice the distortion introduced in the embedding process. As with robustness and payload, the perceptibility of the watermark varies based on its intended application; watermarks added to scenic imagery are best if they go unseen, whereas more obvious watermarks are desired for other applications. While not an example of a *digital* watermark, currency issued by the United States mint carry a number of different and very visible watermarks intended to prevent counterfeiting. It is worth noting that the perceptibility of the watermark is generally measured from the watermarked image only; different attacks, when applied to this image, may increase or decrease the mark's perceptibility.

These three characteristics of watermarks are in conflict to a certain degree. Increasing the payload of a watermark, for instance, is likely to reduce that watermark's robustness or increase the level of perceptibility of that watermark [4, page 109]. Consequentially tradeoffs are a normal part of designing or choosing a watermarking system for a particular application.

2.3 Applications for Watermarking Systems

2.3.1 Resolution of Ownership

The most frequently cited application for watermarking, the ability to resolve ownership disputes, is achieved by embedding a unique identifier in an image. In the event that the image is subsequentially used without consent, the presence of the watermark identifies the owner who may then demand payment for or prevent the use of the image.

Watermarks used to enforce ownership must be robust in the face of a number of different attacks. In order to deter unlicensed use, any processing that successfully removes the watermark should also destroy the value of the image in which it was originally embedded. Also, since the value of the carrier image is being protected, watermarks intended for this purpose should be imperceptible to a human viewer under reasonable conditions.

2.3.2 Image Authentication

A completely different application for watermarks, driven primarily by the insurance and legal industries, is the certification of an image's authenticity. Given an image in connection with an insurance claim, for instance, it is necessary for the adjuster to ensure that it has not been altered to exaggerate losses (and thus inflate the claim).

Authentication is generally achieved by the use of a very fragile watermark - one that is destroyed whenever the image is altered in any way. Digital cameras produced by a trusted party could automatically embed such a watermark into each image in order to assure that no digital effects have altered the appearance of the image. In order to assure the authenticity of the image, then, one must simply observe the presence of the watermark in that image.

Such watermarks have some unique requirements. In addition to ensuring the its destruction by any type of image processing, the watermark must not be able to be forged. If it were possible to interrogate several authentic images in such a way as to obtain a copy of the watermark, then that copy could be embedded in any other image - authenticating an image regardless of its actual content!

Another interesting observation regarding such marks is that they require a payload of only one bit. It doesn't matter what form the watermark takes, as the only interest is in whether or not the mark is present in the image.

2.3.3 Copy Control

Another novel use of watermarking technology has been suggested that would allow a photographer to determine the origin of unauthorized copies of a particular image. If the photographer licensed the use of a photograph to a small number of individuals it would be possible to send these individuals different versions of the image, each with a unique watermark. Whenever unauthorized

use of the image is found, the watermark could be extracted in order to determine which of the authorized users had illegally distributed the image to others.

While such watermarks are similar to those intended to resolve ownership (in that both require the mark to be robust and imperceptible) this application opens a new avenue of attack due to the fact that the same image is distributed with different watermarks. These so-called collusion attacks will be covered in greater detail later.

3 Kutter's Algorithm

In [6], Kutter et al. describe a basic watermarking system (which we will refer to as the baseline system) for images intended to be used for the enforcement of copyrights. In this paper, the authors present the embedding and extraction steps, and show results that indicate the robustness of the extraction method to signal processing attacks such as blurring, sharpening, or JPEG compression. Also included in this paper is an outline of a method to make the extraction process more robust toward affine geometric attacks; a specific example is included indicating successful extraction in spite of rotation.

Several months later, Kutter published a second paper [7] that outlined a more sophisticated method (the enhanced method) by which the original algorithm could be extended for improved robustness toward affine geometric attacks. In addition to a description of modified embedding and extraction steps, the bulk of the paper is spent describing a method by which a watermarked image, attacked by some affine transformation, could be corrected - that is, transformed in such a way as to undo the effects of the attack.

Both the baseline and enhanced methods are spatial watermarking methods, and both operate on RGB imagery. That is, the image is given as three arrays or channels containing the red, green, and blue intensities at each pixel location. In order to reduce the visual offensiveness of the watermark, embedding is performed on the image's blue channel where changes are less apparent due to the characteristics of the human visual system. Both take as input a binary watermark, the image into which is to be embedded, a secret key, and a set of parameters.

In the following sections we will describe these methods in the order that they were introduced. In some places detail will be added where the original description was incomplete.

Before proceeding, we must address a matter of notation. Two different coordinate systems are used in the following sections. When referring to the color planes of an image, we will use a convention where position (i, j) is the ith row counting down from the top and the jth column counting right from the left edge. When referring to positions of peaks within the autocorrelation function, the position (x, y) will refer to the Cartesian coordinates in the XY plane.

3.1 Baseline Algorithm

3.1.1 Baseline Embedding

Before the embedding takes place, the binary message is altered in two ways. First, the sequence 01 is appended to the beginning of the message in order to ensure the presence of a known pattern which will be used during the extraction. Secondly, the message is polarized, replacing all 0's with -1's, including the leading 0 (the same effect could be achieved by changing all 1's to -1's and then all 0's to 1's). Also, the embedding key is used to seed a pseudo-random number generator that will be used to determine the embedding locations.

The first embedding parameter is the density parameter, a value between 0 and 1 which indicates the probability that any given pixel in the blue channel will have part of the message embedded there. A density value of 0 means that no pixels will contain message data (and thus the message won't actually be embedded) whereas a value of 1 means that all pixels in the blue channel will contain some part of the message.

Initially the watermark, an array of the same size as the blue channel, contains all zeros. The first step of the embedding process is to determine which of the pixels in the blue channel will receive part of the message. For each pixel location (i, j), a pseudo-random number is generated in the range $[0, 1)$ and compared to the density parameter. If the random number is less than the density parameter, that pixel is designated to carry part of the message payload. For each pixel that is designated to carry message data a second random number is generated, the value of which determines which bit of the message it will carry. Since the number of pixel locations in the image is large, and because the density parameter is non-zero, each bit will be embedded redundantly throughout the image. The value of the watermark at position (i, j) is changed to include information about the nth bit of the message.

$$mark_{i,j} = b_n \kappa$$

where b_n is the bit to be embedded at this location, and κ is an overall strength parameter. Once this process has been repeated for all positions (i, j), the value of $mark$ is the complete, unscaled watermark.

The mark is then embedded in the image by adding a scaled version of the watermark to the blue channel,

$$blue_{i,j} = blue_{i,j} + \alpha_{i,j} mark_{i,j}$$

where $\alpha_{i,j}$ is a local strength value that attempts to minimize the visibility of the mark at that position.

In his original paper, Kutter suggests using the pixel's luminance (the overall brightness, taking into account the responses of the color-sensing cones in the human eye) as the visual mask α.

3.1.2 Baseline Extraction

The first step in the watermark extraction process is to filter the watermarked image's blue channel with a kernel k. The purpose of this filtering step is to remove all of the scenic content of the blue channel, leaving only the changes introduced during embedding, $\hat{w}$.

$$\hat{w} = blue * k$$

Ideally, the filtered blue channel would have a value of 0 at all locations except those where parts of the message were embedded, where the value would be equal to $b_n \kappa \alpha_{i,j}$. While this isn't possible to do for all images, a filter is selected that will have a good performance on most images. Errors due to sub-optimal filter selection are overcome by the redundancy of the embedded watermark. In the original paper, the authors suggest the filter

$$k = \begin{bmatrix} 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & -1 & -1 & 12 & -1 & -1 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{bmatrix}$$

Once the blue channel has been filtered in this way, the extraction continues by seeding the pseudo-random number generator with the known embedding key. An accumulator array (initially containing all zeros) is created with an entry for each bit of the embedded message (including the leading 01).

We then traverse the image, generating a pseudo-random number for each pixel location (i, j). As in the embedding step, this number is compared to the density parameter to determine whether or not it contains some part of the message. If a pixel is determined to contain part of the message a second number, n, is generated in order to identify which bit was embedded there. Note the implication that the length of the binary message is known at the time of extraction. Once a pixel is identified to contain part of the nth bit of the watermark, the accumulator is updated

$$acc_n = acc_n + \hat{w}_{i,j}$$

Once the entire image has been traversed in this manner, the last step is to map the accumulated values to 0s and 1s, hopefully recovering the same message that was embedded. Since each pixel carrying part of the nth bit was offset in the same direction (either positive or negative) the simplest way to do this would be to map positive values in the accumulator array to 1 and negative values to 0. While this works in many instances, it is susceptible to error when the image has been processed (e.g. JPEG compressed). In order to remove any bias, an adaptive threshold is generated from the first two bits of the watermark, known

to be 0 1, by averaging the first and second entries in the accumulator array.

$$thresh = \frac{acc_1 + acc_2}{2}$$

Each subsequent accumulator value is then compared to this threshold, giving the extracted message; those values which are lower than the threshold are mapped to 0 while those above the threshold are mapped to 1.

In addition to the extracted message, the extraction process yields a second output: a confidence measure. This confidence measure is equal to acc_1 - acc_0.

3.1.3 Robustness of the Baseline Watermark

In the original paper, a scheme is outlined to successfully extract the message even in the case of a geometric attack. The basis of this method is that the confidence value can be used to determine when the message has successfully been extracted. If the image is known to have been rotated since the time that the watermark was embedded, for instance, the attacked image could be rotated by arbitrary angles, performing extraction on each of the resultant images. When done in an exhaustive manner, one needs only keep track of the extracted message and the confidence corresponding to the angle of rotation. The correct angle (which would be the negative of the angle through which the original image was rotated) could be identified by its high confidence measure, which would be the global maximum.

Similarly, other affine geometric attacks could be countered by performing an exhaustive search of whatever type of operation had attacked it. In every case, the confidence measure identifies which of the extracted message is the correct one. Obviously, a visual inspection of the original and attacked images can help estimate the extent of the attack and thus narrow the search.

3.1.4 Analysis of the Baseline Algorithm

From a use case perspective, this method is rather nice. The only pieces of information that must be known in order to extract the message are the embedding key, the message's length, and the density parameter. In contrast to other schema, the original (unmarked) image is not required during this step. However, were this the case, we see that detector performance would be improved; instead of filtering the image's blue channel, we could simply subtract from it the values in the blue channel of the original image.

At first glance, the reader may be suspicious of just how robust this watermark is in practice. With respect to image processing attacks, the robustness of this method is based in the fact that the same bit of the message is embedded in multiple locations. The embedding process can be thought of as adding noise to the blue channel of the image which, while less visible to the human eye, is problematic for a number of reasons. Like most still image compression methods, the JPEG compression standard significantly alters high frequency features like those introduced during embedding. This, more than anything, illustrates

something of a Catch 22 that is inherent in robust watermarks: We attempt to make these marks invisible to the human eye with the realization that image compression methods are likely to remove them precisely because the viewer wouldn't notice them.

With respect to geometric attacks, the analysis is no more kind. The method presented to cope with these types of attacks is inefficient at best. While a visual inspection can help to narrow the extent of searching, this depends on the ability of the observer to notice potentially small changes in the image's orientation, size, etc. Moreover, the use of compound attacks (such as a rotation followed by a scaling) causes the search space to rapidly increase in size. While this method may suffice for simple geometric attacks, it can hardly be considered a solution in the general case.

Further exacerbating the watermark's weakness to geometric attacks is the fact that it is applied in the spatial domain. Because of the fact that the pixels which contain message data are chosen by a pseudo-random number generator, the extraction is destined to fail in the case of image misregistration. If, for instance, the first row of pixels were removed from the image then all of the embedding locations would be off by one row, causing extraction to fail miserably. Were the original size of the image known at the time of extraction, it would be possible to resize or pad the image in order to assure that the dimensions match and that the random number generation is performed correctly.

With this analysis in mind, we consider the revised watermarking scheme.

3.2 The Robust Watermark

The enhanced embedding method, as outlined in [7], derives its increased robustness by embedding the same watermark at four different positions in the same image. The relative positions of the watermark in the attacked image, when compared to the original positions at the time of embedding, reflect the type of attack that has been performed on the image. Once the attack has been identified, the inverse attack can be applied to generate a fixed image from which the watermark can be extracted.

This method is only intended to recover the watermarks from images that have been attacked with affine geometric transformations described by a 2-by-2 matrix. A pixel at location (i, j) in the original image is transformed to location (i', j') by the following relationship:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} i' \\ j' \end{bmatrix}$$

where the values of a, b, c, and d indicate the type (i.e. rotation, scaling, etc.) and extent (angle, factor, etc.) of the transformation. In order to determine the inverse transform, it is necessary to locate two sets of corresponding positions before and after the attack. Once these corresponding positions

$$(i_1, j_1) \rightarrow (i'_1, j'_1), \qquad (i_2, j_2) \rightarrow (i'_2, j'_2),$$

are identified, the inverse transform is found as follows

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \begin{bmatrix} i_1 & i_2 \\ j_1 & j_2 \end{bmatrix} \begin{bmatrix} i'_1 & i'_2 \\ j'_1 & j'_2 \end{bmatrix}^{-1}$$

Note that the solution in [7] is incorrect, as the terms in the matrix product are reversed.

3.2.1 Robust Embedding

In order to accommodate the multiple embedding of the watermark without overlapping (which would cause interference), we must substantially change the embedding process. While any group of non-overlapping areas would suffice, the author suggests dividing the image into four different sets, the first of which is composed of only those pixels in even rows and even columns. The second image contains pixels from even rows and odd columns, the third odd rows and even columns, while the fourth set contains those pixels from odd rows and odd columns. These four sets are obviously non-overlapping in the original and thus will not give rise to interference.

The same watermark will be embedded in each of these four sets of pixels at different locations. These locations are determined by two shift parameters: δ_x and δ_y, both of which must be odd. An unmasked watermark of size u-by-v, where

$$u = \lfloor \frac{numRows}{2} \rfloor - \delta_y, \qquad v = \lfloor \frac{numCols}{2} \rfloor - \delta_x$$

is generated as before, and then is expanded by adding a row of zeros between every pair of adjacent rows, and a column of zeros between every pair of adjacent columns. This assures that the non-zero values of the mark occur only in odd rows and odd columns.

The watermark is then embedded four times in the blue channel, as follows:

$$blue_{i,j} = blue_{i,j} + \alpha_{i,j} mark_{i,j},$$

$$blue_{i+\delta_y,j} = blue_{i+\delta_y,j} + \alpha_{i+\delta_y,j} mark_{i,j},$$

$$blue_{i,j+\delta_x} = blue_{i,j+\delta_x} + \alpha_{i,j+\delta_x} mark_{i,j},$$

$$blue_{i+\delta_y,j+\delta_x} = blue_{i+\delta_y,j+\delta_x} + \alpha_{i+\delta_y,j+\delta_x} mark_{i,j}$$

Note that the dimensions of $mark$ are actually smaller than $blue$ by at least $\delta_y + 1$ rows and $\delta_x + 1$ columns. Because the value of the watermark is zero in every even row and column and because the shift parameters are odd, the watermark will not interfere with itself. Also, since δ_x and δ_y are odd, the four embeddings add the watermark to the four sets of pixels defined above.

Figure 1: Watermarked *baboon* image and corresponding autocorrelation function

3.2.2 Fixing an Attacked Image

In order to fix an attacked image, it is first necessary to determine the form of the transformation by analyzing the attacked image's filtered blue channel, $\hat{w}'$. Considering only geometric transformations, $\hat{w}'$ is simply the transformed value of $\hat{w}$, the watermarked image's filtered blue channel. We determine the type of the transformation by observing the relative positions of the watermark in the attacked image using the autocorrelation function, defined as

$$R(\hat{w}')_{x,y} = \sum_u \sum_v \hat{w}'_{u,v} \hat{w}'_{x+u,y+v}.$$

Note that the autocorrelation function is symmetric, has (potentially) non-zero values at (x, y) where $-numRows < y < numRows$ and $-numCols < x < numCols$, and has a global maximum at $(0, 0)$. This function provides a measure of self-similarity within the attacked image's filtered blue channel.

In the case that the watermarked image hasn't been transformed, the autocorrelation of the filtered blue channel will have 9 peaks at (x, y) locations

$$(0, 0), \text{largest peak}$$

$$(\pm\delta_x, 0), (0, \pm\delta_y), \text{large peaks}$$

$$(\pm\delta_x, \pm\delta_y), \text{small peaks.}$$

An example of this is shown in Figure 1, where $\delta_x = 23$, and $\delta_y = 43$. Because they are more easily identified, and because only two (non-symmetric) points are required to find the inverse of the attack matrix, we choose to focus on the peaks at locations $(\delta_x, 0)$ and $(0, \delta_y)$, which are shown in green.

In the case that the watermarked image has been attacked, the peaks of the autocorrelation function will reflect the transformed locations of these peaks. See, for example, figure 2. The carrier image has been rotated by 45°, as have the peaks of the autocorrelation function. In order to determine the parameters

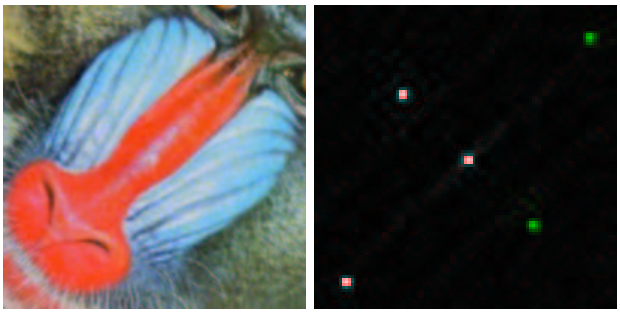

Figure 2: An attacked image and the corresponding autocorrelation function

of the attack, then, we extract the 5 largest peaks, one of which must be the center. For each pair of these peaks that don't form a line through the center peak, we determine the geometric transformation that would map it back to our two original positions. Since we can choose four such pairs of peaks, and because each pair could map in two different ways back to the original peak positions, we end up with 8 different transformations.

In order to extract the watermark in spite of the attack, we try each of the 8 transformations. Each transformation yields a different $\hat{w}$ (one of which, which we hope, is approximately equal to the filtered blue channel of the watermarked image), from which we attempt to extract a message. For each of these we keep both the extracted message and the confidence measure. Once all eight transformations have been attempted, we find the one that resulted in the highest confidence measure during extraction. The message corresponding to this confidence measure is given as the overall extracted message.

While most of these operations are well defined, the process of extracting the peaks from the autocorrelation is less so. In [7], Kutter suggests that

> Peak detection is implemented by first computing the gradient of the autocorrelation function. The locations of gradient maxima are then used in the autocorrelation function to define square regions in which the maxima correspond to peaks. [7, page 428]

While this gives a basic idea of how to implement this, it omits several details such as the size of the square regions, what type of gradient, etc. Moreover, this gives no indication of how to find peaks that do not occur at whole numbered coordinates within the autocorrelation function. Such a method is necessary in order to get good detector performance as slightly different transformation matrices will result in images of different sizes from which the message may be difficult to extract (as explained previously).

3.2.3 Extraction from a Fixed Blue Channel

The extraction from a fixed image is a fairly straightforward extension of the baseline extraction method. First, the fixed blue channel is separated into four separate images, each containing pixels from a single embedding of the watermark. In Matlab syntax, these images are:

$$image_1 = blue([1, 3, 5...2u-1], [1, 3, 5...2v-1])$$

$$image_2 = blue([1, 3, 5...2u-1], [\delta_x + 1, \delta_x + 3...\delta_x + 2v - 1])$$

$$image_3 = blue([\delta_y + 1, \delta_y + 3...\delta_y + 2u - 1], [1, 3, 5...2v-1])$$

$$image_4 = blue([\delta_y + 1, \delta_y + 3...\delta_y + 2u - 1], [\delta_x + 1, \delta_x + 3...\delta_x + 2v - 1])$$

where u and v are the size of a single embedded watermark. Basically, image 1 is composed of the first u-by-v pixels in the blue channel that appear at odd row and column coordinates. Image two is composed of the first u-by-v pixels that appear in odd rows and even columns, starting with location $(1, \delta_x + 1)$. Image three is composed of the first u-by-v pixels in even rows and odd columns, starting with $(1+\delta_y, 1)$. Finally, the fourth image is composed of the first u-by-v pixels in even rows and even columns, starting with $(1 + \delta_y, 1 + \delta_x)$.

From these four images, we extract the message as before. The accumulator vectors from each are added together to give an overall accumulator vector. We use the leading 01, as before, to generate an adaptive threshold against which all of the other entries are compared.

3.2.4 Analysis of the Robust Algorithm

We see that, by embedding the same simple watermark into an image four times, we can identify an affine transformation that has been applied to our watermarked image. Having done this the image can be fixed by applying the inverse transformation, thus allowing the extraction as if no attack had taken place. Due to the symmetries of the autocorrelation function, however, we must attempt 8 different extractions to assure that we can also handle images that have been flipped in the X or Y direction.

It is worth noting that, by using the autocorrelation function to fix an attacked image, we are introducing a second robustness question. Not only do we need to be concerned with the robustness of the watermark, but we must also pay attention to the robustness of the autocorrelation function. Since the autocorrelation function is performed on the filtered blue channel, which contains mostly high frequency information, we must be particularly concerned with how the autocorrelation function survives attacks such as low-pass filtering and lossy compression.

It is also not apparent how this method, as suggested in the title *Watermarking Resisting to Translation, Rotation, and Scaling* [7], is actually robust to translation. The author makes no direct reference to translation outside of the title, and the method suggested for fixing an attacked image does not account for translation, as it can't be expressed as a 2-by-2 matrix transformation.

Moreover, the peaks of the autocorrelation function do not change as a result of translation, since it doesn't change the relative positions of the individual marks. More attention will be paid to this subject in the experiments section.

3.3 Implementation Details

Peak detection was implemented in the test code following the general guidelines presented in [7]. The gradients of the autocorrelation function were computed in both the x and y directions, and their absolute values summed. A processed map was computed with the value at each position (x, y) equal to the sum of these values at surrounding positions.

$$map'_{x,y} = |grad_{x-1,y}| + |grad_{x+1,y}| + |grad_{x,y-1}| + |grad_{x,y+1}|.$$

where $grad$ is the sum of the absolute values of the horizontal and vertical gradient functions.

Peaks are identified iteratively by locating the largest value in map' and zeroing the values at those locations nearest it. The area to be zeroed is at least an 11-by-11 sided square centered on the peak. The length of these sides is expanded until the largest value on its perimeter is no longer the largest value in the map outside of the region. Once this condition is satisfied the area is zeroed and the process is iterated until the five peak locations have been identified.

In order for the extraction to have any hope of correctly finding the embedded watermark, it is necessary that the fixed images be of the same size as the original watermarked image. In order to achieve this, it is necessary to include an additional input to the extraction process: the original size of the watermarked image. In order to assure that the fixed images are of the same size as the original, we frame the image before extraction. If the fixed image has n fewer rows/columns than the original image, we add $\lceil \frac{n}{2} \rceil$ blank rows/columns to the top/left of the image and $\lfloor \frac{n}{2} \rfloor$ blank rows/columns to the bottom/right of the image. Similarly, if the fixed image has n too many rows or columns, we remove the outermost rows and columns equally from the perimeter of the image.

4 Evaluation Methodology

For the purposes of this project, Kutter's watermarking method was subjected to a number of tests in order to assess both its robustness and the degree to which observers would find the presence of the mark objectionable. Both of these evaluations were performed over a set of six images, selected from the image library accompanying StirMark [12]: *lena*, *baboon*, *bear*, *fishingboat*, *skyline arch*, and *watch*. These images (shown in appendix A), which span a considerable portion of image space, are the same used for the evaluation of commercially available watermarking packages.

4.1 Robustness Testing

When testing for robustness, we determine the detection rate - the number of successful extractions as compared to the number attempted - over a set of attacked images. In order to get results that allow comparison with other methods, we use the standard set of attacks as implemented in StirMark [11, 9, 10]. StirMark is a DOS-based program that, when given a watermarked image, creates a set of approximately 160 attacked versions of that image. These attacks fall into the general category of "unintentional" attacks - those that might be performed by a user of the image without specific intent to remove the watermark. Intentional attacks, on the other hand, are those performed maliciously by users who know that a watermark is present and, moreover, the method with which they are embedded.

All of the results listed below were found by embedding and attempting the extraction of a 72 bit (including the two bits appended before embedding) message from images that were approximately 200-by-200 pixels in size.

4.1.1 Unintentional Attacks

Affine Geometric Distortions This class of attacks, for which Kutter's algorithm was specifically designed, includes anything that can be expressed as a 3-by-3 matrix transformation applied to coordinates in a homogeneous coordinate system. This includes rotation, translation, scaling, shearing, and flipping (reflection). Attacked images in this category are generated by StirMark - for a complete listing, refer to [11].

Signal Processing Alterations This class of attacks, which typically alter the frequency content of an image, includes such modifications as blurring, sharpening, and JPEG compression. A reasonable sampling of these attacks is also provided by StirMark.

Color Alteration Color alterations are included in StirMark attacks in two different forms. The first attack is a color reduction attack, where RGB code values are moved closer to a gray color at the same brightness (luminance). The second is implicitly performed as a part of JPEG compression, which includes an RGB $\rightarrow YC_rC_b$ (luminance-chrominance) rotation followed by a sub-sampling of the C_b and C_r components.

Content Removal Attacks These types of attacks are those that endanger the watermark by removing parts of the image that carry its content. StirMark implements two attacks that fit in this category: cropping and random row and column removal.

Digital to Analog Conversions Attacks in this category consist of those that take the watermarked digital image and convert it to an analog signal, typically for display on a monitor or a hard copy print. In [7], Kutter claims to have successfully removed his watermark from an image that had been printed and then scanned. Since this attack (or anything like it) is not

implemented in StirMark, this test was carried out separately in order to verify this claim.

4.1.2 Malicious Attacks

Averaging and Naive Removal (Collusion) This type of attack is generally performed when the opposition knows that an image is watermarked, but doesn't know the method with which the message is embedded. The notion of this attack is that the two or more versions of the same image, each containing a different message, are averaged to produce a new, presumably watermark-free, image. It is expected that this method will not be successful due to the redundant embedding of each bit of the message.

Informed Removal This type of attack takes into account Kerkhoff's Principle, where we assume that the opposition knows the watermark embedding and extraction methods but not the key. Such an attack tailored for Kutter's algorithm would be one that replaces each pixel value in the blue channel by the average of the four pixels that it borders. Because the filtering step effectively measures the difference between a pixel's value and its expected value based on its neighbors, this attack should significantly reduce the meaningful content in the filtered blue channel. An unanswered question regarding this type of attack, however, is whether or not such a removal would damage the appearance of the carrier image.

Multiple Watermarking This question applies, in the context of copyright enforcement, to the case where an unauthorized user applies a watermark to an image that already carries the mark of its rightful owner. Since there is no notion of order in this system there is no reason to believe that Kutter's watermark could properly resolve which watermark had been embedded first. It is unknown, however, whether the original mark could be recovered at all, and even whether the second mark could be successfully recovered.

Forgery While the application in the context of copyright enforcement isn't particularly compelling, it is interesting to examine the question of whether or not someone could embed another person's message in an image without knowing the embedding key. If the image is of the same size as two images that have been previously embedded with the same key and message, it would be possible to do this by finding those pixel locations where the filtered blue channels of each image have large values of the same sign. Using these locations, we could perform an embedding similar to the baseline method and could effectively embed that message in a third image without knowing the key. Such a mark would have a lower confidence upon extraction, but would result in the desired detection.

4.2 Visual Perceptibility

In order to determine how noticeable the watermark is in a particular image, we use two different measures. The first, peak signal to noise ratio, is a numeric measure defined as

$$PSNR = 10log_{10}\frac{N^2 max_{i=1}^{N} X_i}{\sum_{i=1}^{N}(X_i - X_i')^2}$$

where X_i and X_i' are the ith pixel values from the original and watermarked images, respectively, and N is the number of pixels in the image. Larger PSNR values are desirable, as they indicate that little change has been introduced to the image.

This measure, while it gives an estimate, does not accurately predict the degree to which a viewer will detect the presence of the watermark. In order to determine this, we use a panel of 10 judges to compare an original image to its watermarked descendant. Each judge assigns a value from 0 to 4, with 0 indicating that the mark isn't detectable and a 4 indicating that the presence of the watermark is very objectionable. These results are averaged over the 10 judges, giving one measurement for each image.

5 Evaluation of Kutter's Algorithm

In order to verify the claims made in [7], a Matlab implementation of the algorithm (as described in section 3.2) was tested against images produced by StirMark. The parameters used for this testing were based on the suggested values in [7] and were expanded to include a range that produced images of reasonable quality while giving noticeable changes in the detection rate.

All measurements given in sections 5 and 6 were made during the course of this project. No results were given in [7] with which to compare these numbers.

As a point of reference, the Matlab embedding code executed for approximately 0.6 seconds on a 200-by-200 pixel image. Extraction of the message from this same image required 6.5 seconds of processing time, much of which was spent calculating the autocorrelation function. These numbers were found on a system using a 400MHz UltraSPARC II processor. It should be noted that Matlab implementations are significantly slower than a similarly optimized implementation in a compiled language such as C or C++.

5.1 Results

5.1.1 Watermark Perceptibility

The results in table 1 are in line with expectations. PSNR decreases with increases in either the watermark's strength or the embedding density parameters. Similarly, the subjective rating increases (meaning that the visibility of the watermark increases) as strength or density increase.

	baboon	*bear*	*fishingboat*	*lena*	*skyline arch*	*watch*
PSNR, density = 0.50						
κ=0.15	44.47	47.61	45.55	45.54	48.61	50.63
κ=0.20	42.13	45.11	43.07	43.04	46.28	48.11
κ=0.25	40.35	43.19	41.19	41.10	44.47	46.17
PSNR, κ = 0.20						
density=0.40	43.09	46.02	44.06	44.00	47.02	49.16
density=0.50	42.13	45.11	43.07	43.04	46.28	48.11
density=0.60	41.28	44.31	42.22	42.17	45.54	47.25
Subjective Rating, density = 0.50						
κ=0.15	1.1	0.8	1.3	1.5	1.1	0.9
κ=0.20	1.3	1.1	2.2	2.2	2.1	1.6
κ=0.25	1.9	1.8	3.0	3.0	2.7	1.8
Subjective Rating, κ = 0.20						
density=0.40	1.3	0.9	1.9	2.2	1.4	1.0
density=0.50	1.3	1.1	2.2	2.2	2.1	1.6
density=0.60	1.4	1.5	2.7	2.6	2.1	1.7

Table 1: Watermark perceptibility of Kutter's algorithm.

5.1.2 Robustness toward Unintentional Attacks

The robustness results, however, are somewhat disappointing. Table 2 summarizes the decode rates for these images, separated by whether or not the attack included a compression step. The results indicate that this watermark isn't particularly robust to attacks that include JPEG compression. Most of the compressed images from which the watermark was successfully removed turn out to be those that were subject to signal processing attacks, and not the geometric transformations. This indicates that the geometrically attacked images are not being fixed properly, most often due to an inability to properly locate the desired peaks in the autocorrelation function. This implies that, while the watermark itself may be robust to JPEG compression, the peaks of the autocorrelation are not.

It should come as no surprise that the detection rate generally increases when the watermark strength is increased. For some of the images we see a diminishing return for increased watermark strength; it seems that some of these attacked images can't be successfully decoded no matter the strength. Analysis of some of these images indicates that these failed detections are due to a misframed image, which we will attempt to rectify in a later experiment.

We see that increasing the watermark density has less of an effect than does increasing its strength. It seems to have a more marked effect on the JPEG compressed images, perhaps because the added redundancy is necessary in order to counter the removal of watermark data.

In an attempt to duplicate the results in [7], a copy of the *lena* image was

	baboon	*bear*	*fishingboat*	*lena*	*skyline arch*	*watch*
uncompressed, density = 0.50						
κ=0.15	88%	58%	70%	76%	37%	21%
κ=0.20	79%	74%	79%	87%	74%	47%
κ=0.25	86%	72%	89%	92%	80%	66%
compressed, density = 0.50						
κ=0.15	0%	1%	1%	1%	1%	0%
κ=0.20	7%	3%	1%	2%	1%	0%
κ=0.25	17%	7%	7%	8%	2%	0%
uncompressed, κ = 0.20						
density=0.40	84%	70%	80%	89%	57%	37%
density=0.50	79%	74%	79%	87%	74%	47%
density=0.60	88%	72%	93%	92%	76%	60%
compressed, κ = 0.20						
density=0.40	2%	5%	1%	1%	1%	0%
density=0.50	7%	3%	1%	2%	1%	0%
density=0.60	6%	7%	2%	8%	1%	0%

Table 2: Decode rates for the proposed algorithm.

watermarked with *density* = 0.50 and κ=0.20. That image was printed on an HP laser printer and scanned at 600 dpi in two different ways: in the original orientation and rotated by approximately 45° in the clockwise direction. Both images were passed through the watermark extraction software and neither resulted in a successful detection. The image in its original orientation was properly handled until the framing step, which misaligned the image relative to its original position, resulting in a failed detection. The rotated image failed due to an inability to detect the peaks of the autocorrelation function (shown in figure 3), which had a number of features that could easily be mistaken for peaks.

5.1.3 Robustness toward Malicious Attacks

All of the following results were obtained using the enhanced watermarking method (as described in section 3.2) with the *density* = 0.50 and κ=0.20.

Averaging and Naive Removal This attack proved remarkably ineffective for most of the images. The results in table 3 show the number of messages that were successfully extracted from an image whose blue channel was the average of 1, 2, 3, 4, or 5 differently marked blue channels. With the exception of the *watch* image, all of the others could successfully detect each of three watermarks in an image whose blue channel was the average of the three marked blue channels. With the *watch* image only one watermark could successfully be extracted when as many as three watermarked blue channels were averaged, but none could be detected in

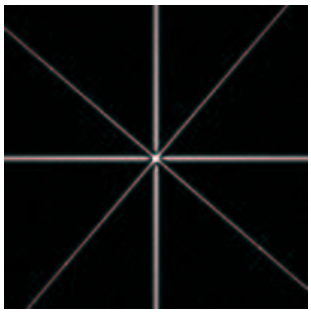

Figure 3: Autocorrelation of the rotated print-scan test image

	baboon	*bear*	*fishingboat*	*lena*	*skyline arch*	*watch*
1 mark embedded	1	1	1	1	1	1
2 marks embedded	2	2	2	2	2	1
3 marks embedded	3	3	3	3	3	1
4 marks embedded	4	4	3	3	4	0
5 marks embedded	3	2	3	3	5	0

Table 3: Results of the naive removal attack. Each entry represents the number of unique watermarks that were extracted from an image whose blue channel is the average of differently marked blues.

the case of 4 or more. Half of the images could give a successful extraction for 4 watermarks, and one (*skyline arch*) allowed the extraction of all 5 messages from a blue channel that was the average of 5 differently marked blues.

Informed Removal This method proved much more effective. In all but one case (the *baboon* picture) the watermark was damaged enough to prevent successful extraction. Moreover, the change in the image's appearance was minimal, as illustrated in table 4. The $PSNR$ value of these images was generally better than that of the watermarked originals because it removed or dampened the noise introduced during embedding. The judges did not evaluate these images, as an initial assessment by the author determined that the changes were below the visual threshold.

Multiple Watermarking As in the case of naive removal, the watermark proved to be remarkably robust with respect to multiple watermarking. Table 5 shows the number of different messages that were successfully extracted from an image that had 1, 2, 3, 4, or 5 marks embedded. With five of the six test images it was possible to embed and successfully extract as many as 5 different watermarks with different keys. For the remaining

	baboon	*bear*	*fishingboat*	*lena*	*skyline arch*	*watch*
bit errors (of 70)	0	67	64	2	58	58
PSNR (marked)	42.13	45.11	43.07	43.04	46.28	48.11
PSNR (attacked)	42.36	46.73	44.68	45.83	47.71	46.38

Table 4: Results of the informed removal attack. Shows the number of bits of the watermark which were corrupted, and the change in PSNR caused by the attack.

	baboon	*bear*	*fishingboat*	*lena*	*skyline arch*	*watch*
1 mark embedded	1	1	1	1	1	1
2 marks embedded	2	2	2	2	2	2
3 marks embedded	3	3	3	3	3	3
4 marks embedded	4	4	4	4	3	3
5 marks embedded	5	5	5	5	5	4

Table 5: Results of the multiple watermark attack. Each entry represents the number of unique watermarks that can be extracted from an image with multiple messages.

image (*watch*), it was possible to extract four of the 5 embedded watermarks.

Forgery The method outlined above successfully allowed the embedding of a watermark in an image by using two images, of the same size, that bore the same mark. Using marked versions of the *lena* and *baboon* images, the watermark was successfully embedded in the *fishingboat* image without knowledge of the embedding key. The mark was extracted from the *fishingboat* image with a lower confidence than either of the two originals, but was successful.

5.2 Analysis

The results from these tests indicate a number of areas for potential improvement:

- The robustness against attacks that include JPEG compression must be addressed. The algorithm is not able to undo geometric attacks because of its inability to locate the correct peaks within the autocorrelation function. While this might be solved in some cases by the use of a better detection scheme, a more general solution is desirable that would make these peaks more readily identifiable.
- The inability to properly frame a fixed image significantly reduces the detection rate. A more sophisticated framing method is desired in order

to increase the detection rate in attacked images which are uncompressed.

- The visual masking method suggested by Kutter seems sub-optimal. Images, particularly *skyline arch* and *baboon*, with large blue areas expose the watermark at lower embedding strengths than those without such areas.
- This algorithm is not secure with respect to Kerkhoff's Principle. Malicious attackers who know the embedding method can easily remove an image's watermark with little impact on the carrier image.

6 Experiments with Further Enhancements

6.1 A Revised Encoding for Improved Robustness to JPEG Compression

6.1.1 Motivation

The results of our robustness testing indicate that this watermarking method is not robust with respect to images that have been JPEG compressed. Because this method is intended to be used for copyright enforcement, this shortcoming is unacceptable and must be improved upon. In order to do this, we consider the specifics of JPEG compression and why these failures occur. In particular, we focus our attention on the handling of the color data in the compression process. An input image is given as arrays of coordinates in the RGB color space, which are converted to the equivalent representation in the YC_rC_b color space.

In this space, the first coordinate (Y) represents the luminance at that location, while the C_r coordinate represents the difference between the red and luminance values and C_b represents the difference between the blue and luminance values. The luminance is computed as a linear combination of the input R, G, and B values, with the coefficient associated with B being significantly smaller than that of R or G. Because of this, most of the information content of the blue channel, particularly in blue or yellow colored regions of the image, is contained in the C_b coordinate.

Once the color space conversion is complete, the compression algorithm continues by sub-sampling the C_b and C_r channels by a factor of two in each direction. In effect, a block of four pixels in the C_b channel is replaced by its average (or something similar). If one of the four pixels in this block contains watermark data, much of it will be lost because of this sub-sampling step, resulting in decreased robustness.

In order to insure against this sub-sampling, we try embedding the watermark in 2-by-2 pixel neighborhoods instead of single locations. It is anticipated that the resulting images will show improved robustness toward JPEG compression. Unfortunately, since we are effectively decreasing the resolution of the watermark, it is expected that the subjective ratings of these images will worsen.

	baboon	*bear*	*fishingboat*	*lena*	*skyline arch*	*watch*
decode rate, uncompressed, density = 0.50						
κ=0.15	53%	39%	57%	89%	22%	3%
κ=0.20	63%	55%	76%	89%	45%	14%
κ=0.25	67%	75%	80%	93%	53%	13%
decode rate, compressed, density = 0.50						
κ=0.15	9%	1%	7%	5%	1%	0%
κ=0.20	13%	6%	13%	16%	5%	0%
κ=0.25	20%	14%	18%	41%	5%	1%
PSNR, density = 0.50						
κ=0.15	44.57	47.74	45.65	45.69	48.83	50.67
κ=0.20	42.25	45.25	43.17	43.18	46.50	48.16
κ=0.25	40.48	43.32	41.28	41.24	44.69	46.22
Subjective Rating, density = 0.50						
κ=0.15	2.0	1.9	2.8	2.5	2.7	1.7
κ=0.20	2.1	2.7	3.2	3.3	3.0	2.4
κ=0.25	2.6	3.2	3.6	3.8	3.5	2.9

Table 6: Robustness and perceptability results with JPEG enhanced embedding.

We experiment to determine how the use of this method changes the tradeoff between robustness toward JPEG compression and the degree to which the watermark is apparent to viewers. The time required for extraction or embedding is not significantly impacted by these changes.

6.1.2 Implementation

The embedding step for this method is only slightly different than embedding our robust watermark. We simply generate a mark with half as many rows and columns as before and then expand it in both directions by a factor of two. Once the unweighted mark is generated, we insert a copy of row 1 between the first and second rows, a copy of row 2 between the second and third rows, and so on. Similarly, we expand the mark by copying columns. The resulting mark will have 2-by-2 pixel neighborhoods that are identical. The watermark is visually masked and added to the blue channel giving our marked image.

The extraction step follows the same process as before, with one exception. Since we've copied the rows and columns of the watermark, the expected distance between peaks of the autocorrelation function doubles. In order to account for this, the (x, y) coordinates of the extracted peaks are divided by two.

6.1.3 Results

The results from this experiment are fairly disappointing. While we succeeded in increasing the decode rates for JPEG compressed images, the result is a

watermark that is still unusable for copyright enforcement. We are still unable to extract the message half of the time for any image. Moreover, our ability to successfully detect the watermark in uncompressed images was reduced as a result of switching to this method. Most of these new errors were caused by an inability to locate the peaks of the autocorrelation function. The reason for this is that we reduced the gradient around the area where the peaks should be found, as a result of the fact that the correlation was high when the shift was within 1 position of the true peak. It should be noted that this problem could be taken care of by only calculating the autocorrelation function at even shifts, at the expense of no longer handling those images that had been scaled by 0.5 (or less) in either direction.

The larger problem with this method, though, is that the increased robustness toward JPEG compression did not compare favorably with the worsening appearance of the images. The subjective ratings for these images increased substantially, with a significant number of viewers assigning the worst possible comparison. As a result, this method can't be viewed as an improvement on the algorithm as originally presented.

6.2 Automatic Framing

6.2.1 Motivation

As noted above, many of the failed detections for uncompressed images were the result of failed framing of the fixed image. While the fixed image may look to the human eye to be the same as the watermarked original from which it was derived, small errors in our matrix terms can result in extra rows or columns which are nearly blank. If the naive framing method keeps one of these while discarding one that contains more image data, the result will be misframed and the watermark will be undetectable.

Because of this, a method is desired to perform a more intelligent framing. This can be achieved because the image is almost in its original form, with the possibility that it may have too many or too few (in the case of cropping) rows and columns. Since we know the watermarking key, we can also determine the locations at which the watermark should have been embedded. While we don't know the expected sign of $\hat{w}$ at any given location (as that would be equivalent to knowing the message), we can accumulate the absolute values at these locations. We can also accumulate these values starting with the first pixel in the second row, or the first pixel in the second column. If, of these values, the sum is largest for locations starting in the second row, we can assume that the original image actually began at the second row. We do this over all possible shifts (positive as well as negative) and use the maximum sum to locate that position where the watermark actually begins. The filtered blue channel is then framed so that this point is in the upper-left corner.

It is expected that this method will significantly increase the detection rate, at least in the case of uncompressed images. Because we can only add the absolute values of the filtered blue channel, which changes significantly when

the image is JPEG compressed, we may not be able to accurately locate the beginning point. Because this method applies only to the decoding step, no changes are made to the encoding step. Thus, the visibility of the watermark remains the same, leaving only robustness to be examined.

Because this method involves computing the cross-correlation of two fairly large arrays, it is expected that the time required for extraction will be significantly increased by using this method.

6.2.2 Implementation

As during the watermark extraction, we generate an u-by-v (as defined in section 3.2.1) logical array identifying those locations where the watermark was embedded. We then insert a row of zeros between each pair of adjacent rows of this watermark embedding map, and then insert a column of zeros between each pair of adjacent columns. We compute the cross-correlation (the autocorrelation function, described earlier, is the case of cross-correlation where the two inputs are the same) of this result with the absolute value of $\hat{w}$. At this point, there should be four locations at which the cross correlation result has large values - one for each of the embedding locations. An embedding map is generated by adding shifted versions of the cross correlation function to account for the offsets of the multiple embeddings.

$$map_{x,y} = xcorr_{x,y} + xcorr_{x+\delta_x,y} + xcorr_{x,y+\delta_y} + xcorr_{x+\delta_x,y+\delta_y}$$

where $xcorr$ is the result of the cross-correlation between the watermark embedding locations and the absolute value of $\hat{w}$.

We find the (x, y) location of the map's maximum value. This value is where the upper left corner of the fixed image should be. If this value is $(0, 0)$ then the input image is properly aligned on the upper and left edges. In the event that the x coordinate is negative, we must add $|x|$ blank columns to the left side of the image; a positive value means that x columns must be removed from the left. Similarly, the value of y indicates how may rows must be added or removed from the top of the image.

Once these rows have been added or removed, the framing of the image is completed by adding or removing rows and columns from the bottom and right edges of the image in order to assure that the result is the same size as the original image.

6.2.3 Results

These results, shown in table 7, indicate that the automatic framing method provides a significant improvement over the naive method for uncompressed images. The fact that there were no significant improvements for the compressed imagery is consistent with the fact that the failed extractions for these were due to an inability to locate the peaks of the autocorrelation function.

Not shown in the table of results is the fact that, when using automatic framing, it was possible to extract the watermark from our *lena* image that

	baboon	*bear*	*fishingboat*	*lena*	*skyline arch*	*watch*
naive framing, uncompressed, density = 0.50						
κ=0.15	88%	58%	70%	76%	37%	21%
κ=0.20	79%	74%	79%	87%	74%	47%
κ=0.25	86%	72%	89%	92%	80%	66%
auto-framing, uncompressed, density = 0.50						
κ=0.15	96%	80%	82%	96%	53%	22%
κ=0.20	96%	89%	97%	95%	86%	61%
κ=0.25	97%	92%	97%	97%	95%	84%
naive framing, compressed, density = 0.50						
κ=0.15	0%	1%	1%	1%	1%	0%
κ=0.20	7%	3%	1%	2%	1%	0%
κ=0.25	17%	7%	7%	8%	2%	0%
auto-framing, compressed, density = 0.50						
κ=0.15	0%	1%	1%	2%	1%	0%
κ=0.20	8%	3%	1%	3%	1%	0%
κ=0.25	17%	7%	7%	8%	2%	0%

Table 7: Decode rates with and without auto-framing.

had been printed and then scanned. This only worked for the image that was scanned in the original orientation, as peak detection still failed for the image that was scanned at a different angle.

In addition to these results, the watermark extraction process using the automatic framing method was able to handle images that had been translated, an attack that is not represented in the StirMark test set. Because translation adds blank rows or columns to only one side of the image, the naive framing removes image content from the other side in order to keep some of these blank pixels. The automatic method has no such problem, and extraction is successful for arbitrary translations.

Finally, as expected, the time necessary for the extraction of the message from an image significantly increased. For our 200-by-200 pixel *baboon* image, the time required for the extraction process increased from 6.5 to 21.7 seconds.

6.3 Using Image Activity as a Visual Mask

6.3.1 Motivation

In our initial testing, it was noticed that images with large, blue areas, such as sky, started to look grainy when the watermark was embedded. This is because sky areas are typically rather high in luminance, which means that the watermark, with the high local scaling factor, introduced significant noise in these areas. The presence of yellow dots in the sky was quite objectionable, and suggested that the visual masking method should take this into account.

As suggested in [3] and other sources, we can use knowledge about the human visual system to develop a better embedding method. With respect to Kutter's algorithm, this comes in the form of an improved visual masking weight α. Instead of using the pixel's luminance to determine the local strength, we can take advantage of the fact that humans are less sensitive to changes made in areas of high texture than to those made in areas of uniform color. In order to achieve this, we use as a local weight the standard deviation of a small neighborhood of the image's luminance centered on the pixel in question. For the purposes of this experiment the neighborhood size was 5-by-5 pixels.

It is expected that the use of this visual mask will improve the tradeoff between robustness and the subjective rating of the watermarked image. The use of this visual masking method is not, however, expected to increase the $PSNR$ values of these image. This is because $PSNR$ is a numerical measure that has no notion of the characteristics of the human visual system which we are exploiting.

6.3.2 Implementation

While the standard deviation is a simple measure that indicates the absence or presence of texture within a certain part of an image, the values are typically much lower than the luminance at the same location. In order to assure that the same value of κ results in a watermark of similar strength, we scale the standard deviation values in order to match the watermark's power. That scale value is

$$Scale = \frac{\sum_i \sum_j |luma_{i,j} mark_{i,j}|}{\sum_i \sum_j |std_{i,j} mark_{i,j}|}$$

where *luma* is an array containing the luminance at each pixel, and *std* is an array containing the standard deviations in a 5-by-5 neighborhood centered at that pixel. At the edges of the image, where the neighborhood would include pixels outside of the original range, the edges were treated as reflectors. Thus, the value of $std_{1,1}$ is equal to the standard deviation of the neighborhood

$$\begin{array}{ccccc} luma_{3,3} & luma_{3,2} & luma_{3,1} & luma_{3,2} & luma_{3,3} \\ luma_{2,3} & luma_{2,2} & luma_{2,1} & luma_{2,2} & luma_{2,3} \\ luma_{1,3} & luma_{1,2} & luma_{1,1} & luma_{1,2} & luma_{1,3} \\ luma_{2,3} & luma_{2,2} & luma_{2,1} & luma_{2,2} & luma_{2,3} \\ luma_{3,3} & luma_{3,2} & luma_{3,1} & luma_{3,2} & luma_{3,3} \end{array}$$

The automatic framing method was used during this evaluation, in order to maintain high decode rates for uncompressed images.

6.3.3 Results

The first observation that should be made is that these results, when compared with the results in table 1, illustrate that the $PSNR$ value alone is not a good

	baboon	*bear*	*fishingboat*	*lena*	*skyline arch*	*watch*
decode rate, uncompressed, density = 0.50						
κ=0.15	95%	82%	91%	97%	41%	32%
κ=0.20	96%	91%	95%	97%	89%	67%
κ=0.25	96%	96%	97%	97%	95%	88%
decode rate, compressed, density = 0.50						
κ=0.15	1%	1%	1%	2%	1%	0%
κ=0.20	13%	2%	2%	9%	1%	0%
κ=0.25	16%	7%	7%	18%	5%	0%
PSNR, density = 0.50						
κ=0.15	43.68	47.91	44.01	43.47	46.59	49.44
κ=0.20	41.26	45.50	41.64	41.12	44.33	46.98
κ=0.25	39.40	43.65	39.84	39.35	42.59	45.10
Subjective Rating, density = 0.50						
κ=0.15	0.1	0.3	2.0	2.2	1.0	0.5
κ=0.20	0.4	0.3	3.0	3.2	1.3	1.4
κ=0.25	0.7	0.9	3.4	3.4	1.6	1.9

Table 8: Robustness and perceptability results with activity-based embedding.

indicator of overall image quality. Switching to the activity-based embedding method, we see that three images (*baboon*, *skyline arch*, and *watch*) had lower $PSNR$ values despite the fact that viewers rated them as more like the unmarked images.

The scaling method seems to have been successful in producing similar decode rates for this embedding as the luminance embedding for the same values of κ.

Overall, the results of this experiment are mixed. While the tradeoff between decode rate and the subjective rating was considerably improved for four of the images, it significantly worsened for the other two (*lena* and *fishingboat*). Because neither of these two images had an area of high-contrast activity, the areas where the visual mask allowed strong embedding were those where edges were present. Figure 4 shows the standard deviation visual masks for these two images (a brighter appearance indicates a high value of α); note that most of the brightness is centered on edges in the image. Because of this, large distortions were introduced at edges, which resulted in unpleasant looking images.

A metric other than standard deviation, which gives large values in areas of high activity but low values around edges, would be desirable to resolve the problems seen with these two images.

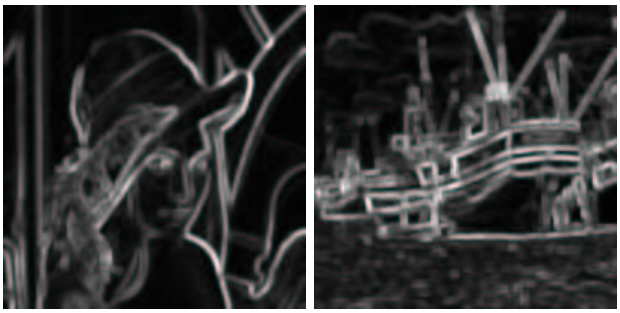

Figure 4: Standard Deviation visual masks for *lena* and *fishingboat*.

6.4 Using Color Error as a Visual Mask

6.4.1 Motivation

Like the previous effort, the purpose of this experiment is to develop a method of visual masking that provides an improved tradeoff between the robustness of the watermark and the subjective rating of the marked image. In order to do this, we take advantage of the fact that the RGB color space is not perceptually linear, which is to say that a change in the values of the blue channel will be less apparent in, for instance, a red colored area of an image than it will be in a blue area. In order to measure these color differences, we use the Δe^*_{ab} metric of the CIEL*a*b* system, as described in [2]. The important feature of the L*a*b* color space is that it is perceptually uniform, meaning that the euclidean distance between two points (colors) serves as a measure of the difference between them. In order to give the reader an intuition, the L^* coordinate represents the brightness of a pixel while the a^* and b^* coordinates give a measure of red/green and blue/yellow, respectively.

$$\Delta e^*_{ab} = \sqrt{\Delta L^{*^2} + \Delta a^{*^2} + \Delta b^{*^2}}$$

This metric expresses the difference between two colors – the original and the color resulting from the addition of the watermark – by calculating the length of their connecting vector in the L*a*b* color space.

We find this mask by first producing an estimate of the watermarked image, where the value of α is equal to 1 at all locations. That image is compared with the original, and the Δe^*_{ab} is found at each pixel location. This error estimate is used to generate the actual visual mask α: those locations where the estimated error is high will get low values in α and those with low errors will get high values.

6.4.2 Implementation

The first step in generating this visual mask is finding the estimate of the watermarked image by embedding the mark with a uniform visual mask. Both the

	baboon	*bear*	*fishingboat*	*lena*	*skyline arch*	*watch*
decode rate, uncompressed, density = 0.50						
κ=0.15	96%	79%	82%	96%	41%	25%
κ=0.20	97%	91%	96%	97%	63%	62%
κ=0.25	96%	92%	96%	97%	84%	87%
decode rate, compressed, density = 0.50						
κ=0.15	0%	1%	1%	2%	1%	0%
κ=0.20	9%	1%	1%	6%	1%	0%
κ=0.25	16%	1%	2%	17%	1%	0%
PSNR, density = 0.50						
κ=0.15	44.71	50.02	45.40	45.32	49.98	51.56
κ=0.20	42.26	47.57	43.06	42.89	47.88	49.10
κ=0.25	40.39	45.66	41.23	41.02	46.36	47.15
Subjective Rating, density = 0.50						
κ=0.15	0.7	0.0	1.1	1.1	0.4	0.3
κ=0.20	1.1	0.2	2.1	1.8	0.8	0.7
κ=0.25	1.3	0.4	2.6	2.4	1.6	1.2

Table 9: Robustness and perceptability results with color error visual mask.

original and estimate image are transformed into L*a*b* space and the Δe^*_{ab} values are computed at each location. These values are scaled to the range [0,1] by dividing by the maximum value over the entire image. These scaled values are inverted by the function

$$f(x) = |x - 1|,$$

and then scaled, as in the previous experiment, to produce a mark of the same power. As with the activity-based embedding, the results were obtained using the automatic framing method to increase detector performance.

6.4.3 Results

For the most part, it seems that the scaling of the watermark worked in giving similar decode rates for the color encoding as the luma encoding with the same value of κ. The notable exception to this is the *skyline arch* image, which features a lot of blue and yellow colors where our changes would be more visible. Even in this case, the use of the color embedding seems to have improved the tradeoff between visibility and robustness. With the luminance embedding and κ=0.20, we had a decode rate of 86% and a subjective rating of 2.1. The color-based embedding, on the other hand, had a decode rate of 84% and a subjective rating of 1.6, though this was when κ=0.25. This represents a significant improvement in the subjective rating for a negligible change in decoding ability.

The *bear* image seems to have shown the most improvement as a result of moving toward this embedding method. Decode rates were essentially un-

changed while the subjective ratings of the watermarked images dropped no less than 0.8 for any value of κ.

The remainder of the images show reasonable gains, most notably at lower embedding strengths. More importantly, none of these results indicate that, in any combination, the tradeoff between robustness and subjective visibility changes for the worse. As a result, it is fair to say that the color-based embedding method is superior to the luminance-based method presented in [7].

7 Conclusion

Based on the results of these tests, it should be obvious that the use of this watermark for its intended purpose would be inappropriate due to the low detection rate for images that have been JPEG compressed. As an overall indicator of this, we can look to a rating value derived from the StirMark test, which is essentially an average decode rate for different types of attack. As a reference, the Digimarc system has an average of 78 and the SureSign product has an average of 70. This method's average of 50 is clearly a step below either of these, and further analysis indicates that this is because these two systems are more successful with images that have been JPEG compressed.

Despite this, the experiments performed in conjunction with this testing indicate some significant improvements over the original system. In particular, we show that the use of color error estimates as a visual masking method is preferable to using the image's luminance. This should be kept in mind for future endeavors in watermarking.

8 Future Work

- As suggested in [8], the use of error correcting codes could be used to improve the rate of successful watermark extraction. While this method would not likely result in a significant improvement for JPEG compressed images (since most of these failures were due to failed peak detection) it could improve the fundamental tradeoff between robustness and the strength of the watermark.

- In order to improve the algorithm's performance on images that have been JPEG compressed, the method of peak detection in the autocorrelation function should be revisited. While many of the failed extractions were associated with autocorrelation where there were no obvious peaks, there were a number of instances where the locations were noticeable to the human eye but not to the extraction method.

- Given the success of the color-based embedding scheme, as well as the more moderate success of the activity-based encoding, it would be interesting to experiment with a weighted average of these two. Some of the more objectionable images generated with color embedding were those where

the mark was obvious in areas of low activity. A combination of these two might provide a significant improvement, given that it doesn't produce artifacts at the edges.

- In order to improve the speed of the extraction step, one could investigate the embedding of the watermark in a non-symmetric manner. Instead of embedding the watermark with offsets of $(0,0)$, $(\delta_y,0)$, $(0,\delta_x)$, and (δ_y,δ_x) we could replace the last one with (δ_x,δ_y). This would result in an autocorrelation function that has fewer symmetries. Because of this, the number of potential inverse matrices would be reduced, lessening the effort required during extraction.

9 Acknowledgements

The author would like to thank Martin Kutter for his timely responses to questions via e-mail.

Secondly, I would like to thank the 10 judges who helped in the subjective rating of the images: Jeff Baumes, Danielle Brahm, Gregg Bryant, Meredith Graham, Montgomery Q. Mitra, Efrain Morales, Mike Murdoch, Ricardo Rosario, Dave Rossing, and Jennifer Stanek.

A StirMark Test Images

Figure 5: *baboon* and *fishingboat* images

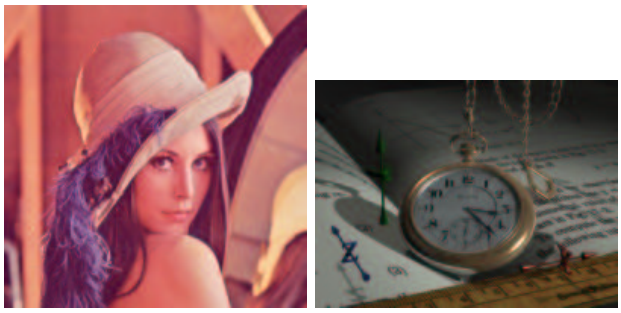

Figure 6: *lena* and *watch* images

Figure 7: *skyline arch* and *bear* images

B Matlab Embedding Code

```
function EmbedNewWatermark(inName, outName, mark, key, parameters)
%function EmbedNewWatermark(inName, outName, mark, key, parameters)
%
% Embeds the resilient watermark in an image.
%
% Inputs:
%  inName - the name of the source image file
%  outName - the file to which the watermarked image shall be saved
%  mark - a binary string representing the watermark to be embedded
%  key - the random seed used as a key for the algorithm
%  parameters - a structure containing the parameters of the embedding. Fields:
%       strength - the strength of the watermark
%       density - the percent of pixels that are involved in the watermark
%       shiftRows - the row displacement for overlaping watermarks
%       shiftCols - the column displacement for overlaping watermarks
%       maskMethod - the visual masking method to be used
%
% Outputs: none
%

if (nargin == 4)
   parameters = load('defaultWMparams', 'parameters');
end
```

```
rand('state', key);

img = double(imread(inName));
[numRows, numCols, numChannels] = size(img);

markRows = floor((numRows - parameters.shiftRows)/2);
markCols = floor((numCols - parameters.shiftCols)/2);

if (numChannels ~= 3)
   error('This function only works on RGB images');
end

[embedBits,locations] =GenerateSingleMark(parameters, mark, markRows, markCols);

origins = [1 1 ; parameters.shiftRows+1 1 ; 1 parameters.shiftCols+1 ; ...
           parameters.shiftRows+1 parameters.shiftCols+1];
ends(:,1) = origins(:,1) + 2 * markRows - 1;
ends(:,2) = origins(:,2) + 2 * markCols - 1;

wMark = zeros(size(img(:,:,3)));

for i = 1:4
   tempMark = wMark(origins(i,1):2:ends(i,1), origins(i,2):2:ends(i,2));
   tempMark(locations) = embedBits * parameters.strength;
   wMark(origins(i,1):2:ends(i,1), origins(i,2):2:ends(i,2)) = tempMark;
end

visualMask = GenerateVisualMask(img, parameters, wMark);

img(:,:,3) = img(:,:,3) + wMark .* visualMask;

[junk,junk,ext] = fileparts(outName);

switch ext
   case '.jpg',
      imwrite(uint8(img), outName, 'Quality', 100);
   case '.jpeg',
      imwrite(uint8(img), outName, 'Quality', 100);
   case '.ppm',
      WritePPM(uint8(img), outName);
   otherwise,
      imwrite(uint8(img), outName);
end

%%%% End EmbedNewWatermark %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [embedBits, locations] = ...
                    GenerateSingleMark(params, mark, numRows, numCols)

mark = 2 * [0 1 double(mark) - double('0')] - 1;
markbits = length(mark);

locations = find( rand(numRows, numCols) < params.density);
numSites = length(locations);

embedMap = floor(rand(numSites, 1) * markbits) + 1;
embedBits = mark(embedMap)';

%%%% End GenerateSingleMark %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function visualMask = GenerateVisualMask(img, parameters, wMark)

nhSize = 2;

RHO    = 0.299;
BETA   = 0.114;
GAMMA  = 0.587;

luma = RHO .* img(:,:,1) + GAMMA .* img(:,:,2) + BETA .* img(:,:,3);
intendedPower = sum(sum(abs(wMark .* luma)));

switch parameters.maskMethod
   case 'luma',
      visualMask = luma;

   case 'std',

      superLuma = luma([nhSize:-1:1 1:end end:-1:end-(nhSize-1)],:);
      superLuma = superLuma(:, [nhSize:-1:1 1:end end:-1:end-(nhSize-1)]);
      clear luma

      next = 1;

      for rowOff = 0 : 2 * nhSize
         for colOff = 0 : 2 * nhSize
            temp = superLuma(1 + rowOff : end - (2 * nhSize - rowOff), ...
                             1 + colOff : end - (2 * nhSize - colOff));
            linImg(next,:) = temp(:)';
            next = next + 1;
         end
      end
```

```
      visualMask = reshape(std(linImg), size(img,1), size(img,2)) * 5;
      clear linImg superLuma temp

   case 'deltaE',
      testBlue = img(:,:,3) + luma .* wMark;

      img2 = img;
      img2(:,:,3) = testBlue;

      deltaE = GetDeltaE(img, img2);
      deltaE = deltaE / max(max(deltaE));
      visualMask = abs(deltaE - 1);
      clear deltaE

end

actualPower = sum(sum(abs(wMark .* visualMask)));
scaleVal = intendedPower / actualPower;
visualMask = visualMask * scaleVal;

%%%% End GenerateVisualMask %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function deltaE = GetDeltaE(img1, img2)

[L1, a1, b1] = RGBtoLab(img1/255);
[L2, a2, b2] = RGBtoLab(img2/255);

deltaE = sqrt((L1-L2).^2 + (a1-a2).^2 + (b1-b2).^2);

%%%% End GetDeltaE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [L, a, b] = RGBtoLab(img)

RGBtoXYZ = [0.412453 0.357580 0.180423; ...
            0.212671 0.715160 0.072169; ...
            0.019334 0.119193 0.950227];

red = img(:,:,1);
green = img(:,:,2);
blue = img(:,:,3);
rgbvec = [red(:)' ; green(:)' ; blue(:)'];
clear red green blue
XYZvec = RGBtoXYZ * rgbvec;
clear rgbvec
L = 116 * XYZvec(2,:).^(1/3) - 16;
a = 500 * ((XYZvec(1,:)/0.95).^(1/3) - XYZvec(2,:).^(1/3));
```

```
b = 200 * (XYZvec(2,:).^(1/3) - (XYZvec(3,:)/1.09).^(1/3));
clear XYZvec

L = reshape(L, size(img,1), size(img,2));
a = reshape(a, size(img,1), size(img,2));
b = reshape(b, size(img,1), size(img,2));
```

C Matlab Extraction Code

```
function [mark, confidence, map] = ...
ExtractNewWatermark(inName, key, parameters, originalSize, fixedImage)
%function [mark, confidence, map] =
% ExtractNewWatermark(inName, key, parameters, originalSize, fixedImage)
%
% Extracts the robust watermark from an image.
%
% Inputs:
%  inName - the name of the watermarked image file
%  key - the random seed used as a key for the algorithm
%  parameters - a structure containing the parameters of the embedding. Fields:
%       strength - the strength of the watermark
%       density - the percent of pixels that are involved in the watermark
%       neighborhood - the size of the cross-shaped prediction neighborhood
%       markbits - the number of bits in the watermark, including leading 1 0
%  originalSize - the size of the image at the time of embedding
%  fixedImage (optional) - the name of a file to which the fixed image
%                                should be saved
%
% Outputs:
%  mark - the binary string payload of the watermark
%  confidence - a measure of the certainty with which the mark has been found
%  map - a map of the estimated locations of the watermark

confidence = 0;
mark='';

[junk,junk,ext] = fileparts(inName);

switch ext
   case '.ppm',
      img = ReadPPM(inName);
   otherwise,
      img = double(imread(inName));
end

[numRows, numCols, numChannels] = size(img);

if (numChannels ~= 3)
   error ('This function only works on RGB images.');
end

blue = img(:,:,3);
```

```
clear img

wMark = Prefilter(blue);
clear blue
map = xcorrf2(wMark);
map = NormalizeMap(map, originalSize, parameters);

%save tempMap map wMark

peaks = ExtractPeaks(map, 5);

%Crop map for display in GUI
minp = round(min(peaks(:,2:-1:1)) + floor(size(map)/2) - 10);
maxp = round(max(peaks(:,2:-1:1)) + floor(size(map)/2) + 10);
map = map(minp(1):maxp(1), minp(2):maxp(2));
clear minp maxp

%Get candidate transforms
[thetas, peaks] = ThetaSort(peaks);
[Aprimes, Bprimes] = GetPotentialPairs(thetas, peaks);
matrices = GetInverseMatrices(Aprimes, Bprimes, parameters);

mark = repmat(char(double('_'*ones(1,parameters.markbits-2))),size(matrices,3),1);
confidence = zeros (1, size(matrices,3));

%Try extracting from the attacked image.
[mark(1,:), confidence(1)] = ...
   ExtractNewWatermark_ChannelFixed(NaiveFrame(wMark,originalSize), ...
                                    key,parameters);

%Try extracting from the candidate fixed images.
for i = 2 : 1 + size(matrices, 3)
   try
      curMark = UndoGeometricAttack(wMark, matrices(:,:,i - 1));
      curMark = AutoFrameChannel(curMark, originalSize, key, parameters);
%     curMark = NaiveFrame(curMark, originalSize);
      [mark(i,:), confidence(i)] = ...
            ExtractNewWatermark_ChannelFixed(curMark,key,parameters);
   catch
   end
end

%Keep the one with the highest confidence.
confidenceMax = max(confidence);
bestInd = min(find(confidence == confidenceMax));
mark = mark(bestInd, :);
```

```
confidence = confidenceMax;

if (nargin == 5)
   msg = ['saving fixed image to ' fixedImage]
   PerformGeometricAttack(inName, fixedImage, matrices(:,:,bestInd));
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function matrices = GetInverseMatrices(Aprimes, Bprimes, parameters)
% Gets the candidate transforms given a set of corresponding points

matrices = zeros(3 , 3, size(Aprimes, 1));

for i = 1 : size(Aprimes, 1)
   matrices(:,:, i) = eye(3);
   LHmatrix = [parameters.shiftCols 0; 0 parameters.shiftRows ];
   RHmatrix = [Aprimes(i,:) ; Bprimes(i,:) ]';

   if (abs(det(RHmatrix)) > .005)
      %Make sure we're not going to cause an error
      matrices(1:2,1:2,i) = (LHmatrix * inv(RHmatrix));
   end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Aprimes, Bprimes] = GetPotentialPairs(thetas, peaks)
%Gets pairs of points (peaks from the autocorrelation function)
% that could be tranformed versions of our starting points

nextSpace = 1;

for i = 2 : size(peaks, 1)

   next = 2 + mod(i - 1, length(thetas) - 1);
   Aprimes(nextSpace : nextSpace + 1, :) = peaks( [i next], :);
   Bprimes(nextSpace : nextSpace + 1, :) = peaks( [next i], :);
   nextSpace = nextSpace + 2;

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [thetas, sortedPeaks] = ThetaSort(peaks)
%Sorts the peaks by their angle relative to the X axis.
%The angle of the origin is defined to be -1
```

```
for i = 1 : size(peaks, 1)

   if ( sum( peaks(i,:).^2 ) == 0 )
      thetas(i) = -1;
   elseif ( peaks(i, 2) >= 0 )
      thetas(i) = acos( peaks(i, 1) / sqrt( sum( peaks(i,:).^2 ) ) );
   else
      thetas(i) = pi + acos( -peaks(i, 1) / sqrt( sum( peaks(i,:).^2 ) ) );
   end

end

[thetas, ind] = sort(thetas);
sortedPeaks = peaks(ind, :);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function img = UndoGeometricAttack(img, matrix)
%Applies the transformation matrix to the watermark channel in an attempt
% to undo the attack.

[oldYMax, oldXMax] = size(img);

corners = [1 1 1;1 oldYMax 1; oldXMax oldYMax 1;oldXMax 1 1]' - 1;

transCorners = round(matrix * corners);

newYMin = min(transCorners(2,:));
newYMax = max(transCorners(2,:));
newXMin = min(transCorners(1,:));
newXMax = max(transCorners(1,:));

newXRange = newXMax - newXMin + 1;
newYRange = newYMax - newYMin + 1;

if (newXRange * newYRange * 8) > 10 * 2^20
   error('UndoGeometricAttack: fixed image would be too big.');
end

map = ones(3 * newXRange * newYRange, 1);

XRamp = [newXMin : newXMax];
YRamp = [newYMax : -1 : newYMin]';

temp = repmat(YRamp,1, newXRange);
```

```
map(2:3:end)=temp(:);

temp = repmat(XRamp, newYRange, 1);
map(1:3:end)=temp(:);
clear temp

invMatrix = matrix ^-1;

sources = invMatrix * reshape(map, 3, newXRange * newYRange);

img = SuperFancyIndex(img, sources, newXRange, newYRange);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function outImg = SuperFancyIndex(img, sources, newXRange, newYRange)
%Performs an indexing, of sorts, to img.

[oldYMax, oldXMax] = size(img);
wm = [0 img(1 : oldXMax * oldYMax)];

floorX = floor(sources(1, :));
floorY = floor(sources(2, :));

sources(sources < 0) = -1;
sourceX = floorX;
sourceY = floorY;
scale = (floorX+1 - sources(1,:)) .* (floorY+1 - sources(2,:));
sourceX(sourceX > oldXMax - 1) = -1;
sourceY(sourceY > oldYMax - 1) = -1;
inds = sourceX * oldYMax + (oldYMax - 1 - sourceY);
inds((sourceX < 0) | (sourceY < 0)) = -1;
outImg = reshape(wm(inds+2).*scale, newYRange, newXRange);

sourceX = floorX + 1;
scale = (sources(1,:) - floorX ) .* (floorY+1 - sources(2,:));
sourceX(sourceX > oldXMax - 1) = -1;
inds = sourceX * oldYMax + (oldYMax - 1 - sourceY);
inds((sourceX < 0) | (sourceY < 0)) = -1;
outImg = outImg + reshape(wm(inds+2).*scale, newYRange, newXRange);

sourceY = floorY + 1;
scale = (sources(1,:) - floorX ) .* (sources(2,:) - floorY);
sourceY(sourceY > oldYMax - 1) = -1;
inds = sourceX * oldYMax + (oldYMax - 1 - sourceY);
inds((sourceX < 0) | (sourceY < 0)) = -1;
outImg = outImg + reshape(wm(inds+2).*scale, newYRange, newXRange);
```

```
sourceX = floorX;
scale = (floorX + 1 - sources(1,:)) .* (sources(2,:) - floorY);
sourceX(sourceX > oldXMax - 1) = -1;
inds = sourceX * oldYMax + (oldYMax - 1 - sourceY);
inds((sourceX < 0) | (sourceY < 0)) = -1;
outImg = outImg + reshape(wm(inds+2).*scale, newYRange, newXRange);

clear sources inds scale sourceX sourceY


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function pf = AutoFrameChannel(pf, originalSize, key, parameters)
%Attempts to frame a fixed image based on the position of the watermark.

rand('state', key);

[numRows, numCols] = size(pf);

markRows = floor((originalSize(1) - parameters.shiftRows)/2);
markCols = floor((originalSize(2) - parameters.shiftCols)/2);

if (numRows < markRows) | (numCols < markCols)
   %Image is too small, just do the naive framing
   pf = NaiveFrame(pf, originalSize);
   return
end

%wmMap will be a map of watermark locations
temp = rand(markRows, markCols) < parameters.density;
wmMap = zeros(2 * size(temp));
wmMap(1:2:end, 1:2:end) = temp;
clear temp

[mapRows, mapCols] = size(wmMap);
[imgRows, imgCols] = size(pf);

map = xcorrf2(abs(pf), wmMap);

%save tempMap map pf wmMap

sums = map(1 : end - parameters.shiftRows, 1 : end - parameters.shiftCols) + ...
       map(1 + parameters.shiftRows : end, 1 : end - parameters.shiftCols) + ...
       map(1 : end - parameters.shiftRows, 1 + parameters.shiftCols : end) + ...
       map(1 + parameters.shiftRows : end, 1 + parameters.shiftCols : end);
```

```
[firstPeak(:,1), firstPeak(:,2)] = find(sums == max(max(sums)));
clear sums map

if (size(firstPeak, 1) ~= 1)
   error('AutoFrameChannel: Too many maxima in sums');
end

extraDims = firstPeak - size(wmMap);
pf = Frame(pf, extraDims, originalSize);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function img = Frame(img, extraDims, originalSize)
%Frames the image, restoring it to its original size.

if (extraDims(1) > 0)
   img = img(1 + extraDims(1) : end, :);
elseif (extraDims(1) < 0)
   img = [zeros(size(img(1,:))); img];
   img = img([ones(1, abs(extraDims(1))) 2:end], :);
end

if (extraDims(2) > 0)
   img = img(:, 1 + extraDims(2) : end);
elseif (extraDims(2) < 0)
   img = [zeros(size(img(:,1))) img];
   img = img(:, [ones(1, abs(extraDims(2))) 2:end]);
end

extraDimsEnd = size(img) - originalSize;

if (extraDimsEnd(1) > 0)
   img = img(1 : end - extraDimsEnd(1), :);
elseif (extraDimsEnd(1) < 0)
   img = [img; zeros(size(img(1,:)))];
   img = img([1 : end-1 end*ones(1, abs(extraDimsEnd(1)))], :);
end

if (extraDimsEnd(2) > 0)
   img = img(:, 1 : end - extraDimsEnd(2));
elseif (extraDimsEnd(2) < 0)
   img = [img zeros(size(img(:,1)))];
   img = img(:, [1 : end-1 end*ones(1, abs(extraDimsEnd(2)))]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [mark, confidence] = ...
```

```
        ExtractNewWatermark_ChannelFixed(pf, key, parameters, originalSize)
%Extracts the watermark from the fixed watermark channel.
%Assumes that size(pf) = originalSize

rand('state', key);

[numRows, numCols] = size(pf);

markRows = floor((numRows - parameters.shiftRows)/2);
markCols = floor((numCols - parameters.shiftCols)/2);

[embedMap, locations] = GetMarkLocations(parameters, markRows, markCols);

origins = [1 1 ; parameters.shiftRows+1 1 ; 1 parameters.shiftCols+1 ; ...
           parameters.shiftRows+1 parameters.shiftCols+1];
ends(:,1) = origins(:,1) + 2 * markRows - 1;
ends(:,2) = origins(:,2) + 2 * markCols - 1;

bitSums = zeros(1, parameters.markbits);

for i = 1:4
   %Sum over the 4 embeddings.

   segment = pf(origins(i,1):2:ends(i,1), origins(i,2):2:ends(i,2));
   markPixels = segment(locations);
   clear segment

   for j = 1 : parameters.markbits
      bitSums(j) = bitSums(j) + sum(markPixels(embedMap == j));
   end
end

confidence = abs(bitSums(1) - bitSums(2));
threshold = mean(bitSums(1:2));
markBits = bitSums > threshold;
mark = char(markBits(3:end) + double('0'));


%%%% End ExtractNewWatermark_Fixed %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [embedMap, locations] = GetMarkLocations(params, numRows, numCols)
%Gets the watermark locations, as well as identifies which bit is embedded
% at each location

locations = find( rand(numRows, numCols) < params.density);
numSites = length(locations);
```

```
embedMap = floor(rand(numSites, 1) * params.markbits) + 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function img = NaiveFrame(img, originalSize)
%Performs naive framing, making sure that the image is of its original size.

extraDims = floor((size(img) - originalSize)/2);

if (extraDims(1) > 0)
   img = img(1 + extraDims(1) : end, :);
elseif (extraDims(1) < 0)
   img = [zeros(size(img(1,:))); img];
   img = img([ones(1, abs(extraDims(1))) 2:end], :);
end

if (extraDims(2) > 0)
   img = img(:, 1 + extraDims(2) : end);
elseif (extraDims(2) < 0)
   img = [zeros(size(img(:,1))) img];
   img = img(:, [ones(1, abs(extraDims(2))) 2:end]);
end

extraDimsEnd = size(img) - originalSize;

if (extraDimsEnd(1) > 0)
   img = img(1 : end - extraDimsEnd(1), :);
elseif (extraDimsEnd(1) < 0)
   img = [img; zeros(size(img(1,:)))];
   img = img([1 : end-1 end*ones(1, abs(extraDimsEnd(1)))], :);
end

if (extraDimsEnd(2) > 0)
   img = img(:, 1 : end - extraDimsEnd(2));
elseif (extraDimsEnd(2) < 0)
   img = [img zeros(size(img(:,1)))];
   img = img(:, [1 : end-1 end*ones(1, abs(extraDimsEnd(2)))]);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function map = NormalizeMap(map, originalSize, parameters)
%Normalizes the autocorrelation function, accounting for the fact that
% values toward the outside are based on fewer terms in the summation.

attackedWidth = (size(map, 2) + 1) / 2;
attackedHeight = (size(map, 1) + 1) / 2;
```

```
attackedDiag = sqrt(attackedHeight.^2 + attackedWidth.^2);

minDim = min(originalSize);

imgScale = attackedDiag/minDim;

maxShift = max(parameters.shiftRows, parameters.shiftCols) + 1;

mapScale = [1:attackedHeight attackedHeight-1:-1:1]' * ...
           [1:attackedWidth attackedWidth-1:-1:1];
mapScale = max(mapScale, (attackedWidth - maxShift * imgScale) * ...
               (attackedHeight - maxShift * imgScale));

map = map ./ mapScale;
```

References

[1] Jana Dittmann, Mark Stabenau, Ralf Steinmetz. *Robust MPEG Video Watermarking Technologies.* Proceedings of the Sixth ACM International Conference on Multimedia, 1998, pp. 71–80.

[2] R. W. G. Hunt. *The Reproduction of Colour* England: Fountain Press. 1995.

[3] Mohan S Kankanhalli, K. R. Ramakrishnan. *Content Based Watermarking of Images.* Proceedings of the Sixth ACM International Conference on Multimedia, 1998, pp. 61–70.

[4] Stefan Katzenbeisser, Fabien A. P. Petitcolas, editors. Information Hiding: Techniques for Steganography and Digital Watermarking. Boston, MA: Artech House. 2000.

[5] Jack Kelly. *Terror Groups Hide Behind Web Encryption.* USA Today, Feb. 2 2001.

[6] Martin Kutter, Frederic Jordan, Frank Bossen. *Digital Watermarking of Color Images Using Amplitude Modulation.* Journal of Electronic Imaging, Vol. 7(2), April 1998, pp. 326–332.

[7] Martin Kutter. *Watermarking Resisting to Translation, Rotation, and Scaling.* SPIE Conference on Multimedia Systems and Applications, November 1998, pp. 423–431.

[8] Christopher Martin. *Digital Image Watermarking.* RIT Master's Thesis, 2000.

[9] Fabien A. P. Petitcolas, Ross J. Anderson, Markus G. Kuhn. *Attacks on Copyright Marking Systems*, in David Aucsmith (Ed), Information Hiding, Second International Workshop, IH'98, Portland, Oregon, U.S.A., April 15-17, 1998, Proceedings, LNCS 1525, Springer-Verlag, ISBN 3-540-65386-4, pp. 219–239.

[10] Fabien A. P. Petitcolas. *Watermarking Schemes Evaluation.* I.E.E.E. Signal Processing, vol. 17, no. 5, pp. 58–64, September 2000.

[11] StirMark 3.1 Overview,
www.cl.cam.ac.uk/~fapp2/watermarking/stirmark/

[12] Digital Watermarking Photo Database,
www.cl.cam.ac.uk/~fapp2/watermarking/benchmark/image_database.html