UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

ESCOLA POLITÉCNICA

DEPARTAMENTO DE ELETRÔNICA E DE COMPUTAÇÃO

MARCA D'ÁGUA DIGITAL PARA INSERÇÃO IMPERCEPTÍVEL DE INFORMAÇÃO DE CALIBRAÇÃO EM IMAGENS PARA DIMENSIONAMENTO 3-D

Autor: _____

Gabriel Mayrink da Rocha Hospodar

Orientador: _____

Prof. José Gabriel Rodríguez Carneiro Gomes, Ph.D.

Examinador: _____

Prof. Mariane Rembold Petraglia, Ph.D.

Examinador: _____

Prof. Gelson Vieira Mendonça, Ph.D.

DEL

Novembro de 2007

À minha família.

Agradecimentos

- A Deus.
- Especialmente à minha família.
- Ao professor, orientador e amigo José Gabriel Rodríguez Carneiro Gomes pela clareza, pela simplicidade e pelo brilhantismo com que conduziu a orientação deste trabalho, agregando bastante valor à minha formação.
- Às amizades criadas no ambiente universitário, essenciais para o sucesso dessa missão devido ao companheirismo e ao espírito de equipe.
- Aos amigos de longa data pelo apoio, pela paciência e pelos momentos de descontração.
- Ao Departamento de Eletrônica e de Computação e à Escola Politécnica da UFRJ pelo excelente curso de Engenharia oferecido. Aos excelentes professores que souberam ensinar e compartilhar suas experiências magistralmente.

Resumo

Técnicas de marca d'água digital para inserção imperceptível de informação em imagens têm diversas aplicações. As mais comuns envolvem proteção de direitos autorais, permitindo-se a identificação de um comprador de uma cópia legal de um produto, aplicações na área de segurança, onde por exemplo uma assinatura digital pode autenticar mapas militares etc.

Atualmente encontra-se em desenvolvimento no Laboratório de Processamento Analógico e Digital de Sinais (PADS) da Universidade Federal do Rio de Janeiro um sistema de dimensionamento tridimensional submarino que será utilizado para aplicações da Petrobras. Neste sistema, é útil inserir informações de calibração das câmeras nas próprias imagens conservando o tipo de arquivo *bitmap*, o que pode ser feito através do uso de marcas d'água digitais.

Este trabalho aborda os métodos *Lower Bit Modulation* e *Quantization Index Modulation* para inserção de marca d'água digital em imagens, comparando-os em função das distorções apresentadas nas imagens marcadas e das capacidades de informação que cada método suporta.

As implementações foram feitas na linguagem de programação C. Os testes realizados mostraram que o método *Lower Bit Modulation* apresentou melhores resultados, inserindo as informações de calibração de forma imperceptível e com razão sinal-ruído de aproximadamente 71 dB.

Palavras-chave: marca d'água digital, inserção imperceptível de informação, *Lower Bit Modulation*, *Quantization Index Modulation*, imagens de calibração.

Sumário

Lista de Figuras **vii**

Lista de Tabelas **ix**

1 Introdução **1**

1.1 Contribuições do Trabalho 2

1.2 Organização do Documento 3

2 Teoria **4**

2.1 *Bitmap* 4

2.2 *Lower Bit Modulation* (LBM) 6

2.3 *Quantization Index Modulation* (QIM) 8

2.4 Razão Sinal-Ruído 10

3 Métodos **13**

3.1 *Lower Bit Modulation* (LBM) 15

3.2 *Quantization Index Modulation* (QIM) 18

4 Resultados **22**

4.1 *Lower Bit Modulation* (LBM) 22

4.2 *Quantization Index Modulation* (QIM) 25

4.2.1 Inserção de 1 Bit por Bloco 25

4.2.2 Inserção de 2, 3 ou 4 Bits por Bloco 29

4.3 Marca d'Água nas Imagens de Calibração 31

5 Conclusão **37**

Referências **38**

Apêndice A – Código de Escrita de Marca d'Água do Método LBM **39**

Apêndice B – Código de Leitura de Marca d'Água do Método LBM **53**

Lista de Figuras

1 Espaço de cores RGB 7

2 Exemplo de *bitmap*, com destaque para três *pixels* da região esquerda inferior da imagem 7

3 Diagramas de bloco 14

4 Fluxograma que descreve a escrita da marca d'água 16

5 Fluxograma que descreve a leitura da marca d'água 17

6 Imagem hospedeira lena.bmp 23

7 Performance do método *Lower Bit Modulation* (Tabela 2) 25

8 Comparação das imagens marcadas pelo método *Lower Bit Modulation* 26

9 Subtração entre a imagem marcada com mensagem de 20000 caracteres e a imagem hospedeira original 27

10 Comparação entre os métodos LBM (Tabela 2) e QIM: 1 bit (Tabela 3) 29

11 Comparação das imagens marcadas pelo método *Quantization Index Modulation*: 1 bit 30

12 Comparação entre os métodos LBM (Tabela 2) e QIM (Tabelas 3 e 4) 31

13 Comparação entre a imagem hospedeira (a) e a imagem marcada com PSNR = 56,51 dB (b) 32

14 Imagem de calibração original 33

15 Imagem de dimensionamento original 34

16 Imagem de calibração marcada 35

17 Imagem de dimensionamento marcada 36

Lista de Tabelas

1 Parâmetros do cabeçalho do *bitmap* 6

2 Resultados obtidos através do método *Lower Bit Modulation* 24

3 Resultados obtidos através do método *Quantization Index Modulation* - 1 bit por bloco 28

4 Resultados obtidos através do método *Quantization Index Modulation* - 2, 3 e 4 bits por bloco 30

1 Introdução

Sistemas de marca d'água digital têm diversas aplicações importantes. Estes sistemas inserem um sinal, chamado de marca d'água, em outro sinal, chamado de sinal hospedeiro [9]. A inserção da marca d'água não deve causar sérias distorções ao sinal hospedeiro e, além disto, ela deve ser robusta às degradações sofridas pela imagem composta, que é formada pelo sinal hospedeiro e pela marca d'água. Em algumas aplicações as degradações resultam de processos de transmissão.

Dentre as principais aplicações relacionadas à marca d'água digital pode-se citar a proteção de direitos autorais em áudio, vídeos e imagens. Neste caso, o sinal incorporado pode indicar o proprietário de algum registro de autor (*copyright*) ou inibir cópias não autorizadas. Por exemplo, a marca d'água pode ser um sinal digital que identifica unicamente o comprador da cópia legal. Caso cópias ilícitas do trabalho fossem feitas, todas elas carregariam este identificador, tornando fácil a identificação do proprietário da versão original. Em outro exemplo, o sinal incorporado pode habilitar ou desabilitar a cópia através de um sistema de duplicação que confere a marca d'água antes de iniciar sua execução. Este último sistema foi proposto para possibilitar a funcionalidade de permitir apenas uma cópia em gravadores de vídeo digital [2]. Alternativamente, um tocador de mídia padronizado poderia decidir entre tocar ou não um disco, ajudando no combate à pirataria [6].

Outras aplicações incluem monitoramento automático de propagandas em rádios. Os anunciantes podem inserir uma marca d'água digital em seus anúncios e contar quantas vezes ela ocorre num período de transmissão, assegurando assim que seus anúncios sejam transmitidos da forma negociada com a emissora [4]. Além

disto, a assinatura digital pode ser incorporada para permitir a autenticação do sinal hospedeiro, como por exemplo em mapas militares e em outras aplicações de segurança.

A técnica de inserção de marca d'água permite o aperfeiçoamento de sistemas de distribuição de áudio, transmitindo simultaneamente um sinal digital através de uma rádio comercial analógica (AM e/ou FM) sem interferir na recepção analógica convencional [1], [7]. Assim, o sinal analógico é o hospedeiro enquanto que o sinal digital é a marca d'água deste sistema híbrido. Já que este incorporamento não degrada o sinal hospedeiro de forma significativa, os receptores analógicos convencionais podem demodular o sinal hospedeiro analógico. Além disto, receptores digitais mais modernos podem decodificar o sinal digital inserido no sinal analógico, que pode representar total ou parcialmente o áudio, permitindo o recebimento de informação sobre a rádio ou o trânsito, por exemplo.

1.1 Contribuições do Trabalho

Este trabalho pretende contribuir para um projeto de colaboração entre o Laboratório de Processamento de Sinais Analógicos e Digitais (PADS), da Universidade Federal do Rio de Janeiro, e a empresa Petrobras.

O projeto trata de um sistema de dimensionamento tridimensional submarino que, analogamente à visão humana, funciona através de um par de imagens. Dentro deste projeto, a inserção de informações sobre imagens do tipo *bitmap* é uma funcionalidade útil. A fim de se transmitirem as imagens sem a necessidade de anexação dos dados de calibração das câmeras em um arquivo texto separado ou em bytes adicionais após o final da imagem, estes dados devem ser inseridos sobre as próprias imagens sem alterar os seus formatos de arquivo (*bitmap*).

Conseqüentemente, este trabalho apresenta e compara dois métodos de inserção de marca d'água digital em arquivos de imagens do tipo *bitmap* com a finalidade de escolher o melhor deles. O primeiro método estudado foi o *Lower Bit Modula-*

tion [8], onde os bits menos significativos da quantização do sinal hospedeiro são substituídos por uma representação binária do sinal marcado. O segundo método abordado foi o *Quantization Index Modulation* [8], onde o sinal hospedeiro é particionado em vetores e conseqüentemente é mapeado para outro conjunto de vetores de acordo com a marca d'água. As implementações dos métodos primeiramente foram feitas com o auxílio de um pacote matemático e, posteriormente, utilizou-se a linguagem de programação C.

O principal parâmetro utilizado para escolher o método a ser empregado no projeto foi a razão sinal-ruído. Além disto, verificou-se a quantidade de informação que cada método suportava e a invisibilidade da marca d'água nas imagens. Diversos testes foram realizados sobre os métodos de inserção de marca d'água digital e as razões sinal-ruído foram comparadas.

1.2 Organização do Documento

O presente trabalho está dividido em cinco capítulos. O Capítulo 2 aborda a teoria utilizada para o desenvolvimento deste trabalho, apresentando seções sobre arquivos de imagens do tipo *bitmap*, sobre os métodos de inserção de marca d'água digital *Lower Bit Modulation* e *Quantization Index Modulation* e sobre a razão sinal-ruído. O Capítulo 3 apresenta os métodos utilizados para o desenvolvimento deste projeto, explicando como se implementa a inserção da marca d'água num *bitmap* de forma imperceptível. O Capítulo 4 apresenta os resultados obtidos levando em consideração a razão sinal-ruído. Por fim, o Capítulo 5 apresenta as conclusões obtidas.

2 Teoria

Este capítulo apresenta a teoria utilizada para o desenvolvimento deste projeto. Primeiramente, realizou-se um estudo sobre o tipo de arquivo de imagem *bitmap*, apresentado na Seção 2.1. Em seguida, as teorias sobre as técnicas de inserção de marca d'água digital *Lower Bit Modulation* e *Quantization Index Modulation* são apresentadas nas Seções 2.2 e 2.3, respectivamente. Na Seção 2.4, introduz-se o conceito de razão sinal-ruído, figura de mérito utilizada para as análises dos métodos de inserção de marca d'água.

2.1 *Bitmap*

Em computação, *bitmap* é um formato de arquivo de imagem usado em diversos sistemas operacionais para representar uma estrutura de dados geralmente em forma de grade retangular de elementos de imagem. Cada elemento de imagem, chamado de *pixel* devido à aglutinação de *Picture Element*, pode ser descrito como um ponto de cor que é visível através de um dispositivo mostrador, como um monitor de computador, um papel etc. A extensão de um arquivo *bitmap* é formada pelos caracteres ".bmp".

Geralmente, o número de bits utilizado para representar a cor de cada *pixel* da imagem é 1 (2 cores possíveis), 4 (16 cores possíveis), 8 (256 cores possíveis), 16 (65.536 cores possíveis) ou 24 (16,7 milhões de cores possíveis). Dependendo do número de cores possíveis na imagem, um *pixel* será armazenado usando $n/8$ bytes, onde n é o número de bits utilizado para representar cada *pixel*, já que um byte (B) é formado por oito bits.

Nas imagens que contêm 24 bits por *pixel*, cada *pixel* possui três componentes, chamadas de RGB: vermelha (*Red*), verde (*Green*) e azul (*Blue*). Sendo assim, cada componente de um *pixel* é representada por um byte, que pode assumir 256 valores inteiros.

Normalmente os arquivos *bitmap* não são comprimidos, o que faz com que eles sejam muito maiores do que imagens comprimidas com o mesmo número de *pixels*, implicando em maior qualidade. Por exemplo, uma imagem de dimensões 800×600 onde utilizam-se 24 bits para representar cada *pixel* ocupa quase 1,4 megabytes (MB).

A Equação (2.1) apresenta o cálculo aproximado do tamanho (T), em bytes, de um arquivo *bitmap* que utiliza n bits para representar cada *pixel*. O número 54 nesta equação é o tamanho do cabeçalho do arquivo *bitmap*, em bytes, que será explicado mais adiante. A altura e a largura são dadas em *pixels*.

$$T \approx 54 + \frac{altura \cdot largura \cdot n}{8} \tag{2.1}$$

No arquivo do tipo *bitmap*, caso o número de *pixels* da dimensão horizontal (numBitsHorizontal) da imagem não seja divisível por quatro, zeros são inseridos após numBitsHorizontal em cada linha para que a nova largura da imagem seja divisível por quatro. O número de zeros inseridos é igual ao valor do resto da divisão de numBitsHorizontal por quatro.

A estrutura de um arquivo *bitmap* é formada por um cabeçalho, que contém informações sobre o arquivo, e pela imagem propriamente dita, que é uma matriz de números inteiros com valores das quantizações das cores dos *pixels*. O cabeçalho possui sempre 54 bytes de informação enquanto que a imagem pode possuir tamanho variável.

A Tabela 1 apresenta os parâmetros que formam o cabeçalho de um arquivo *bitmap* assim como seus tamanhos e suas breves descrições. Os nomes dos parâmetros são auto-explicativos.

Tabela 1: Parâmetros do cabeçalho do *bitmap*

Parâmetro	Descrição	Tamanho (B)
Tipo	Assinatura - 'BM'	2
Tamanho	Tamanho do arquivo em bytes	4
Reservado 1	Vazio	2
Reservado 2	Vazio	2
Offset de Bits	Número de bytes antes do primeiro *pixel*	4
Tamanho da Estrutura	40 bytes	4
Largura	Largura da imagem em *pixels*	4
Altura	Altura da imagem em *pixels*	4
Planos	Número de planos = 1	2
Contador de Bits	Número de bits/*pixel*	2
Compressão	*Flag* de compressão	4
Tamanho da Imagem	Tamanho da imagem em bytes	4
XPixelsPerMeter	Resolução horizontal em *pixel*/metro	4
YPixelsPerMeter	Resolução vertical em *pixel*/metro	4
Cores Usadas	Número de cores usadas	4
Cores Importantes	Número de cores importantes	4

Além disto, a Figura 1 apresenta uma ilustração do espaço de cores RGB enquanto que a Figura 2 apresenta um exemplo ampliado de um *bitmap* onde as porcentagens das componentes vermelha, azul e verde são explicitadas para três *pixels* diferentes.

2.2 *Lower Bit Modulation* (LBM)

O *Lower Bit Modulation* (LBM) é um método de inserção de marca d'água digital onde os bits menos significativos da quantização do sinal hospedeiro são substituídos por uma representação binária do sinal a ser inserido. Cada bit da marca d'água é escrito sobre um *pixel* diferente. Conseqüentemente, a leitura da informação marcada no sinal é feita *pixel* por *pixel*.

A escrita de cada bit da marca d'água digital consiste em igualar o bit menos significativo do valor da quantização do sinal hospedeiro com o bit da marca d'água.

Para realizar a leitura da marca d'água no sinal marcado, basta a verifica-

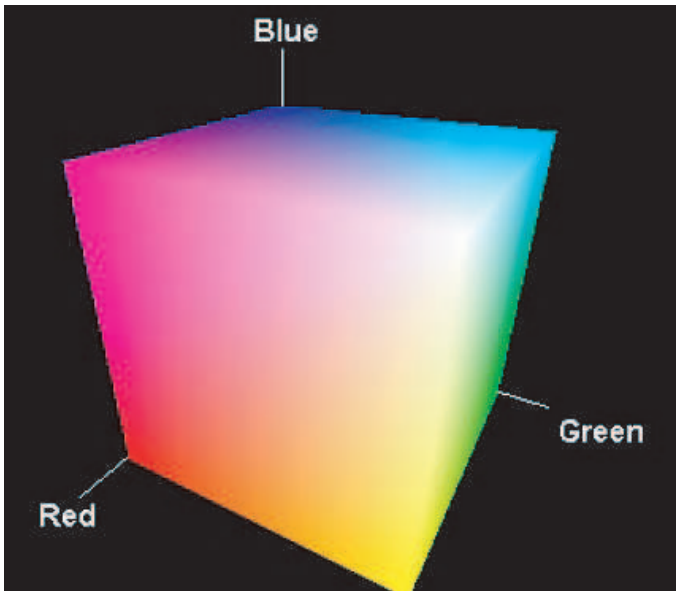


Figura 1: Espaço de cores RGB

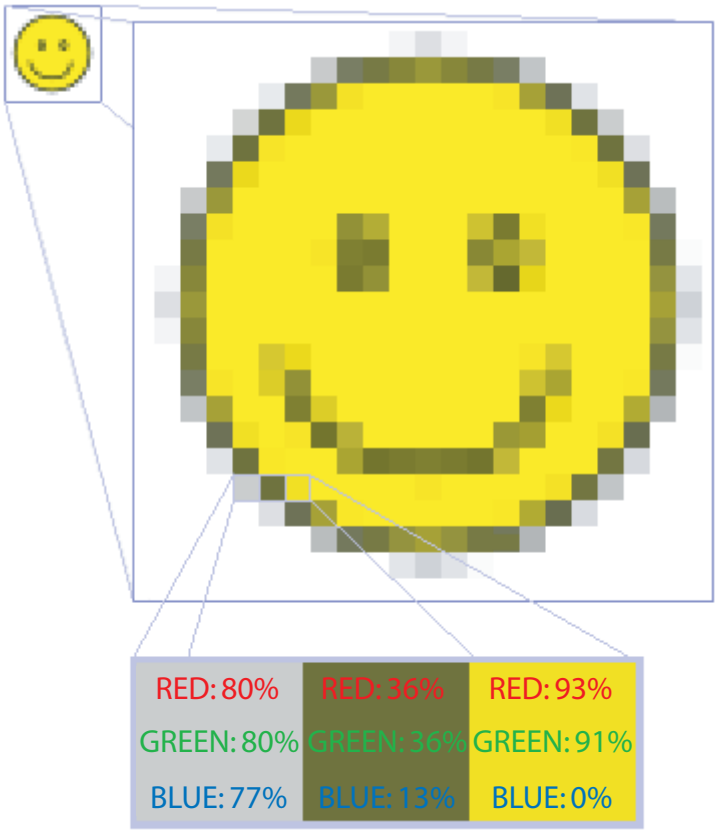


Figura 2: Exemplo de *bitmap*, com destaque para três *pixels* da região esquerda inferior da imagem

ção dos bits menos significativos dos valores das quantizações que formam o sinal marcado. A marca d'água é composta pelos valores destes bits.

2.3 *Quantization Index Modulation* (QIM)

O *Quantization Index Modulation* (QIM) é um método de inserção de marca d'água digital onde o sinal hospedeiro é particionado em blocos (vetores). Posteriormente, os vetores são mapeados para um outro conjunto de vetores de acordo com a marca d'água. O mapeamento geralmente envolve compressão e é feito com o auxílio de uma tabela de consulta. A tabela de consulta contém vetores que representam de forma comprimida os blocos do sinal hospedeiro. A abordagem QIM insere um bit de informação por bloco.

A abordagem mais tradicional do QIM insere cada bit da marca d'água digital através da substituição do bloco do sinal hospedeiro onde a informação do bit será inserida pelo bloco da tabela de consulta que, além de ser referenciado por um índice de mesma paridade do bit a ser inserido, possuir a menor distância euclidiana [1] para o bloco original do sinal hospedeiro.

A geração da tabela de consulta é feita através do *Linde-Buzo-Gray Algorithm* (LBG) [3]. Este algoritmo consiste de iterações que, sucessivamente, calculam o conjunto de vetores da tabela de consulta mais apropriada, dado um conjunto T com M vetores de entrada para treinamento. A partir do conjunto de treinamento T com M vetores, o LBG gera uma tabela de consulta C com N novos vetores chamados de níveis de quantização. A tabela de consulta C também recebe o nome de dicionário ou *codebook*.

A Equação (2.2) apresenta o cálculo do centróide de um conjunto R, onde x_i pertence a R e $\|\mathrm{R}\|$ representa a cardinalidade de R.

[1] A distância euclidiana (d) entre dois vetores x e y de comprimento L é definida por $d = \sqrt{\sum_{i=1}^{L}(x_i - y_i)^2}$.

$$cent(\mathrm{R}) = \frac{1}{\|\mathrm{R}\|} \sum_{i=1}^{\|\mathrm{R}\|} x_i \tag{2.2}$$

- Seguem os passos do algoritmo LBG:

1) Começar com um *codebook* (conjunto de células de quantização definidas pelas seus centróides $y_i, i = 1, \ldots, N$) inicial C_1;

2) Repetir:

2a) Dado um dicionário $\mathrm{C}_k = \{y_i; i = 1, \ldots, N\}$, associar cada vetor do conjunto de treinamento a um dos centróides em C_k através da seleção do centróide que está mais perto do vetor de treinamento;

2b) Recalcular o centróide y_i para cada aglomerado i ($i = 1, \ldots, N$) criado no passo 2a), para obter o novo dicionário C_{k+1};

2c) Caso uma célula vazia tenha sido criada no passo anterior, o centróide da célula com o maior número de vetores associados é dividido em dois novos centróides;

2d) Calcular a distorção média associada a C_{k+1}, e definida por $\frac{1}{M}\sum_{i=1}^{M} d_i^2$, até que sua diferença em relação à distorção média da iteração anterior seja pequena o suficiente.

Este trabalho busca inserir marca d'água digital em imagens do tipo *bitmap*, que permitem apenas níveis de quantização formados por números inteiros para representar os *pixels* da imagem. Porém, como a tabela de consulta gerada pelo LBG possui vetores cujas componentes não são necessariamente valores inteiros, a substituição de blocos por vetores selecionados do dicionário conforme a paridade da marca d'água não apresentou bons resultados na prática. A tabela de consulta com arredondamentos para números inteiros durante a execução do LBG não foi gerada, porém uma nova abordagem foi proposta.

A nova abordagem QIM primeiramente divide a marca d'água em blocos de mesmo tamanho, formados por b bits cada. Em seguida, a informação referente

a cada bloco de bits é inserida em um bloco diferente do sinal hospedeiro. Cada bloco de bits de informação está inserido num bloco do sinal hospedeiro se o valor do somatório dos elementos deste bloco for divisível pelo número formado pela representação decimal associada ao bloco de bits de informação inserido. Para que isto ocorra, alteram-se os valores do bloco do sinal hospedeiro, fazendo a sua soma conter a informação do bloco de bits inserido.

Considerando uma imagem do tipo *bitmap*, o valor x a ser somado ou subtraído num bloco da imagem para que um bloco de bits da marca d'água seja escrito é dado na Equação (2.3), onde a operação $mod(c, d)$ retorna o resto de c/d, b é o número de bits a ser inserido por bloco, r é o resto da divisão da soma dos *pixels* do bloco por 2^b e num é o número representado pelo conjunto de bits na base decimal. Caso $x > 2^{b-1}$, deve-se ainda fazer $x = x - 2^b$. Em posse de x, ele deve ser adicionado ao somatório dos *pixels* do bloco em análise através de adições ou subtrações unitárias sobre x *pixels* diferentes aleatoriamente escolhidos, minimizando-se assim o erro quadrático devido à alteração dos *pixels* do bloco.

$$x = mod((2^b - r + num), 2^b) \tag{2.3}$$

Para realizar a leitura da marca d'água, deve-se saber b, isto é, quantos bits de informação estão inseridos por bloco do sinal hospedeiro. Conseqüentemente, calcula-se o resto da divisão do somatório dos *pixels* de cada partição por 2^b e em seguida converte-se este resultado para representação binária utilizando-se b bits. Desta forma, recupera-se cada conjunto de bits inserido no sinal hospedeiro.

2.4 Razão Sinal-Ruído

A razão sinal-ruído é um parâmetro que relaciona um sinal de referência com este mesmo sinal acrescido de ruído. O ruído afeta a fidelidade da representação do sinal original. Como diversos sinais apresentam faixas dinâmicas vastas, a razão sinal-ruído normalmente é expressa em decibel.

O decibel (dB) é uma medida adimensional da razão entre duas quantidades expressa por: $X_{dB} = 10 * \log_{10}\left(\frac{X}{X_0}\right)$, onde X é o valor da quantidade que está sendo medida, X_0 é nível de referência e X_{dB} é a quantidade expressa em decibéis, relativa ao nível X_0.

Neste trabalho a razão sinal-ruído é freqüentemente referenciada pelo acrônimo PSNR, que significa *Peak Signal-to-Noise Ratio*. A PSNR é definida a partir do erro médio quadrático, referenciado pelo acrônimo MSE, de *Mean Squared Error*. Sejam I e K duas imagens monocromáticas de dimensões $P \times Q$ onde uma delas é uma aproximação ruidosa da outra. Define-se o MSE como apresentado na Equação (2.4).

$$\text{MSE} = \frac{1}{PQ} \sum_{i=0}^{P-1} \sum_{j=0}^{Q-1} \|I(i,j) - K(i,j)\|^2 \tag{2.4}$$

Segue a definição da PSNR na Equação (2.5), onde n é o número de bits usado para representar cada *pixel* da imagem. Em todas as imagens utilizadas neste trabalho, cada componente de um *pixel* é representada por $n = 8$ bits, ou seja, $2^n - 1 = 255$.

$$\text{PSNR} = 10 * \log_{10}\left(\frac{(2^n - 1)^2}{\text{MSE}}\right) \tag{2.5}$$

Para imagens policromáticas, com três componentes (RGB) por *pixel*, a definição de PSNR é a mesma com a exceção de que o MSE é o somatório dos quadrados das diferenças dividido pelo tamanho da imagem e por três.

A PSNR também pode ser entendida como uma medida de similaridade através de um cálculo sobre as diferenças entre os *pixels* equivalentes da imagem referencial e da imagem ruidosa. Nota-se da Equação (2.5) que o valor da PSNR varia entre zero, quando não há similaridade entre as imagens, ou seja, todas as diferenças são iguais a 255, e infinito, quando as imagens são idênticas. Como esperado, a PSNR aumenta à medida que a quantidade de ruído sobre a imagem em análise diminui.

O *Mean Opinion Score* (MOS), que é um método subjetivo de teste de qualidade, não foi utilizado como figura de mérito para escolha do melhor método de inserção de marca d'água digital devido à dificuldade de comparação visual entre as imagens marcadas.

3 Métodos

Este capítulo apresenta as metodologias utilizadas para o desenvolvimento deste trabalho, em que o objetivo é inserir uma marca d'água digital num *bitmap*. Primeiramente houve um estudo sobre o formato digital de imagem *bitmap*, explicado detalhadamente na Seção 2.1. Em seguida, dois métodos de inserção de marca d'água digital foram estudados, realizando-se simulações de escrita e leitura de informação através de um pacote matemático. Finalmente, os dois métodos foram implementados na linguagem de programação C e testados. Os dois métodos são comparados no Capítulo 4.

Diversos métodos de inserção de marca d'água têm sido propostos na literatura [5], [8], [9]. Neste trabalho dois métodos foram abordados: o *Lower Bit Modulation*, apresentado na Seção 3.1, e o *Quantization Index Modulation*, apresentado na Seção 3.2. Em ambas abordagens a marca d'água foi escrita apenas sobre a componente azul da imagem, pois ela é a menos visível.

As rotinas de escrita de marca d'água de ambos métodos possuem três parâmetros de entrada: o nome do arquivo *bitmap* da imagem hospedeira, o nome do arquivo texto contendo uma chave binária e o nome do arquivo texto contendo a mensagem a ser escrita. No final da execução da rotina de escrita de qualquer um dos dois métodos, gera-se a imagem marcada, que é composta pela imagem hospedeira marcada pela chave e pela mensagem. Em posse da imagem original, ou hospedeira, e da imagem marcada, calcula-se a razão sinal-ruído (PSNR) entre ambas. A PSNR é explicada na Seção 2.4 e permite saber o quão ruidosa é a imagem marcada em relação à imagem hospedeira não marcada. Utilizou-se este parâmetro

para comparação de resultados entre as implementações.

Já as rotinas de leitura da marca d'água inserida num *bitmap* possuem dois parâmetros de entrada: o nome do arquivo *bitmap* da imagem marcada e o nome do arquivo texto contendo a chave binária que foi utilizada para gerar a imagem marcada. Estas rotinas geram um arquivo texto contendo a marca d'água (mensagem) inserida na imagem marcada.

As Figuras 3(a) e 3(b) apresentam os diagramas de blocos para escrita e leitura da marca d'água, respectivamente.

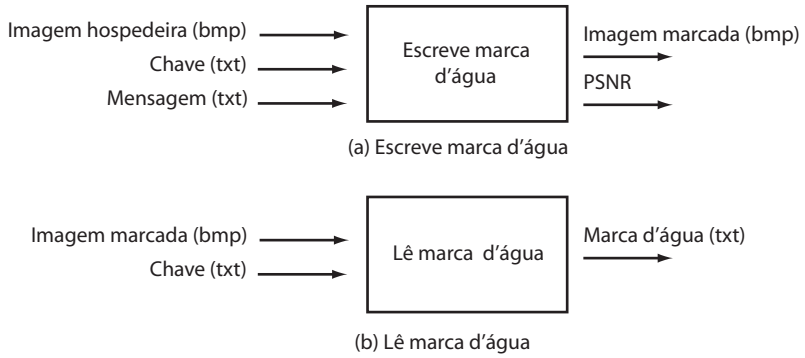


Figura 3: Diagramas de bloco

Com a finalidade de verificar se uma imagem carrega uma mensagem, as rotinas de escrita dos dois métodos inserem uma chave de 512 bits antes da mensagem. Assim, as rotinas de leitura da marca d'água primeiramente fazem a validação da chave. Caso a chave esteja correta, a mensagem é finalmente lida, caso contrário, nada é feito.

Além disto, as rotinas de escrita de marca d'água inserem logo após a chave um conjunto de vinte bits representando o tamanho da mensagem a ser inserida. Isto é feito para que as rotinas de leitura saibam qual o tamanho da mensagem que será lida. Depois da chave e do conjunto de vinte bits serem inseridos no *bitmap*, a mensagem é finalmente inserida através da representação binária do código ASCII [1] de cada caractere que a compõe utilizando-se 8 bits para cada caractere.

[1] ASCII (acrônimo para *American Standard Code for Information Interchange*) é uma padronização da indústria de computadores, onde cada caractere é manipulado na memória, discos etc., sob forma de código binário. O código ASCII é formado por todas as combinações possíveis de 8 bits.

A Figura 4 apresenta o fluxograma que descreve a rotina de escrita da marca d'água digital na imagem hospedeira. O fluxograma que descreve a rotina de leitura é apresentado na Figura 5.

3.1 *Lower Bit Modulation* (LBM)

Esta seção apresenta a metodologia utilizada para a implementação do método LBM. Cada bit da chave ou da mensagem é escrito sobre um *pixel* diferente. Conseqüentemente, a leitura da informação marcada na imagem será feita *pixel* por *pixel*.

A escrita da marca d'água digital na imagem consiste em pegar cada bit da informação, seja ele da chave ou da mensagem, e compará-lo com o resto da divisão por dois (paridade) do valor da componente azul do *pixel* onde pretende-se inserir a informação do bit em questão. Objetiva-se que a paridade do *pixel* fique igual à paridade do bit de informação.

Sendo assim, caso tanto a paridade do *pixel* quanto o bit em análise tenham o mesmo valor, nada é preciso ser feito, pois o bit em análise já estará inserido na imagem. Caso contrário, soma-se um ao *pixel* em questão para ele mudar de paridade e conter a informação do bit em análise. Se esta soma saturar o *pixel*, deve-se desfazê-la e, em seguida, subtrair-se um do valor do *pixel*.

Como exemplo, se o valor da componente azul de um *pixel* for 180 e o bit da marca d'água a ser inserido valer 0, nada é preciso ser feito pois o bit menos significativo da representação binária do número 180 é 0, ou seja, o *pixel* de valor 180 já carrega a informação de um bit 0 da marca d'água. Caso o bit da marca d'água valesse 1, far-se-ia necessário adicionar um ao *pixel* para que seu último bit na representação binária mudasse para 1.

Para realizar a leitura da marca d'água na imagem marcada, basta verificar a paridade de cada *pixel* da chave, do conjunto de vinte bits e da mensagem. Um *pixel* com valor par carrega a informação de um bit 0, enquanto que um *pixel* com

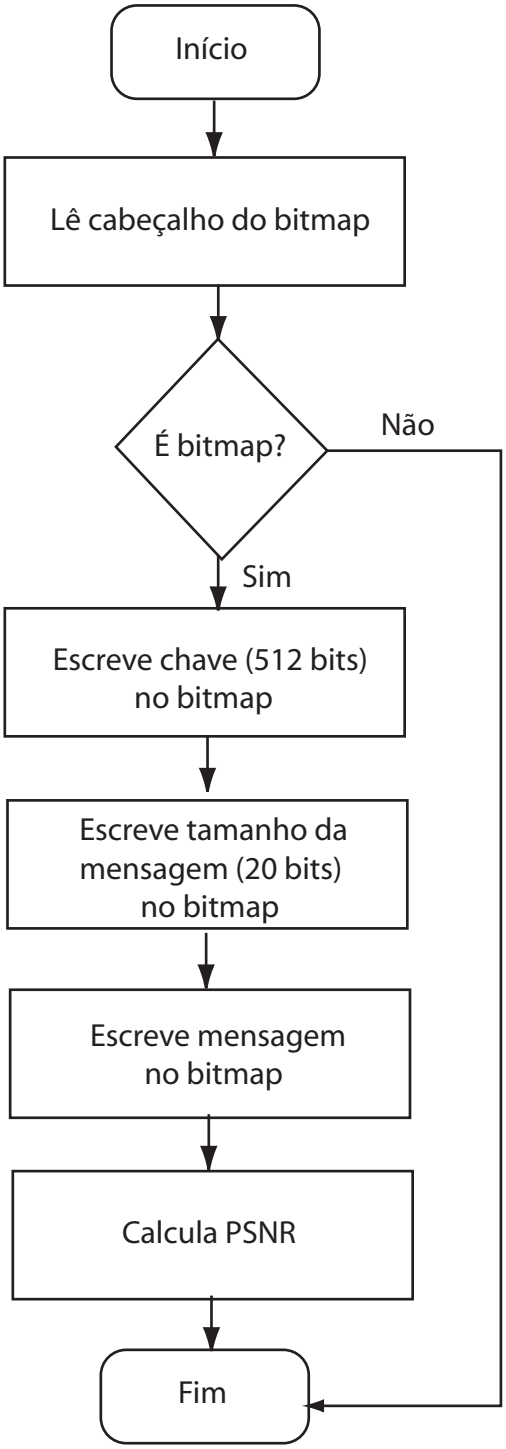


Figura 4: Fluxograma que descreve a escrita da marca d'água

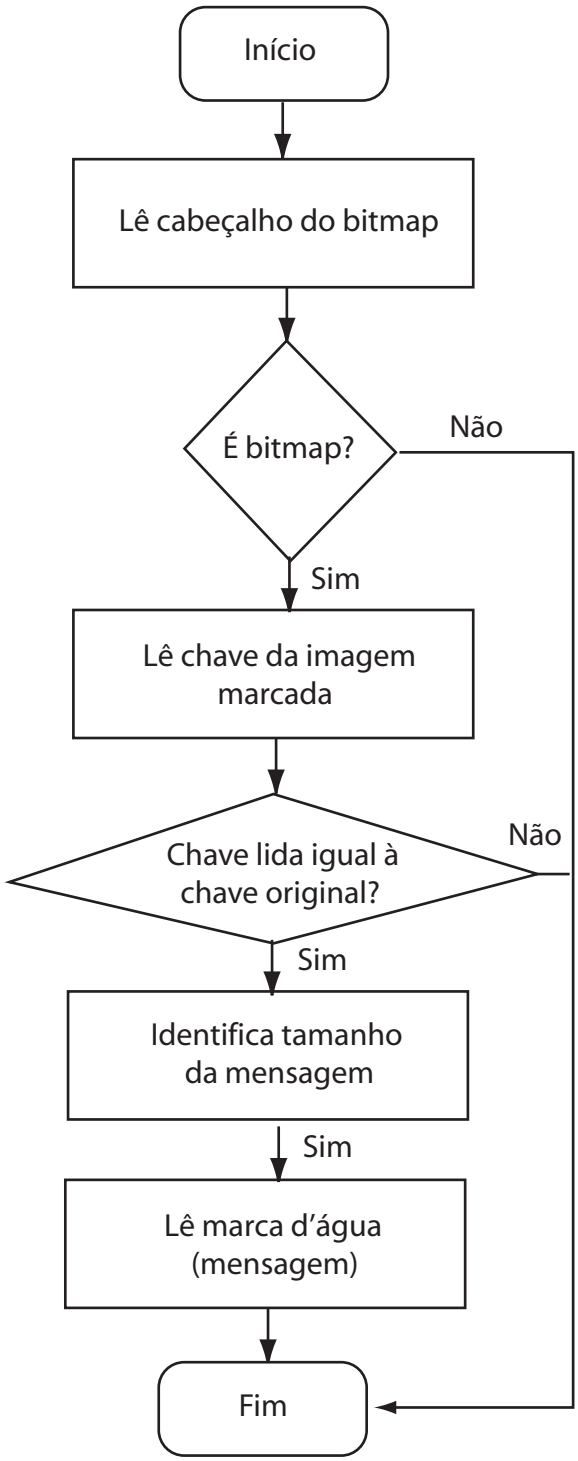


Figura 5: Fluxograma que descreve a leitura da marca d'água

valor ímpar carrega a informação de um bit 1.

Os códigos de escrita e de leitura da marca d'água implementados em linguagem C encontram-se nos Apêndices A e B, respectivamente.

3.2 *Quantization Index Modulation* (QIM)

Esta seção apresenta a metodologia utilizada para a implementação do método QIM [3], [8]. A idéia inicial desta abordagem trata de dividir a imagem hospedeira em um conjunto de blocos e em seguida mapear o conjunto de blocos para outro conjunto de acordo com a marca d'água. Neste trabalho, cada bit de informação é escrito sobre um bloco diferente de 4×4 *pixels*. Conseqüentemente, a leitura da informação marcada da imagem será feita por bloco 4×4.

A primeira implementação deste método utilizou o *Linde-Buzo-Gray Algorithm* (LBG) para criar uma tabela de consulta a partir de vetores de treinamento. Citado na Seção 2.3, o LBG começa com cada bloco 4×4 da imagem hospedeira tendo sido transformado em vetor 16×1. Após isto, o algoritmo gera uma tabela de consulta com 256 vetores 16×1 de tal forma que eles representem, de forma comprimida, os blocos da imagem hospedeira. A tabela é ordenada conforme a primeira componente de cada vetor. Em seguida, subdivide-se esta tabela em duas partes de acordo com a paridade do índice de cada vetor, ou seja, os vetores com índices ímpares vão para uma tabela de índices ímpares chamada de parte 1, enquanto que os vetores com índices pares vão para a tabela de índices pares chamada de parte 0. A rotina de escrita de marca d'água se baseia na tabela de consulta, previamente dividida em parte 0 e parte 1, para inserir a mensagem na imagem *bitmap*.

Para cada bit da marca d'água que será inserido na imagem, o algoritmo seleciona consecutivamente um vetor 16×1 da imagem hospedeira onde a informação deste bit será adicionada. Caso o bit da marca d'água valha 0, a rotina de escrita procura na tabela de consulta de índices pares o vetor que tem a menor distância euclidiana para o vetor 16×1 originado da imagem hospedeira. Depois disto, este

vetor que possui a menor distância euclidiana volta a ter dimensão 4×4 e é substituído na imagem hospedeira, carregando consigo a informação do bit da marca d'água. Caso um bit da marca d'água valha 1, o mesmo procedimento é realizado sobre a tabela de índices ímpares.

A rotina de leitura da marca d'água primeiro divide a imagem em blocos de dimensão 4×4 e depois os reorganiza para vetores 16×1. Em seguida, os vetores 16×1 da imagem marcada são comparados com os vetores das tabelas de consulta até que um vetor igual seja encontrado. Caso o vetor encontrado pertença à tabela de consulta de índices pares, sabe-se que o bit de informação carregado neste bloco vale 0. Caso o vetor encontrado pertença à tabela de consulta de índices ímpares, o bit de informação vale 1.

A implementação citada acima não forneceu resultados satisfatórios pois a tabela de consulta gerada pelo LBG tem precisão infinita enquanto que o arquivo *bitmap* salva apenas valores inteiros para quantizar os seus *pixels*. Para contornar este problema, tentou-se arredondar os valores da tabela de consulta tornando-os valores inteiros e, posteriormente, remover vetores que eventualmente ficassem iguais, o que causou distorção excessivamente alta e grande número de vetores iguais.

Conseqüentemente, partiu-se para uma nova abordagem QIM subdividida em quatro casos em função da quantidade de bits (1, 2, 3 ou 4) de informação que são inseridos por bloco 4×4 da imagem hospedeira. A chave, o conjunto de vinte bits com o tamanho da marca d'água e a própria marca d'água são particionados em conjuntos de b bits, $b = 1$, 2, 3 ou 4, para que cada conjunto seja inserido em um bloco 4×4 consecutivo da imagem hospedeira, gerando assim a imagem marcada.

Um conjunto de bits de informação é inserido num bloco 4×4 da imagem hospedeira quando o resto da divisão da soma dos valores de todos os *pixels* do bloco pela quantidade de permutações que se pode formar com os b bits do conjunto a ser inserido fica igual ao número formado pela representação decimal destes b bits. Para esta igualdade ocorrer, alteram-se os valores dos *pixels* do bloco 4×4, fazendo-se com que a informação referente ao conjunto de bits da marca d'água em análise seja

inserida. A quantidade de permutações que se pode formar com um conjunto de b bits é 2^b.

No primeiro caso, onde se insere um bit por bloco 4×4 da imagem hospedeira, primeiramente calcula-se a soma dos valores dos *pixels* do bloco onde este bit será escrito. Em seguida, verifica-se o resto da divisão desta soma por dois, que pode ser 0 ou 1. Caso o resto seja igual ao bit em questão, nada é preciso ser feito. Caso contrário, soma-se 1 a um *pixel* aleatório deste bloco 4×4, o que neste caso simplesmente faz a soma dos *pixels* do bloco mudar de paridade e carregar a informação do bit.

No segundo caso, onde se inserem dois bits por bloco 4×4, após calcular-se a soma dos valores dos *pixels* do bloco onde este par de bits será escrito, deve-se realizar a divisão deste valor por 4 e guardar o resto. Caso o resto seja igual ao número decimal representado pelo par de bits em questão, que pode ser 0, 1, 2 ou 3, nada é preciso ser feito. Caso, por exemplo, o resto valha 2 e o par de bits seja 11, representando o número 3, deve-se somar 1 a um *pixel* aleatório no bloco 4×4 em questão para que a informação referente a este par de bits seja inserida na imagem.

A generalização é feita como foi mostrado na Seção 2.3: o valor x a ser somado ou subtraído num bloco 4×4 da imagem para que um conjunto de bits da marca d'água seja escrito é dado pela Equação 2.3. Este valor deve ser adicionado ao somatório dos *pixels* do bloco em análise através de adições ou subtrações unitárias sobre *pixels* diferentes aleatoriamente escolhidos. Por exemplo, se $x = -2$, sorteia-se um *pixel* do bloco 4×4 e realiza-se a subtração de uma unidade do valor dele. Em seguida, sorteia-se outro *pixel* e realiza-se a segunda operação de subtração de uma unidade do valor dele. Conseqüentemente, subtraiu-se 2 do somatório dos *pixels* deste bloco e o conjunto de bits da marca d'água foi inserido na imagem.

Para realizar a leitura da marca d'água, deve-se saber quantos bits de informação estão inseridos por bloco da imagem marcada. Calcula-se o resto da divisão da soma dos *pixels* de cada bloco por 2^b, onde b é o número de bits que foi escrito por bloco, e em seguida converte-se este resultado para binário utilizando-se b bits na

representação. Desta forma, recupera-se cada conjunto de b bits inserido na imagem marcada e consegue-se reconstruir a mensagem através do código ASCII formado pelos agrupamentos consecutivos de oito bits.

4 Resultados

Este capítulo apresenta os resultados obtidos através dos métodos LBM e QIM, explicados repectivamente nas Seções 3.1 e 3.2, para inserção de marca d'água em imagens do tipo *bitmap*. Foram realizadas quatro abordagens sobre o método QIM.

Primeiramente foi gerada uma chave binária com 512 bits gravada em um arquivo texto. Além disto, criou-se um conjunto de onze arquivos texto compostos por caracteres aleatórios que simulam as mensagens a serem escritas na imagem hospedeira. As quantidades de caracteres em cada arquivo são dadas por valores que formam uma escala aproximadamente logarítmica, composta por 10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000 e 20000 caracteres.

Os algoritmos LBM e QIM foram executados sobre a chave seguida de cada um dos arquivos que simulam as mensagens e os valores da PSNR foram comparados. Os resultados apresentados nas Seções 4.1 e 4.2 foram gerados com a componente azul da imagem hospedeira lena.bmp. Esta imagem é apresentada na Figura 6. Ela tem dimensão 512×512 e cada *pixel* é representado por 24 bits.

A Seção 4.3 apresenta os resultados da inserção das informações de calibração nas imagens geradas pelo par de câmeras utilizadas no projeto.

4.1 *Lower Bit Modulation* (LBM)

A Tabela 2 apresenta os resultados obtidos através deste método. Nota-se que à medida que o número de caracteres dos arquivos texto aumenta, o valor da

Figura 6: Imagem hospedeira lena.bmp

Tabela 2: Resultados obtidos através do método *Lower Bit Modulation*

Número de Caracteres	PSNR (dB)
10	82,23
20	81,70
50	80,42
100	78,83
200	76,75
500	73,38
1000	70,73
2000	67,91
5000	64,04
10000	61,04
20000	58,03

PSNR diminui, indicando uma maior degradação da imagem marcada.

A Figura 7 mostra o gráfico referente à Tabela 2, de onde percebe-se o comportamento aproximadamente linear da PSNR em relação ao logaritmo do número de caracteres da marca d'água na medida em que este número cresce.

Todas as marcas d'água ficaram imperceptíveis ao olho humano como mostrado nas Figuras 8(a), 8(b), 8(c) e 8(d), que permitem a comparação entre a imagem hospedeira lena.bmp sem marca d'água e as imagens marcadas com mensagens de 10, 500 e 20000 caracteres, respectivamente.

As imagens marcadas foram subtraídas da imagem hospedeira original para observar-se onde houve degradações nas imagens. Caso um *pixel* não tenha sido alterado, sua subtração do *pixel* equivalente na imagem hospedeira não marcada origina um novo *pixel* de valor zero, que visualmente é preto. Como o trabalho foi desenvolvido apenas sobre a componente azul, não houve qualquer tipo de degradação nas componentes verde e vermelha da imagem hospedeira. A componente azul teve alguns *pixels* alterados, implicando em valores diferentes de zero no resultado da subtração da imagem hospedeira não marcada.

Tais alterações podem ser vistas nos pontos azuis da imagem apresentada na Figura 9, que apresenta o resultado da subtração entre a imagem marcada com

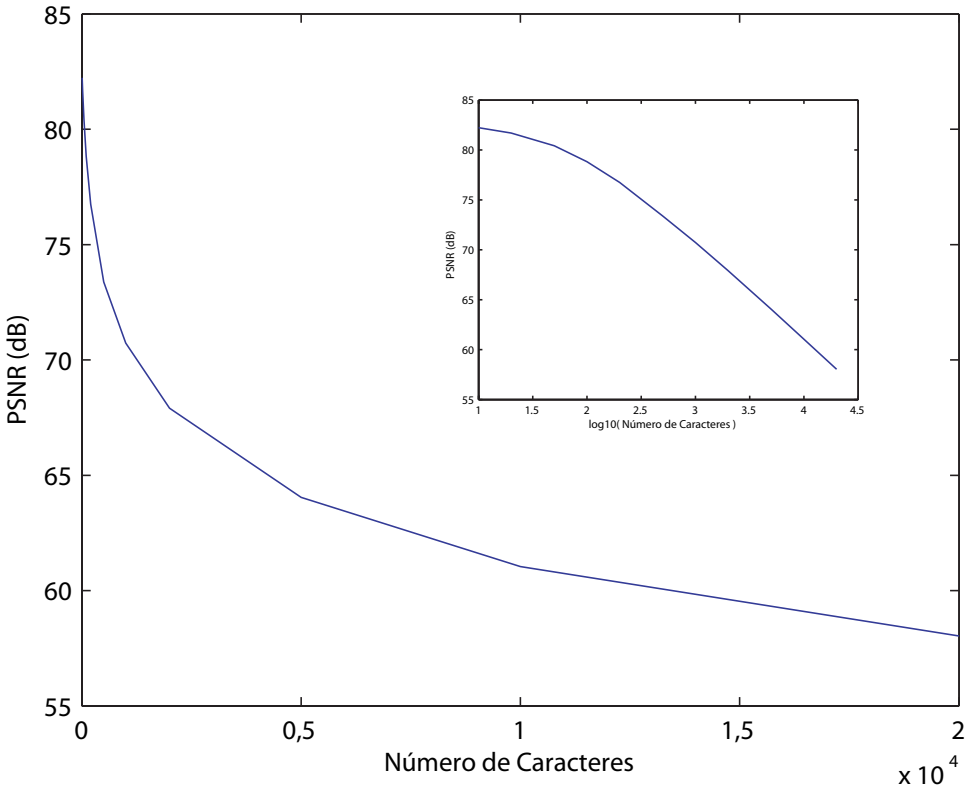


Figura 7: Performance do método *Lower Bit Modulation* (Tabela 2)

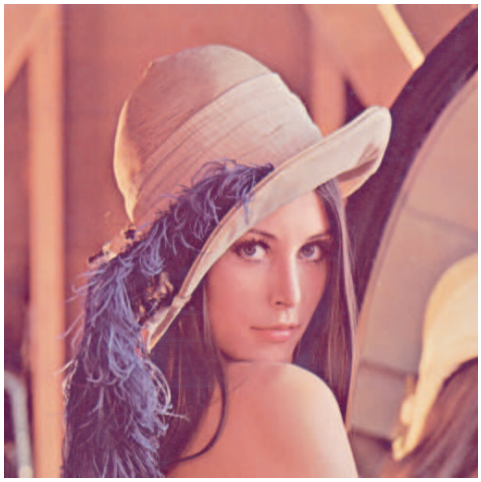
uma mensagem de 20000 caracteres e a imagem hospedeira original. O resultado da subtração foi multiplicado por 255. O valor 255 serve para que os pontos onde houve alteração apareçam como pontos azuis na Figura 9.

4.2 *Quantization Index Modulation* (QIM)

O método QIM foi dividido em quatro abordagens. Cada uma delas insere 1, 2, 3 ou 4 bits de informação por bloco 4×4 na imagem hospedeira. Os resultados das abordagens encontram-se nas Seções 4.2.1 (1 bit) e 4.2.2 (2, 3 ou 4 bits).

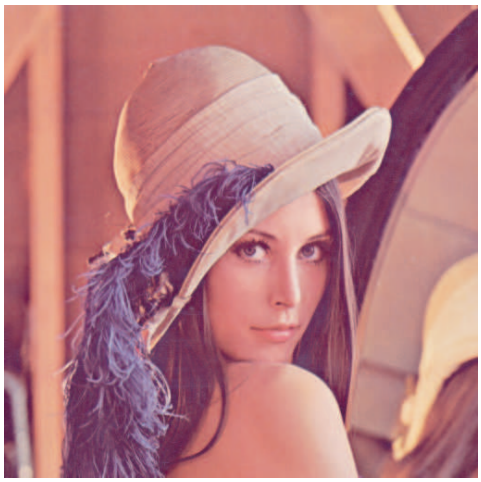
4.2.1 Inserção de 1 Bit por Bloco

Os resultados em que 1 bit de informação foi inserido em cada bloco 4×4 da lena.bmp estão apresentados na Tabela 3. Nota-se que a capacidade de inserção da marca d'água nesta abordagem é menor do que a capacidade do método LBM,

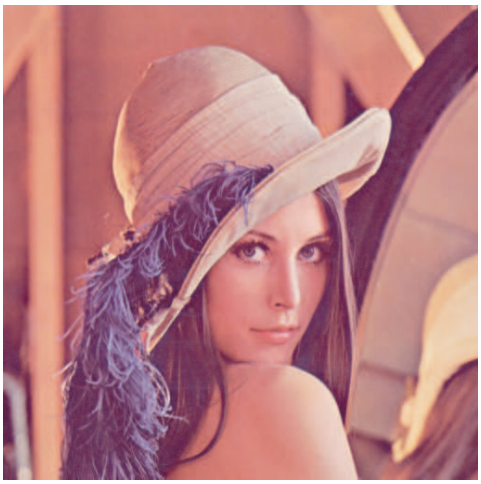


(a) Imagem hospedeira: lena.bmp

(b) Imagem marcada com mensagem de 10 caracteres



(c) Imagem marcada com mensagem de 500 caracteres



(d) Imagem marcada com mensagem de 20000 caracteres

Figura 8: Comparação das imagens marcadas pelo método *Lower Bit Modulation*

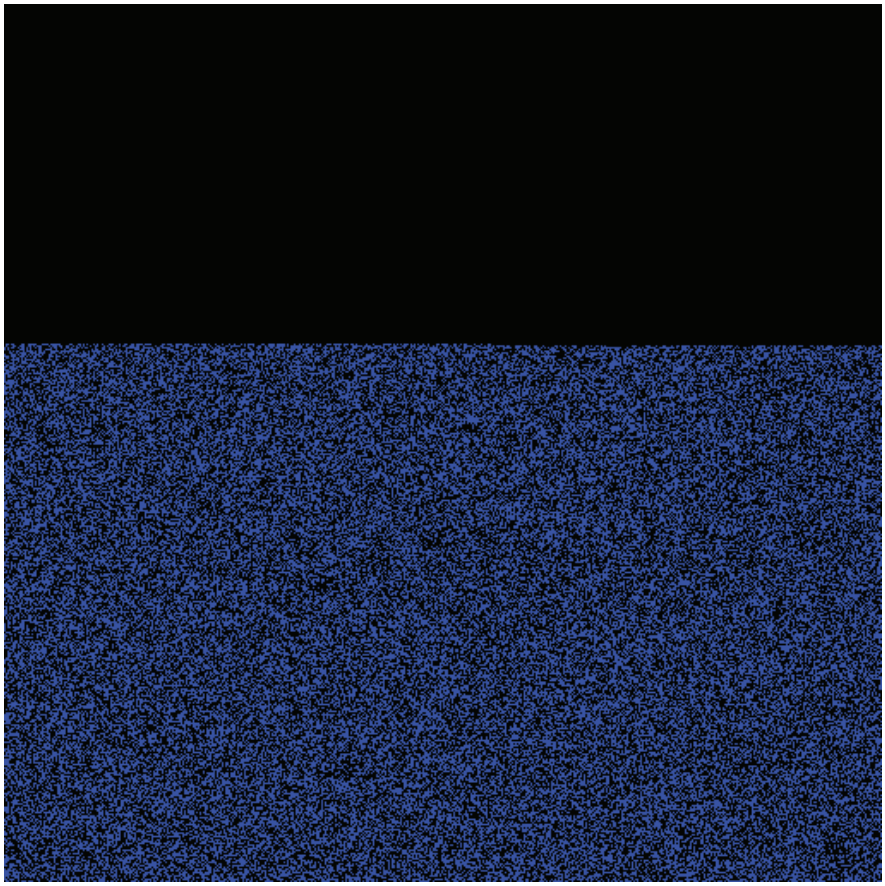


Figura 9: Subtração entre a imagem marcada com mensagem de 20000 caracteres e a imagem hospedeira original

porque apenas 1 bit é inserido a cada 16 *pixels*. Por isto, testes foram realizados apenas com os arquivos que continham até 2000 caracteres.

Tal perda de capacidade ocorre pois agora cada bloco 4×4 da imagem hospedeira, que tem dimensão 512×512, poderá conter somente a informação de 1 bit, ou seja, pode-se incluir no máximo $\frac{512*512}{4*4} = 16384$ bits de informação. Como exemplo, o arquivo texto que simula uma mensagem de 5000 caracteres contém $5000*8$ bits $= 40000$ bits, excedendo a capacidade máxima de uma imagem 512×512 nesta abordagem.

Tabela 3: Resultados obtidos através do método *Quantization Index Modulation* - 1 bit por bloco

Número de Caracteres	PSNR (dB)
10	82,30
20	81,80
50	80,50
100	78,89
200	76,80
500	73,47
1000	70,81
2000	67,91

A Figura 10 apresenta a comparação dos resultados desta abordagem, mostrados na Tabela 3, com os do método LBM, mostrados na Tabela 2. Nota-se que a curva da abordagem QIM: 1 bit encontra-se muito pouco acima da curva do método LBM, indicando que ela degrada a imagem hospedeira da mesma forma que o método LBM. Apesar disto, o método LBM possui uma capacidade consideravelmente (16 vezes) maior de inserção de informação na imagem hospedeira, $512 * 512 = 262144$ bits, em contrapartida aos 16384 bits desta abordagem QIM.

Sendo assim, o método LBM foi considerado melhor do que esta abordagem do método QIM.

Visualmente, as marcas d'água QIM: 1 bit ficaram imperceptíveis ao olho humano como está mostrado nas Figuras 11(a), 11(b), 11(c) e 11(d), que permitem a comparação entre a imagem hospedeira lena.bmp sem marca d'água e as imagens

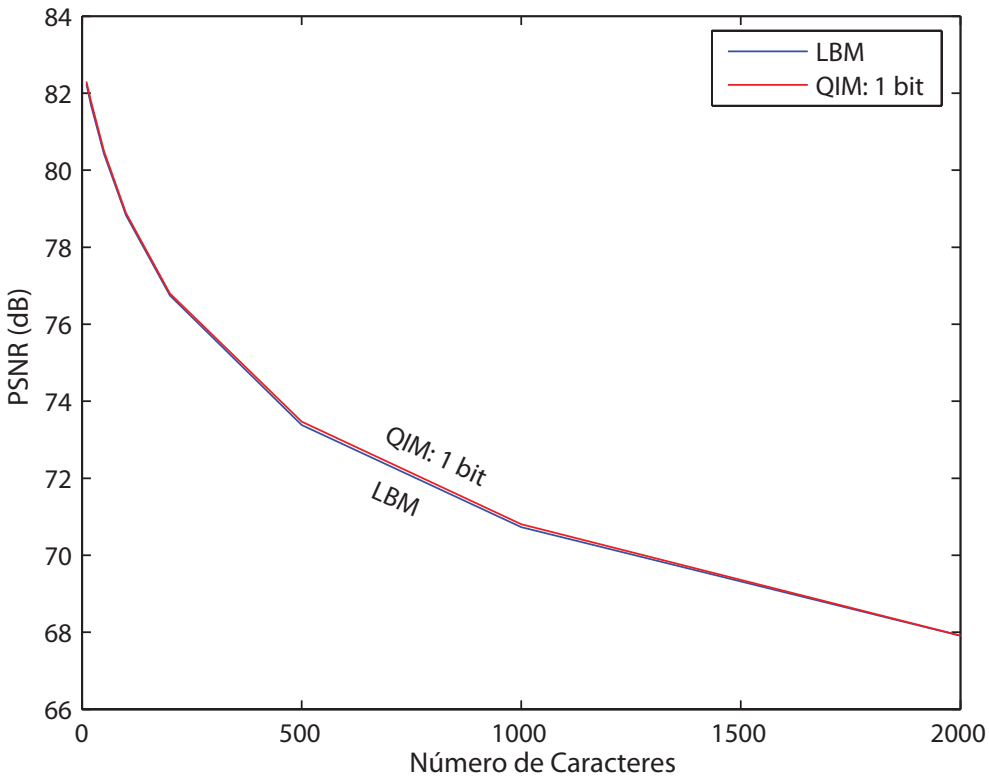


Figura 10: Comparação entre os métodos LBM (Tabela 2) e QIM: 1 bit (Tabela 3) marcadas com mensagens de 10, 100 e 2000 caracteres, respectivamente.

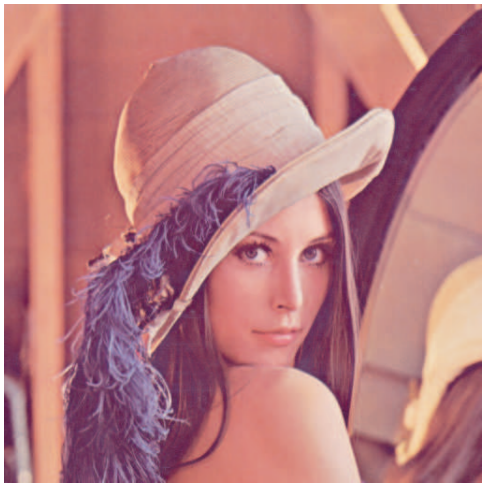
4.2.2 Inserção de 2, 3 ou 4 Bits por Bloco

Esta subseção apresenta os resultados do método QIM onde cada dois, três ou quatro bits de informação são incorporados por bloco 4×4 da imagem hospedeira. A Tabela 4 apresenta as razões sinal-ruído em função dos números de caracteres das mensagens. Percebe-se que à medida que se aumenta o número de bits de informação inseridos por bloco da lena.bmp, os valores da PSNR diminuem consideravelmente, ou seja, a imagem marcada fica mais degradada.

O gráfico apresentado na Figura 12 compara os desempenhos dos resultados apresentados na Tabela 4 com os resultados mostrados anteriormente nas Tabelas 2 e 3. Infere-se que o método LBM e a abordagem QIM com inserção de 1 bit por bloco apresentam as melhores razões sinal-ruído, indicando menor degradação da imagem hospedeira. Além disto, o método LBM possui maior capacidade de inserção de

(a) Imagem hospedeira: lena.bmp

(b) Imagem marcada com mensagem de 10 caracteres



(c) Imagem marcada com mensagem de 100 caracteres

(d) Imagem marcada com mensagem de 2000 caracteres

Figura 11: Comparação das imagens marcadas pelo método *Quantization Index Modulation*: 1 bit

Tabela 4: Resultados obtidos através do método *Quantization Index Modulation* - 2, 3 e 4 bits por bloco

Número de Caracteres	PSNR (dB)		
	2 bits por bloco	3 bits por bloco	4 bits por bloco
10	80,51	78,19	74,91
20	80,12	77,85	74,18
50	78,78	76,46	73,18
100	77,12	74,78	71,24
200	75,00	72,99	69,55
500	71,66	69,52	65,98
1000	68,95	66,84	63,39
2000	66,11	63,91	60,36
5000	62,13	60,00	56,51

informação, como citado anteriormente. Esses resultados confirmaram a conclusão da Seção 4.2.1: o método LBM funciona melhor que a abordagem QIM na forma aqui proposta, para a inserção de marca d'água digital em arquivos tipo *bitmap*.

O pior valor de PSNR encontrado foi de 56,51 dB e mesmo assim a degradação sobre a imagem hospedeira ficou imperceptível. As Figuras 13(a) e 13(b) apresentam a comparação do *zoom* de 16 vezes entre as partes equivalentes da imagem hospedeira lena.bmp e da imagem marcada com PSNR igual a 56,51 dB, respectivamente.

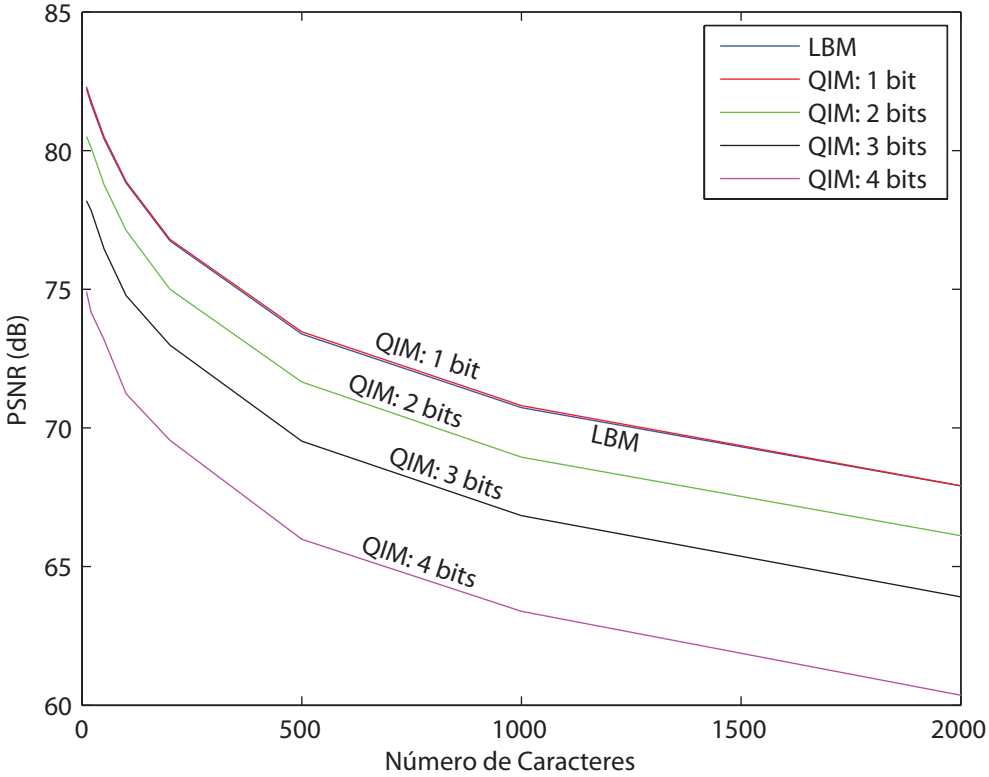
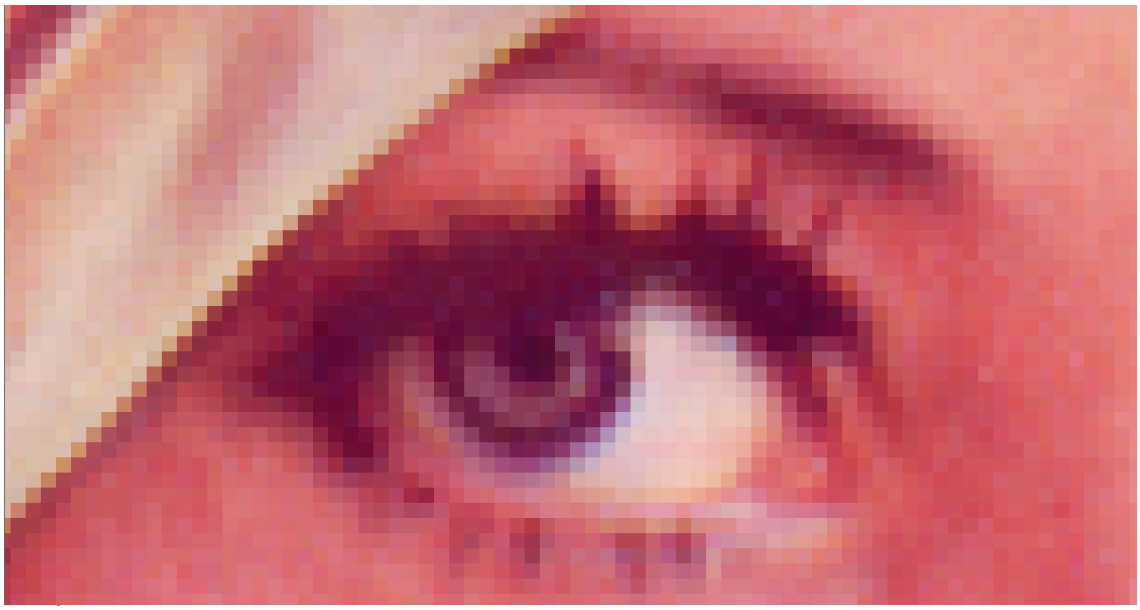


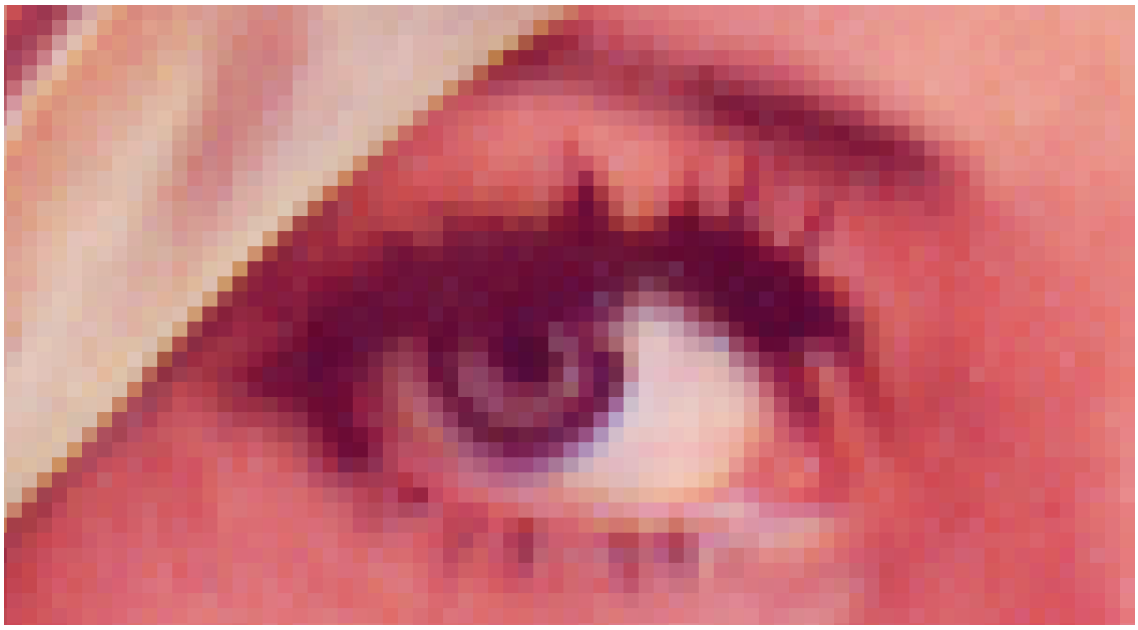
Figura 12: Comparação entre os métodos LBM (Tabela 2) e QIM (Tabelas 3 e 4)

4.3 Marca d'Água nas Imagens de Calibração

Esta seção apresenta os resultados da inserção das informações de calibração sobre as imagens geradas por um par de câmeras NTSC (*National TV System Committee*) utilizadas no projeto de colaboração entre o Laboratório de Processamento de Sinais Analógicos e Digitais (PADS) da UFRJ e a Petrobras. O arquivo texto



(a) Imagem hospedeira não marcada lena.bmp



(b) Imagem marcada com mensagem de 5000 caracteres através da abordagem QIM com inserção de 4 bits por bloco

Figura 13: Comparação entre a imagem hospedeira (a) e a imagem marcada com PSNR = 56,51 dB (b)

que contém os parâmetros de calibração do par de câmeras, ou seja, a marca d'água, possui 1880 caracteres.

O método utilizado para a escrita da marca d'água foi o LBM, pois através dos resultados obtidos nas Seções 4.1 e 4.2 concluiu-se que ele era melhor. As Figuras 14 e 15 apresentam as imagens originais onde a informação foi escrita. Ambas possuem dimensões 640×960 e cada *pixel* é representado por 24 bits.

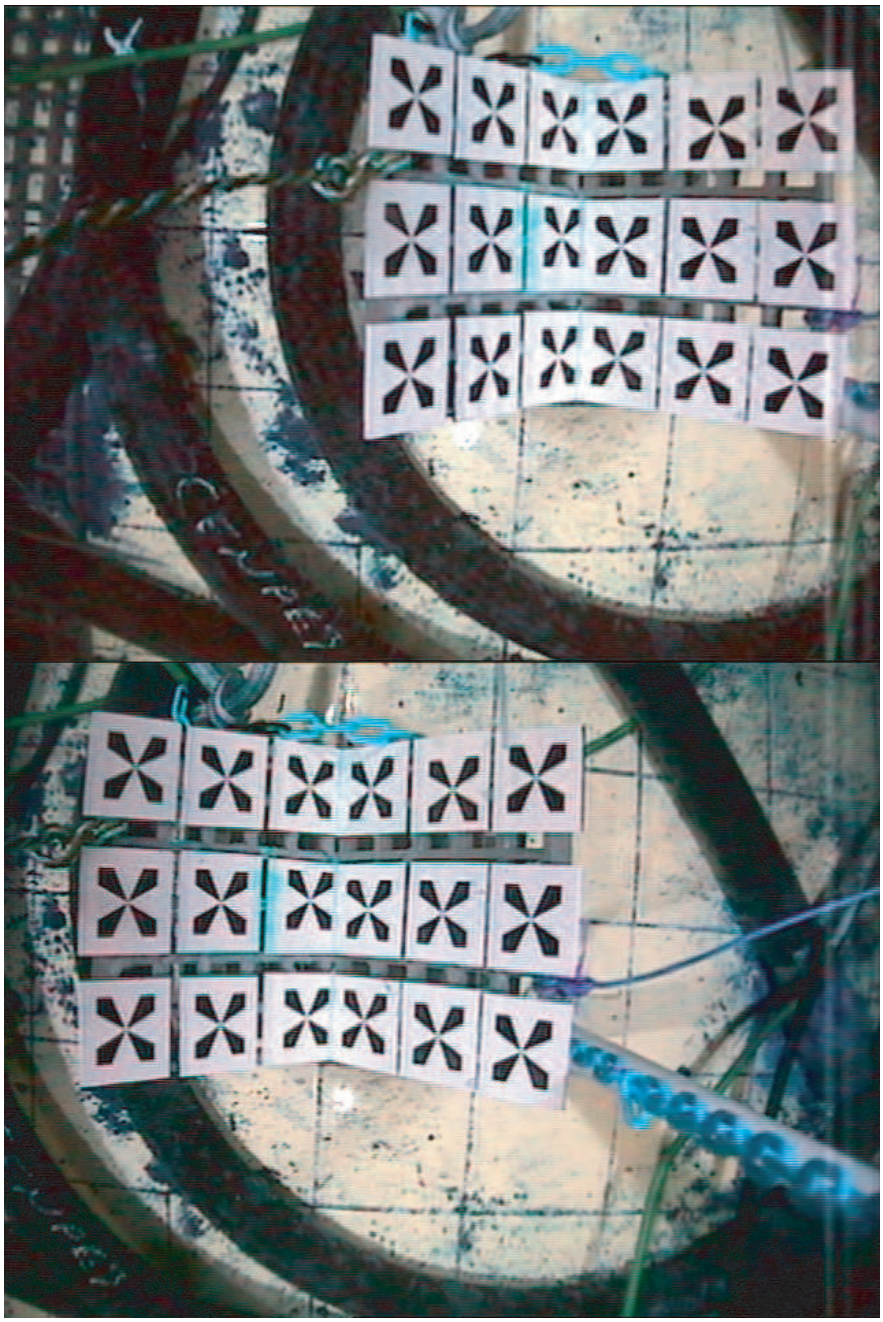


Figura 14: Imagem de calibração original

A escrita das informações de calibração sobre a imagem da Figura 14 apre-

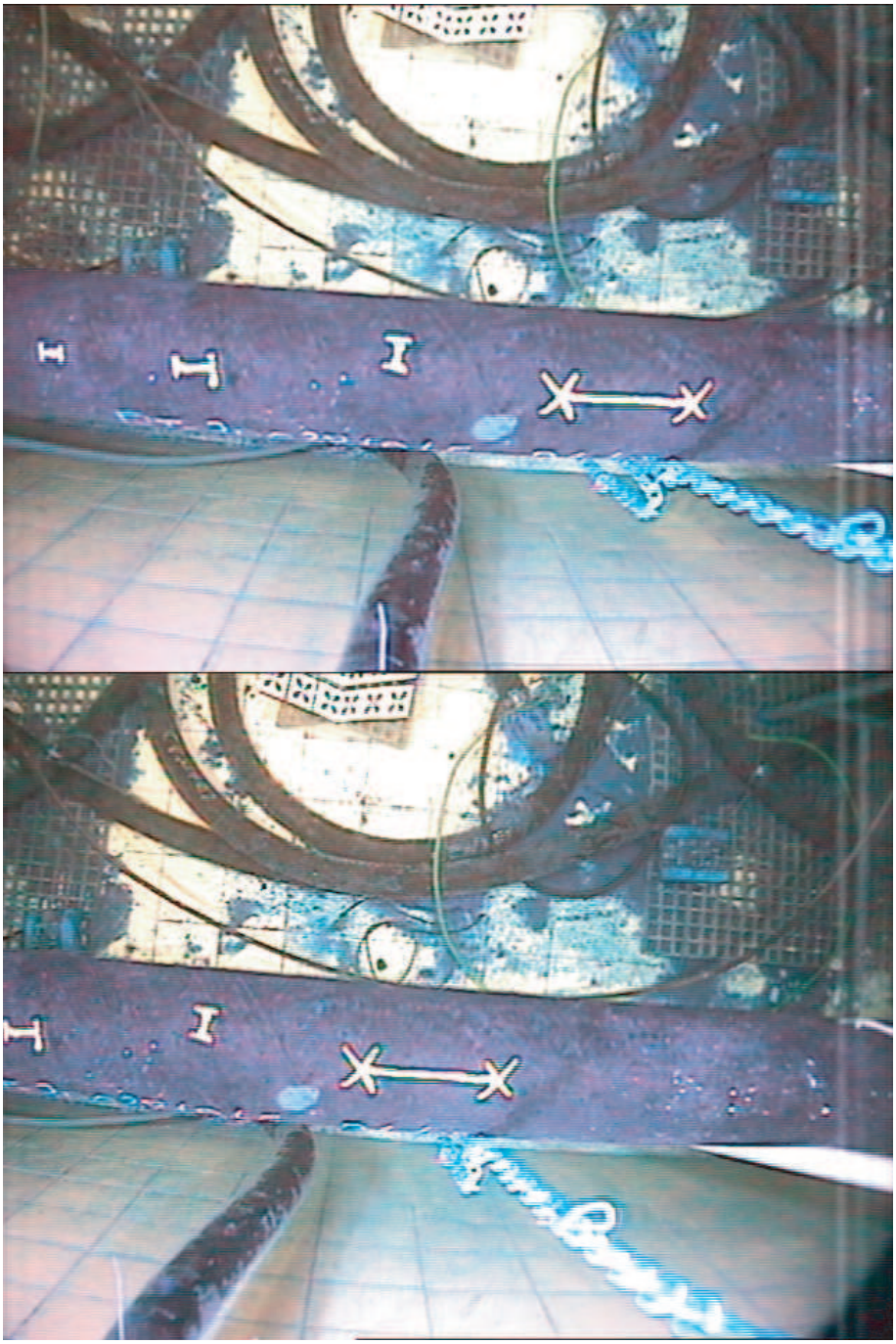


Figura 15: Imagem de dimensionamento original

sentou PSNR = 71,73 dB, enquanto que a escrita da mesma informação sobre a imagem da Figura 15 apresentou PSNR = 71,58 dB. Estes valores de razão sinal-ruído implicam que os ruídos sobre as imagens marcadas são imperceptíveis a olho nú. Além disto, estes resultados estão coerentes com os resultados apresentados na Tabela 2, referente ao método LBM.

As imagens marcadas são apresentadas nas Figuras 16 e 17.

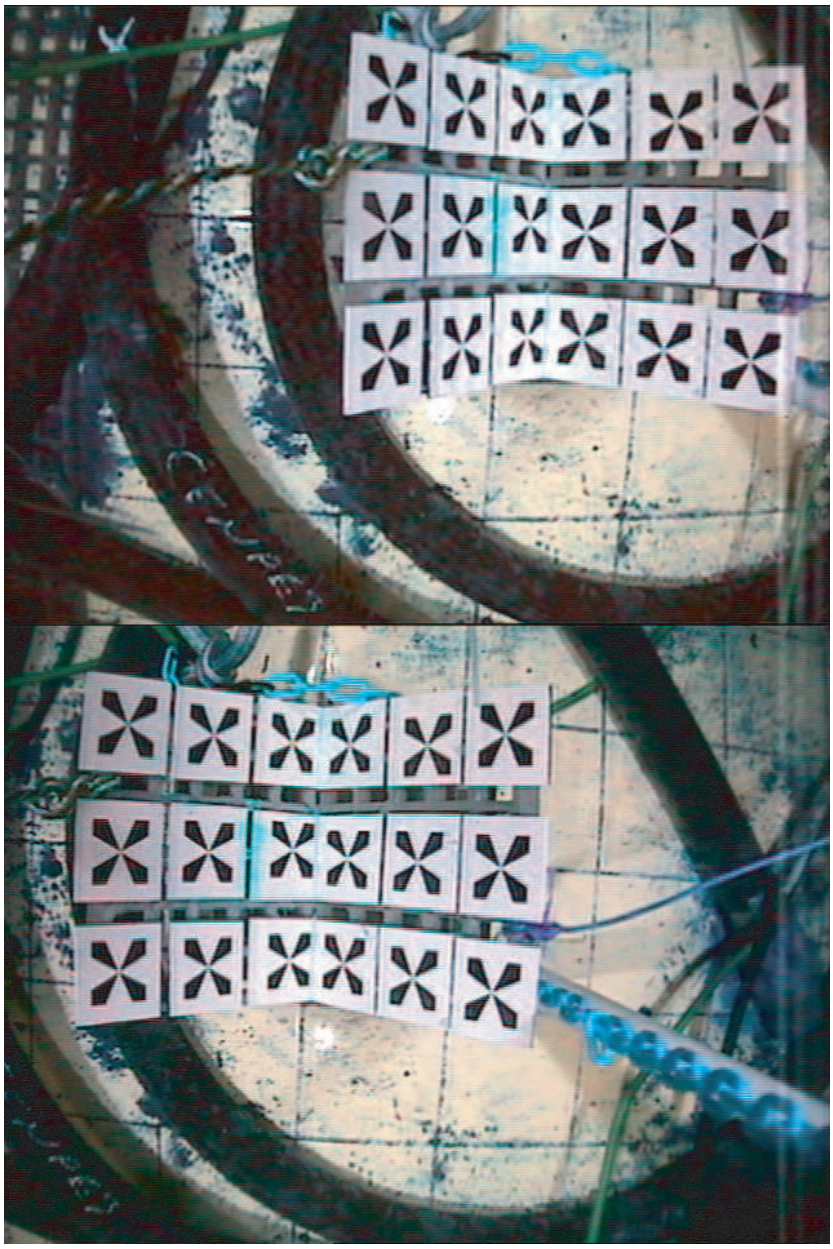


Figura 16: Imagem de calibração marcada

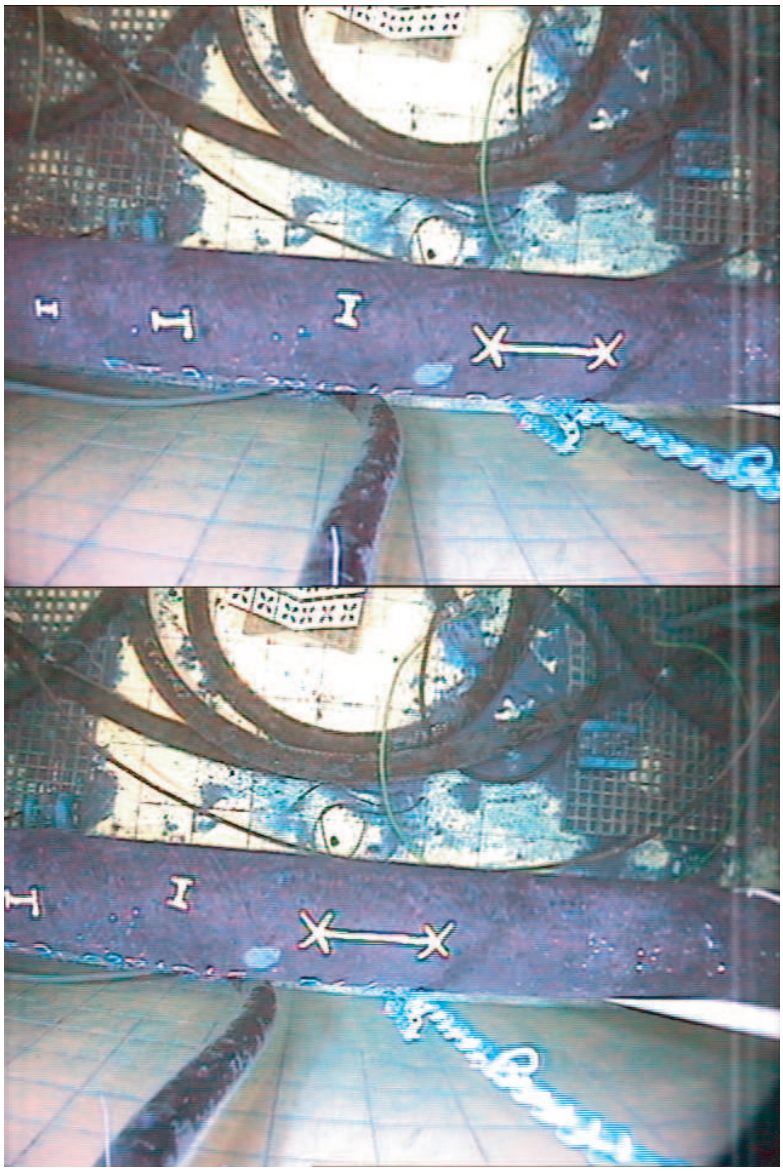


Figura 17: Imagem de dimensionamento marcada

5 Conclusão

O presente trabalho propôs a comparação entre dois métodos de inserção de informação de calibração em imagens para dimensionamento tridimensional a fim de escolher o melhor deles. Os métodos analisados foram o *Lower Bit Modulation* (LBM) e uma adaptação para *bitmaps* do *Quantization Index Modulation* (QIM).

O principal parâmetro utilizado para escolher o melhor método foi a razão sinal-ruído (PSNR). Além disto, verificou-se a quantidade de informação que cada método suportava e a invisibilidade da marca d'água nas imagens. Diversos testes foram realizados sobre os métodos de inserção de marca d'água digital e as razões sinal-ruído foram comparadas.

Os resultados dos testes mostraram que os melhores resultados foram alcançados pelo método LBM e pela primeira abordagem proposta do método QIM. Além disto, como o método LBM apresentou uma maior capacidade de inserção de informação sobre uma imagem hospedeira, ele foi o escolhido.

A escrita das informações de calibração sobre as duas imagens geradas por um par de câmeras NTSC utilizadas no projeto apresentaram relações sinal-ruído de 71,73 dB e 71,58 dB. Tais resultados são perfeitamente aceitáveis neste projeto e a marca d'água ficou imperceptível.

As propostas futuras deste estudo incluem a inserção de ruído sobre o canal de transporte de informação (no caso, imagens) para que sejam feitas análises quanto à robustez dos métodos de inserção de marca d'água digital. Também poderão ser estudados outros métodos de inserção de marca d'água digital para uma futura comparação de resultados.

Referências

[1] B. Chen e C.-E. W. Sundberg, "Broadcasting data in the FM band by means of adaptive contiguous band insertion and precancelling techniques," em *Proceedings IEEE International Conference on Communications*, vol. 2, pp. 823–827, Vancouver, BC, Canadá, Junho 1999.

[2] I. J. Cox e J.-P. G. Linnartz, "Some general methods for tampering with watermarks," *IEEE Journal on Selected Areas in Communications*, vol. 16, pp. 587–593, Maio 1998.

[3] A. Gersho e R. M. Gray, *Vector Quantization and Signal Compression.* Springer-Verlag, New York, 1992.

[4] J. G. R. C. Gomes, M. C. Q. Farias, S. K. Mitra, e M. Carli, "An accurate billing mechanism for multimedia communications," em *Proceedings International Conference on Multimedia and Expo*, vol. 3, pp. 93–96, Baltimore, MD, EUA, Julho 2003.

[5] F. Hartung e M. Kutter, "Multimedia watermarking techniques," *Proceedings IEEE*, vol. 87, pp. 1079–1107, Julho 1999.

[6] J.-P. Linnartz, T. Kalker, e J. Haitsma, "Detecting electronic watermarks in digital video," em *Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 4, pp. 2071–2074, Phoenix, AZ, EUA, Março 1999.

[7] H. C. Papadopoulos e C.-E. W. Sundberg, "Simultaneous broadcasting of analog FM and digital audio signals by means of adaptive precanceling techniques," *IEEE Transactions on Communications*, vol. 46, pp. 1233–1242, Setembro 1998.

[8] F. A. P. Petitcolas e M. G. Kuhn, "Information hiding - a survey," *Proceedings IEEE*, vol. 87, pp. 1062–1078, Julho 1999.

[9] M. D. Swanson e M. Kobayashi, "Multimedia dataembedding and watermarking technologies," *Proceedings IEEE*, vol. 86, pp. 1064–1087, Junho 1998.

APÊNDICE A – Código de Escrita de Marca d'Água do Método LBM

```
/* Escreve mensagem - LBM
Parametros de entrada: nome da imagem hospedeira; chave; mensagem
Saida: 0 = execucao ok */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "mex.h"

#define MESSAGE_NUM_BITS 8
#define MESSAGE_LENGTH_NUM_BITS 20

int nRows;
int nCols;
int nBits;
int numColours;

char * loadFile(char *);
char * dec2binMessage(char *, int);
char * dec2binMessageLength(int, int);
void escreveLBM(unsigned char *,char *, int);
double psnr(unsigned char *, unsigned char *);
long getImageInfo(FILE*, long, int);
```

```
void mexFunction(int nlhs, mxArray *plhs[], int nrhs,
const mxArray *prhs[])
{
    //-------------------------------------------------------------
    // verificacao dos argumentos do numero de argumentos de
        //entrada

    if(nrhs != 3) mexErrMsgTxt("3 parametros de entrada
    requeridos: nome da imagem; chave; mensagem(txt)");
        if(nlhs != 1) mexErrMsgTxt("1 parametro de saida requerido:
        0 = execucao OK");

    //-------------------------------------------------------------
    //gerencia argumentos da funcao mex-----------------------------
    char *str_arqImagem;
    char *str_arqChave;
    char *str_arqMensagem;
    double *saida;

    // carrega nome dos arquivos
    str_arqImagem = mxArrayToString(prhs[0]);
    str_arqChave = mxArrayToString(prhs[1]);
    str_arqMensagem = mxArrayToString(prhs[2]);

    plhs[0] = mxCreateDoubleMatrix(1,1,mxREAL);
    saida = mxGetPr(plhs[0]);

    //-------------------------------------------------------------
    //carrega vetor de componentes azuis (imagem.bmp)-------------

    FILE *host_image;
```

```
unsigned char aux;
unsigned char header[54];
int k;
unsigned char *blueComponents;
unsigned char *redComponents;
unsigned char *greenComponents;

host_image = fopen(str_arqImagem,"rb");

//--Le cabecalho

fread(header, sizeof(unsigned char), 54, host_image);

 char signature[2];
 int xpixpeRm, ypixpeRm;
 long nColors;
 long fileSize;
 long vectorSize;
 int rasterOffset;
 extern int nRows;
 extern int nCols;
 extern int nBits;
 extern int numColours;

 fseek(host_image, 0L, SEEK_SET);

 for(int i=0; i<2; i++)
 {
    signature[i] = (char)getImageInfo(host_image, i, 1);
 }
 if ((signature[0] == 'B') && (signature[1] == 'M'))
             printf("Imagem é Bitmap\n");
```

```
else
{
    printf("Imagem não é Bitmap\n");
    exit(0);
}

nBits = (int)getImageInfo(host_image, 28, 2);
printf("A imagem é: %d-bit\n", nBits);

if (nBits == 24){
    numColours = ldexp(1,nBits/3);
}

rasterOffset = (int)getImageInfo(host_image, 10, 4);

fileSize = getImageInfo(host_image, 2, 4);

nCols = (int)getImageInfo(host_image, 18, 4);
printf("Largura da imagem: %d\n", nCols);

nRows = (int)getImageInfo(host_image, 22, 4);
printf("Altura da imagem: %d\n", nRows);

xpixpeRm = (int)getImageInfo(host_image, 38, 4);

ypixpeRm = getImageInfo(host_image, 42, 4);

nColors = pow(2L,nBits);

vectorSize = (long)((long)nCols*(long)nRows);

fseek(host_image, rasterOffset, SEEK_SET);
```

```
//--Fim le cabecalho

blueComponents = (unsigned char*)
    malloc (sizeof(unsigned char)* nCols * nRows);
if (blueComponents==NULL) {mexPrintf("Memory error"); exit;}
redComponents = (unsigned char*)
    malloc (sizeof(unsigned char)* nCols * nRows);
if (redComponents==NULL) {mexPrintf("Memory error"); exit;}
greenComponents = (unsigned char*)
    malloc (sizeof(unsigned char)* nCols * nRows);
if (greenComponents==NULL) {mexPrintf("Memory error"); exit;}

k=0;
for (int i=0;i< nCols;i++)
    for(int j=0;j< nRows;j++){
        blueComponents[k] = 0;
        redComponents[k] = 0;
        greenComponents[k] = 0;
        k++;
    }

k=0;
for (int i=0;i< nCols;i++)
    for(int j=0;j< nRows;j++){
        aux = 0;
        fread(&aux,sizeof(unsigned char),1,host_image);
        blueComponents[k] = aux;
        fread(&aux,sizeof(unsigned char),1,host_image);
        redComponents[k] = aux;
        fread(&aux,sizeof(unsigned char),1,host_image);
        greenComponents[k] = aux;
```

```
        k++;
    }

fclose(host_image);

//-------------------------------------------------------------------
//carrega chave e mensagem-------------------------------------------

char *chave;
char *mensagem;
char *mensagemBIN_linha;

chave = loadFile(str_arqChave);
mensagem = loadFile(str_arqMensagem);
mexPrintf("tamanho da chave = %i chave = %s\n",
    strlen(chave),chave);
mexPrintf("tamanho da mensagem = %i mensagem = %s\n",
    strlen(mensagem),mensagem);

mensagemBIN_linha=dec2binMessage(mensagem,MESSAGE_NUM_BITS);

//-------------------------------------------------------------------
//escreve chave------------------------------------------------------

char *blueComponentsBCKP;

blueComponentsBCKP = (unsigned char*)
    malloc (sizeof(unsigned char)* nCols * nRows);
if (blueComponentsBCKP == NULL) {mexPrintf("Memory error");
    exit (2);}

strcpy(blueComponentsBCKP,blueComponents);
```

```
escreveLBM(blueComponents,chave,0);

//-------------------------------------------------------------------
//escreve tamanho da mensagem-------------------------------------

int messageLength;
char *messageLength_BIN;

messageLength = strlen(mensagemBIN_linha);
messageLength_BIN = dec2binMessageLength(messageLength,
    MESSAGE_LENGTH_NUM_BITS);

escreveLBM(blueComponents,messageLength_BIN,strlen(chave));

//-------------------------------------------------------------------
//escreve mensagem------------------------------------------------

escreveLBM(blueComponents,mensagemBIN_linha,(strlen(chave)+
    strlen(messageLength_BIN)));

//-------------------------------------------------------------------
//escreve bitmap modificado---------------------------------------

FILE *modified_image;
char newImage[50];

sprintf(newImage,"LBM_%s%s%s.bmp",str_arqImagem,str_arqChave,
    str_arqMensagem);
modified_image = fopen(newImage,"wb");

fwrite(header, sizeof(unsigned char), 54, modified_image);
```

```
    k=0;
    for (int i=0;i< nCols;i++)
        for(int j=0;j< nRows;j++){
            aux = 0;
            aux = blueComponents[k];
            fwrite(&aux,sizeof(unsigned char),1,modified_image);
            aux = redComponents[k];
            fwrite(&aux,sizeof(unsigned char),1,modified_image);
            aux = greenComponents[k];
            fwrite(&aux,sizeof(unsigned char),1,modified_image);
            k++;
        }

    fclose(modified_image);

    //-------------------------------------------------------------
    //calcula psnr

    double PSNR;
    PSNR = psnr(blueComponentsBCKP,blueComponents);

    //-------------------------------------------------------------

    free(blueComponentsBCKP);
    free(blueComponents);
    free(redComponents);
    free(greenComponents);
    mexPrintf("FIM\n");saida[1] = 1.0;
    return;
}
```

```
char * loadFile(char *fileName){

    FILE *pFile;
    long lSize;
    int result;
    char *data;

    pFile = fopen(fileName,"rb");
    if (pFile==NULL) {mexPrintf("File error"); exit (1);}

    // obtem tamanho do arquivo
    fseek (pFile , 0 , SEEK_END);
    lSize = ftell (pFile);
    rewind (pFile);

    // aloca memoria para comportar dados do arquivo
    data = (char*) malloc (sizeof(char)*lSize);
    if (data == NULL) {mexPrintf("Memory error"); exit (2);}

    // copia dados
    result = fread(data,1,lSize,pFile);
    if (result != lSize) {mexPrintf("Reading error"); exit (3);}

    data[lSize-1] = '\0';

    // fim
    fclose (pFile);

    return data;

}
```

```
void escreveLBM(unsigned char *channel,char *info, int delta){

    int sizeInfo;
    int sum;
    int resto;
    int m;
    int msgPart;
    int inc;
    int x;

    x = ldexp(1, 1);

    sizeInfo = strlen(info);

    m=0;
    for (int k=0; k < (sizeInfo/1) ; k++){

        sum = channel[k+delta];

        resto = sum % x;

        if ( (info[m] == '0') ){ //0
            msgPart = 0;
        }else
        if ( (info[m] == '1') ){ //1
            msgPart = 1;
        }

        inc = (x - resto + msgPart) % x;

        if (((channel[k+delta]+inc)!=(numColours-1)) &&
                ((channel[k+delta]+inc) > 0) ){
```

```
            channel[k+delta] = channel[k+delta] + inc;
        }
        else{
            mexPrintf("problema na soma\n");
            channel[k+delta] = channel[k+delta] - inc;
        }

        m = m+1;

    }

}

char * dec2binMessage(char *mensagem, int numBits){

    int j,i=0;
    char *mensagemASCII_BIN;
    unsigned int *aux;
    char aux2[2];
    int alloc;

    alloc = strlen(mensagem)*numBits;

    mensagemASCII_BIN = (char*) malloc (sizeof(char)*4000000);
    if (mensagemASCII_BIN == NULL) {mexPrintf("Memory error");
        exit (2);}

    aux = (unsigned int*) malloc (sizeof(unsigned int)*4000000);
    if (aux == NULL) {mexPrintf("Memory error"); exit (2);}

    sprintf(aux2,"");
    sprintf(mensagemASCII_BIN,"");
```

```
    while (mensagem[i]!='\0'){
        for (j=0;j<numBits;j++){
            aux[i*numBits+j] = mensagem[i] % 2;
            mensagem[i] /= 2;
            sprintf(aux2,"%i",aux[i*numBits+j]);
            strcat(mensagemASCII_BIN,aux2);
        }
        i=i+1;
    }

    free(aux);
    return (mensagemASCII_BIN);
}

char * dec2binMessageLength(int tam_mensagem, int numBits){

    int aux[MESSAGE_LENGTH_NUM_BITS];
    char tam_mensagem_BIN[MESSAGE_LENGTH_NUM_BITS];
    char aux2[2];

    sprintf(aux2,"");
    sprintf(tam_mensagem_BIN,"");

    for (int j=0;j<numBits;j++){
        aux[j] = (tam_mensagem % 2);
        tam_mensagem /= 2;
        sprintf(aux2,"%i",aux[j]);
        strcat(tam_mensagem_BIN,aux2);
    }

    return (tam_mensagem_BIN);
```

```
}

double psnr(unsigned char *original,unsigned char *modified){

  float PSNR, sum, a, b, c;
  int k=0;

  sum = 0;
  for (int i=0; i< nCols; i++)
    for (int j=0; j< nRows; j++){
      sum += (original[k] - modified[k]) * (original[k] -
          modified[k]);
      k++;
    }


  a = sum /( nCols * nRows);
  b = (numColours-1)*(numColours-1);
  c = b/a;

  PSNR = 10 * log10( 3*c );
  mexPrintf("PSNR = %f dB\n",PSNR);

  return ( PSNR );

}

long getImageInfo(FILE* inputFile, long offset, int
numberOfChars)
{
```

```
    unsigned char    *ptrC;
    long             value=0L;
    unsigned char    dummy;
    int              i;

    dummy = '0'; ptrC = &dummy;

    fseek(inputFile, offset, SEEK_SET);

    for(i=1; i<=numberOfChars; i++)
    {
        fread(ptrC, sizeof(char), 1, inputFile);

        value = (long)(value + (*ptrC)*(pow(256, (i-1))));
    }

    return(value);

}
```

APÊNDICE B – Código de Leitura de Marca d'Água do Método LBM

```
/* Le mensagem - LBM
Parametros de entrada: nome da imagem marcada; chave.
Saida: 0 = execucao ok
A mensagem sera gravado num arquivo txt*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "mex.h"

#define MESSAGE_NUM_BITS 8
#define MESSAGE_LENGTH_NUM_BITS 20

int nRows;
int nCols;
int nBits;
int numColours;

char * loadFile(char *);
int verificaChave(char *, char *);
int recuperaTamanhoMensagemLBM(unsigned char *, int, int);
int bin2dec(unsigned char *);
void leMensagemLBM(unsigned char *, unsigned char *, int, int);
```

```
long getImageInfo(FILE*, long, int);

void mexFunction(int nlhs, mxArray *plhs[], int nrhs,
const mxArray *prhs[])
{
    //-------------------------------------------------------------
    // verificacao dos argumentos do numero de argumentos de
        //entrada

    if(nrhs != 2) mexErrMsgTxt("2 parametros de entrada
        requeridos: nome da imagem; chave");
        if(nlhs != 1) mexErrMsgTxt("1 parametro de saida requerido:
        0 = execucao OK");

    //-------------------------------------------------------------
    //gerencia argumentos da funcao mex-------------------------

    char *str_arqImagem;
    char *str_arqChave;
    double *saida;

    // carrega nome dos arquivos
    str_arqImagem = mxArrayToString(prhs[0]);
    str_arqChave = mxArrayToString(prhs[1]);

    plhs[0] = mxCreateDoubleMatrix(1,1,mxREAL);
    saida = mxGetPr(plhs[0]);

    //carrega vetor de componentes azuis (imagem.bmp)-----------

    FILE *host_image;
    unsigned char aux;
```

```
unsigned char header[54];
int k;
unsigned char *blueComponents;
unsigned char *redComponents;
unsigned char *greenComponents;

host_image = fopen(str_arqImagem,"rb");

//--Le cabecalho

fread(header, sizeof(unsigned char), 54, host_image);

 char signature[2];
 int xpixpeRm, ypixpeRm;
 long nColors;
 long fileSize;
 long vectorSize;
 int rasterOffset;
 extern int nRows;
 extern int nCols;
 extern int nBits;
 extern int numColours;

 fseek(host_image, 0L, SEEK_SET);

 for(int i=0; i<2; i++)
 {
    signature[i] = (char)getImageInfo(host_image, i, 1);
 }
 if ((signature[0] == 'B') && (signature[1] == 'M'))
             printf("Imagem é Bitmap\n");
 else
```

```
{
    printf("Imagem não é Bitmap\n");
    exit(0);
}

nBits = (int)getImageInfo(host_image, 28, 2);
printf("A imagem é: %d-bit\n", nBits);

if (nBits == 24){
    numColours = ldexp(1,nBits/3);
}

rasterOffset = (int)getImageInfo(host_image, 10, 4);

fileSize = getImageInfo(host_image, 2, 4);

nCols = (int)getImageInfo(host_image, 18, 4);
printf("Largura da imagem: %d\n", nCols);

nRows = (int)getImageInfo(host_image, 22, 4);
printf("Altura da imagem: %d\n", nRows);

xpixpeRm = (int)getImageInfo(host_image, 38, 4);

ypixpeRm = getImageInfo(host_image, 42, 4);

nColors = pow(2L,nBits);

vectorSize = (long)((long)nCols*(long)nRows);

fseek(host_image, rasterOffset, SEEK_SET);
```

```
//--Fim le cabecalho

blueComponents = (unsigned char*)
    malloc (sizeof(unsigned char)* nCols * nRows );
if (blueComponents==NULL) {mexPrintf("Memory error"); exit;}
redComponents = (unsigned char*)
    malloc (sizeof(unsigned char)* nCols * nRows );
if (redComponents==NULL) {mexPrintf("Memory error"); exit;}
greenComponents = (unsigned char*)
    malloc (sizeof(unsigned char)* nCols * nRows );
if (greenComponents==NULL) {mexPrintf("Memory error"); exit;}

k=0;
for (int i=0;i< nCols ;i++)
    for(int j=0;j< nRows ;j++){
        blueComponents[k] = 0;
        redComponents[k] = 0;
        greenComponents[k] = 0;
        k++;
    }

k=0;
for (int i=0;i< nCols ;i++)
    for(int j=0;j< nRows ;j++){
        aux = 0;
        fread(&aux,sizeof(unsigned char),1,host_image);
        blueComponents[k] = aux;
        fread(&aux,sizeof(unsigned char),1,host_image);
        redComponents[k] = aux;
        fread(&aux,sizeof(unsigned char),1,host_image);
        greenComponents[k] = aux;
        k++;
```

```
    }

fclose(host_image);

//-------------------------------------------------------------
//carrega chave e mensagem-------------------------------------

char *chave;

chave = loadFile(str_arqChave);

//-------------------------------------------------------------
//copia trecho da mensagem que contem a chave-----------------

char *chaveAserVerificada;
int tamanhoChave;

tamanhoChave = strlen(chave);
mexPrintf("tamanho da chave: %i\n",tamanhoChave);

chaveAserVerificada = (unsigned char*)
    malloc (sizeof(unsigned char)*tamanhoChave);
if (chaveAserVerificada == NULL) {mexPrintf("Memory error");
    exit (2);}

strncpy(chaveAserVerificada,chave,tamanhoChave);

mexPrintf("trecho da chave selecionado: %s\n",
    chaveAserVerificada);

//-------------------------------------------------------------
//verifica chave-----------------------------------------------
```

```
int chaveVerificada = 1; //chave ok

chaveVerificada = verificaChave(chaveAserVerificada,chave);

//-------------------------------------------------------------------
//prossegue em funcao da verificacao da chave-------------------

int tamanhoMensagem;
unsigned char *mensagemRecuperada;

if (chaveVerificada ==0 ){
    mexPrintf("Chave falsa\n");
}
else
if (chaveVerificada == 1){
    mexPrintf("Chave OK\n");
    tamanhoMensagem = recuperaTamanhoMensagemLBM
    (blueComponents,strlen(chave),MESSAGE_LENGTH_NUM_BITS);

    mensagemRecuperada = (unsigned char*)
            malloc (sizeof(unsigned char)*tamanhoMensagem);
    if (mensagemRecuperada == NULL){
            mexPrintf("Memory error"); exit (2);}
    for (int i=0;i<tamanhoMensagem;i++){
        mensagemRecuperada[i]=0;
    }

    leMensagemLBM(mensagemRecuperada,blueComponents,
    tamanhoMensagem,(strlen(chave)+MESSAGE_LENGTH_NUM_BITS));
    mexPrintf("mensagem recuperada=%s\n",mensagemRecuperada);
}
```

```
    free(mensagemRecuperada);
    free(blueComponents);
    free(redComponents);
    free(greenComponents);
    mexPrintf("FIM\n");saida[1] = 1.0;
    return;

}

char * loadFile(char *fileName){

    FILE *pFile;
    long lSize;
    int result;
    char *data;

    pFile = fopen(fileName,"rb");
    if (pFile==NULL) {mexPrintf("File error");}

    // obtem tamanho do arquivo
    fseek (pFile , 0 , SEEK_END);
    lSize = ftell (pFile);
    rewind (pFile);

    // aloca memoria para comportar dados do arquivo
    data = (char*) malloc (sizeof(char)*lSize);
    if (data == NULL) {mexPrintf("Memory error");}

    // copia dados
    result = fread(data,1,lSize,pFile);
    if (result != lSize) {mexPrintf("Reading error");}
```

```
    data[lSize-1] = '\0';

    // fim
    fclose (pFile);

    return data;

}

int verificaChave(char *chaveDesconhecida, char *chave){

    int chaveVerificada = 1;
    int k=0;

    while ((chaveVerificada==1) && (k!=strlen(chave))){
        if (chave[k]==1){ //impar
            if ( (chaveDesconhecida[k] % 2) == 0 ){ //par
                chaveVerificada = 0;
            }
        }
        else
        if (chave[k]==0){ //par
            if ( (chaveDesconhecida[k] % 2) == 1 ){ //impar
                chaveVerificada = 0;
            }
        }
        k++;
    }

    return chaveVerificada;

}
```

```
int recuperaTamanhoMensagemLBM(unsigned char *blueComponents,
int delta, int numBits){

    int tamanhoMensagem;
    unsigned char tamanhoMensagemBin[MESSAGE_LENGTH_NUM_BITS];

    for (int i=0; i<numBits; i++){
        tamanhoMensagemBin[i] = (blueComponents[i+delta] % 2);
    }

    tamanhoMensagem = bin2dec(tamanhoMensagemBin);

    mexPrintf("tamanho da mensagem = %i\n",tamanhoMensagem);

    return tamanhoMensagem;

}

int bin2dec(unsigned char *tamanhoMensagemBin){

    int tamanhoMensagem=0;

    for (int i=0;i< MESSAGE_LENGTH_NUM_BITS; i++){
        tamanhoMensagem = tamanhoMensagem +
                ldexp(tamanhoMensagemBin[i], i);
    }

    return tamanhoMensagem;
}

void leMensagemLBM(unsigned char *mensagemRecuperada,
```

```
unsigned char *blueComponents, int tamanhoMensagem, int delta){

    for (int i=0;i<tamanhoMensagem;i=i+MESSAGE_NUM_BITS){
        for (int j=0; j<MESSAGE_NUM_BITS; j++){
            mensagemRecuperada[i/8] = mensagemRecuperada[i/8] +
                ldexp((blueComponents[i + j + delta] % 2) ,j);
        }
        mexPrintf("mensagemRecuperada[%i] = %c\n",i/8,
                mensagemRecuperada[i/8]);
    }
}

long getImageInfo(FILE* inputFile, long offset, int
numberOfChars){

    unsigned char    *ptrC;
    long             value=0L;
    unsigned char    dummy;
    int              i;

    dummy = '0'; ptrC = &dummy;

    fseek(inputFile, offset, SEEK_SET);

    for(i=1; i<=numberOfChars; i++)
    {
        fread(ptrC, sizeof(char), 1, inputFile);

        value = (long)(value + (*ptrC)*(pow(256, (i-1))));
    }
    return(value);
}
```