Adriano Arlei de Carvalho

Estudo e Implementação de Algoritmos Clássicos para Processamento Digital de Imagens

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para obtenção do título de bacharel

Orientador
Mário Luiz Rodrigues Oliveira

Lavras
Minas Gerais - Brasil
2003

Adriano Arlei de Carvalho

Estudo e Implementação de Algoritmos Clássicos para Processamento Digital de Imagens

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para obtenção do título de bacharel

Avaliada em *10 de dezembro de 2003*

Prof. Heitor Augustus Xavier Costa

Prof. Andréia Rodrigues de Assunção Schneider

Mário Luiz Rodrigues Oliveira
(Orientador)

Lavras
Minas Gerais - Brasil

Sumário

Lista de Figuras

Lista de Tabelas

Dedico a ...

Agradecimentos

Agradeço a minha família pelo carinho e apoio, em especial a minha mãe Vanda.
Agradeço aos meus amigos, que estiveram ao meu lado nesta difícil caminhada, em especial a Igor Chalfoun pelo carinho, respeito e amizade inabalável.
Agradeço a Sara Chalfoun pelo carinho e importantes sugestões na escrita da monografia.
Agradeço a prof. Mário pela sugestão do tema e orientação neste projeto.
Agradeço a UFLA por dar-me a oportunidade de expandir meus conhecimentos, em especial ao Departamento de Ciência da Computação e seus professores.

Estudo e Implementação de Algoritmos Clássicos para Processamento Digital de Imagens

O advento da imagem digital possibilitou à arte de registrar e manipular imagens uma evolução incomensurável. Porém algumas imagens não apresentam uma qualidade satisfatória, necessitando de um processo de tratamento (melhoramento). Foi desta necessidade inicial que surgiu o Processamento Digital de Imagens (PDI), que busca um melhoramento da qualidade visual ou enfatizar alguma característica específica. Este projeto tem por objetivo estudar e implementar alguns dos algoritmos clássicos em PDI, formando uma biblioteca para a construção de software livre. Os algoritmos implementados foram baseados nas seguintes etapas do PDI: Realce de imagens e Segmentação de imagens.

Study and Implementation of Classics Algorithms for Digital Image Processing

The advent of the digital image provided to the art of registering and to manipulate images an incommensurable evolution. Nevertheless some images don't present a satisfactory quality, needing a treatment (improvement) process. It was of this initial need that the Digital Processing of Images appeared (PDI), that looks for um improvement of the visual quality or to emphasize some specific characteristic. This project has for objective to study and to implement some of the classic algorithms in PDI, forming a library for the construction of free software. The implemented algorithms will be based in the following stages of PDI: Enhance of images and Segmentation of images.

Capítulo 1

Introdução

A imagem captada pelo sentido visual humano tem uma ampla aplicação, desde a pintura à engenharia, constituindo-se em uma das importantes fontes na aquisição de informações sobre o meio em que vive e forma ponto de referência para a execução de projetos de diferentes naturezas.

A imagem é formada quando há interação entre raios luminosos e objeto. Parte da energia deixa o objeto na forma de novos raios de luz, que podem ser captados pelo sistema visual humano ou por outro sistema de captação de imagem.

A exemplo dos computadores desde seu advento, a imagem digital vem experimentando significativa evolução com a ampliação do leque de possibilidades no sentido de registrar e manipular imagens. Este salto qualitativo veio ao encontro da demanda por imagem que apresente um elevado padrão de qualidade.

1.1 Contextualização

Algumas vezes a imagem não apresenta boa qualidade visual para interpretação humana. Esta perda de qualidade pode ocorrer devido a vários fatores,por exemplo, problemas de iluminação da cena no momento de sua aquisição, problemas com os sistemas de captação de imagem e problemas de transporte da imagem. Desta forma torna-se necessário que a imagem passe por algum processo de tratamento com o objetivo de melhorar a qualidade da informação pictorial da imagem ou enfatizar alguma característica de interesse. O Processamento Digital de Imagem (PDI) é um dos ramos da computação gráfica responsável por esse processo.

Em processamento digital de imagem tanto a entrada do processo quanto a saída são imagens, sendo a saída uma versão melhorada ou destacada da entrada. Esse processo tornou-se imprescindível para alguns setores, tais como, análise de imagens biomédicas, meteorologia por meio de imagens de satélites, reconhecimento de padrões, restauração de pinturas antigas, análise de recursos naturais, monitorameno de poluição, cartografia, geologia, análise de imagens espaciais e

controle de queimadas.

O processamento digital de imagens é dividido em quatro etapas básicas:

- **Aquisição de Imagens**
  Etapa responsável pela captação de imagens de uma cena específica.
- **Realce de Imagens**
  Etapa responsável pela melhoria da qualidade visual ou extração de alguma característica de interesse.
- **Segmentação de Imagens**
  Etapa responsável por encontrar ou extrair objetos de uma imagem.
- **Classificação**
  Etapa responsável pela segmentação específica usando técnicas de reconhecimento de padrões.

1.2 Motivação

A diversidade de aplicações de técnicas para processamento digital de imagens, bem como a verificação do funcionamento dos algoritmos constitui a principal motivação para realização desse projeto.

1.3 Objetivo

Este projeto estudou e implementou alguns dos algoritmos clássicos em processamento digital de imagens para formação de uma biblioteca e de uma aplicação (*software*).

A biblioteca é composta por dez algoritmos com os seguintes objetivos: melhoramento de contraste, extração bordas, filtragem de ruídos e extração de objetos.

O *software* implementado foi denominado **wxPDI**, este sendo de código aberto com licença *GPL*.

Os capítulos seguintes apresentam um visão mais detalhada sobre o processamento digital de imagens. Estes foram divididos da seguinte forma:

- Fundamentação Teórica
  Este capítulo apresenta os conceitos e os passos para o PDI.
- Metodologia
  Este capítulo apresenta a metodologia adotada, os algoritmos que fazem parte da biblioteca e a linguagem utilizada.

- Resultados e Discussões
  Este capítulo apresenta os resultados e as discusões dos algoritmos implementados, a biblioteca e o *software* **wxPDI**.

- Conclusão
  Este capítulo apresenta as conclusões sobre o trabalho realizado.

- Trabalhos Futuros
  Este capítulo propõe trabalhos futuros relacionados a esse projeto.

Capítulo 2

Fundamentação Teórica

Este capítulo descreve os fundamentos e as etapas do processamento digital de imagens.

2.1 Fundamentos para Processamento Digital de Imagens

Esta seção apresenta as propriedades da imagem, passando pela definição, a forma de representação, o armazenamento e as propriedades do *pixel*.

2.1.1 Imagem em Escala de Cinza

Segundo [Banon (2000)], a definição de imagem em escala de cinza pressupõe a existência de dois conjuntos: O conjunto E formado por quadrados adjacentes, dispostos ao longo de um número de linhas e colunas (Figura 2.1), e o conjunto K, formado pelos níveis de cinza ou escalas de cinza, com K variando de 0 a 255.

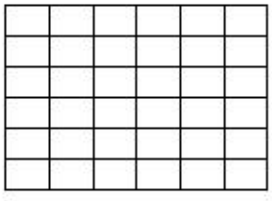

Figura 2.1: Conjunto E

Para [Banon (2000)] um *pixel* é um elemento do produto cartesiano de um quadrado localizado em E e um nível de cinza localizado em K, ou seja, o par ordenado (x, s), onde x é um elemento do conjunto E e s um elemento do conjunto K (Figura 2.2). O mesmo autor define a imagem em escala de cinza como sendo um gráfico de mapeamento de E para K.

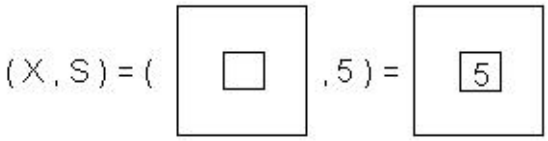

Figura 2.2: Produto cartesiano E x K

2.1.2 Definição Matemática de Imagem

Segundo [Mascarenhas & Velasco (1989)] uma imagem pode ser definida matematicamente como uma função $f(x, y)$, onde o valor nas coordenadas espaciais xy corresponde ao brilho (intensidade) da imagem nessa coordenada (Figura 2.3).

Figura 2.3: imagem [Bastos]

2.1.3 Imagem Digital

Uma imagem de um objeto real é em princípio contínua tanto na sua variação espacial como nos níveis de cinza [Mascarenhas & Velasco (1989)]. Visando o seu processamento computacional a imagem deve ser digitalizada, ou seja discretizar tanto no espaço quanto na amplitude. A digitalização das coordenadas espaciais é chamada amostragem da imagem e a digitalização da amplitude é chamada de quantização dos níveis de cinza [Brito & Carvalho (1998)].

2.1.4 Representação de Imagens

Existem duas maneiras de representar uma imagem: representação vetorial e representação matricial. Estas representações diferem na natureza dos dados que compõe a imagem.

Segundo [Schneider (2001)] uma imagem vetorial é uma imagem de natureza geométrica, ou seja, ela é definida em função de elementos geométricos e parâmetros (Figura 2.4).

Figura 2.4: Representação vetorial [Schneider (2001)]

Segundo [Schneider (2001)] uma imagem matricial é uma imagem de natureza discreta, ou seja, a imagem é formada de elementos independentes, dispostos na forma de uma matriz, cada um contendo uma unidade de informação da imagem (Figura 2.5). Esta representação não armazena nenhuma informação geométrica dos objetos contidos na imagem o que torna difícil a manipulação de sua estrutura. No entanto esse esta é capaz de representar qualquer tipo de imagem.

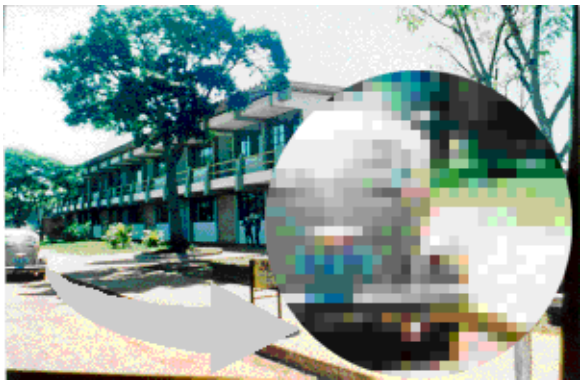

Figura 2.5: Representação matricial [Schneider (2001)]

2.1.5 Propriedades de um *pixel*

Uma propriedade importante de um *pixel* é sua vizinhança, esta seção define a vizinhança 4 e a vizinhança 8 de um *pixel*.

•Vizinhança 4

A vizinhança 4 de um *pixel* P é definida pelo conjunto dos *pixels* adjacentes a P, não levando em conta os *pixels* localizados nas diagonais passando por P (Figura 2.6).

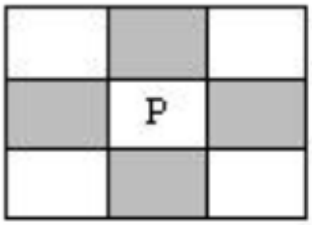

Figura 2.6: Vizinhança 4 de um *pixel*

•Vizinhança 8

A vizinhança 8 de um *pixel* P é formada pelo conjunto de todos os *pixels* que são adjacentes a P (Figura 2.7).

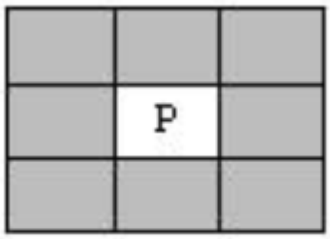

Figura 2.7: Vizinhança 8 de um *pixel*

2.1.6 Formatos de Imagens

O formato de uma imagem refere-se ao modo como os dados da imagem estão armazenados em arquivos e como se faz a interpretação desses dados para a visualização da imagem. Neste projeto foram considerados os seguintes formatos de armazenamento de imagem:

- JPEG;
- PNG;
- BitMap;
- Tiff;

•Formato JPEG

Segundo [Ramos (2000)] o formato JPEG é o mais utilizado no armazenamento e na transmissão de imagens estáticas em multimídia e na *internet*. Dentre as principais características do JPEG estão:

- Grande capacidade de compressão;
- Padronização internacional;
- Pequena perda perceptível
- Ampla aceitação pela comunidade da *internet*

•Formato PNG

O formato PNG, segundo [Ramos (2000)] é indicado para imagens que necessitam de compressão sem perdas, é um formato bastante aplicado na *internet* e em editoração de imagens. Dentre as principais características do PNG estão:

- Formato livre;
- Permite paleta de cores;
- Permite cores com oito, 16, 32, e 48 *bits* por *pixel*;
- Permite transparência em oito e 16 *bits*;
- Permite múltiplas plataformas;

Estrutura de dados do arquivo PNG:
Os primeiros oito bits são reservados para assinatura PNG
IHDR - Cabeçalho da imagem
PLTE - Paleta de cores usadas na imagem
IDAT - Chunks de dados da imagem
IEND - Indicador de fim de dados

Figura 2.8: Estrutura do arquivo PNG [Ramos (2000)]

•Formato BMP

Os formato *bitmap* foi desenvolvido pela *Microsoft* para aplicações no sistema Windows ® . É um formato de estrutura muito simples, tornando mínimas as possibilidades de erros na interpretação dos dados. Os arquivos possuem geralmente a extensão .BMP e aceitam imagens com um, quatro, oito, 16, 24 e 32 *bits* por *pixel* [Ramos (2000)]. A Figura 2.9 abaixo mostra a estrutura do arquivo *bitmap*.

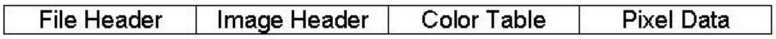

Figura 2.9: Estrutura do arquivo *bitmap* [Ramos (2000)]

•Formato TIFF

O formato TIFF foi desenvolvido em 1986 em uma tentativa de criar um padrão para imagens geradas por equipamentos digitais. O formato é capaz de armazenar imagens em preto ou branco, escalas de cinza e em paletas de cores com 24 ou com 32 emphbits. O TIFF é reconhecido por praticamente todos os programas de imagem [Ramos (2000)]. A Figura 2.10 ilustra a estrutura do arquivo TIFF.

Header	Image File Directory	Directory Entry	Data Block

Figura 2.10: Estrutura do arquivo TIFF [Ramos (2000)]

2.2 Etapas para o Processamento Digital de Imagens

Esta seção apresentará os passos para o processamento digital de imagens, enfatizando as etapas abordadas neste projeto, Aquisição de imagem, Realce de imagem, Segmentação de imagem e Classificação de imagem.

2.2.1 Aquisição de Imagens

Para obtenção de imagens digitais são necessários dois elementos: dispositivos físicos captadores de imagem e digitalizador. Dispositivos físicos são sensíveis a espectros de energia eletromagnética e o digitalizador converte o sinal elétrico desses dispositivos para o formato digital. Estes elementos são chamados de sistemas de imageamento. A exemplo mais conhecido deste sistema é câmera digital (Figura 2.11), outros são *scanners* e sensores presentes em satélites. Detalhes sobre aquisição de imagem podem ser vistos em [Gonzales & Woods (1992)].

Figura 2.11: Câmera digital

2.2.2 Realce de Imagem

O melhoramento de imagem é obtido através de técnicas, tais como, o melhoramento de contraste e filtragem aplicadas com finalidades específicas enfatizando características de interesse ou recuperando imagens que sofreram algum tipo de degradação devido a introdução de ruído, perda de contraste ou borramento.

A aplicação dessas técnicas, designadas como realce de imagem, são transformações radiométricas que modificam o valor dos níveis de cinza dos pontos da imagem.

•Melhoramento de Contraste

Melhoramento de contraste busca melhorar a qualidade visual da imagem através da manipulação dos níveis de cinza. Uma imagem possui valores de intensidade de *pixel*, variando de 0 a 255. Quanto mais espalhados os *pixels* da imagem neste intervalo melhor é o seu contrate. Para [Mascarenhas & Velasco (1989)] contraste consiste numa diferença local de luminância e pode ser definido como a razão dos níveis de cinza médios do objeto e do fundo.

O processo de melhoramento de contraste transforma a escala de cinza de forma pontual, ou seja, o novo valor do ponto depende somente do valor original do ponto. Uma função de transferência mapeia o valor de um ponto para um novo valor. Essa função é definida da seguinte forma: $g(x, y) = T(f(x, y))$ onde, $f(x, y)$ é o valor do nível de cinza original, T é a função de transferência e $g(x, y)$ é o novo valor do ponto.

Uma boa forma de avaliar o contraste de uma imagem é analisar seu shistograma. O histograma é um gráfico que representa a distribuição dos *pixels* para cada nível de cinza da imagem. No eixo horizontal fica a escala de cinza e no eixo vertical fica a quantidade de *pixels*.

A (Figura 2.12) ilustra duas imagens com seus respectivos histogramas, note que trata-se da mesma imagem. Observando o histograma da primeira imagem pode-se notar que a distribuição dos *pixels* concentra-se próximo ao nível zero, exemplificando uma imagem com baixo contraste. A segunda imagem possui alto contraste, seus *pixels* estão melhor distribuídos no histograma, possibilitando um melhor discernimento das informações contidas na imagem.

Funções de transferências modificam histograma para obter um imagem com melhor contraste. As funções a seguir foram retiradas de [Maria (2000)].

- Função Linear => $g(x, y) = a * f(x, y) + b$, onde os parâmetros a e b são valores de ajuste da função que variam de 0 a 255 (figura 2.13). Esta função é utilizada para redistribuir os *pixels* da imagem de forma linear.

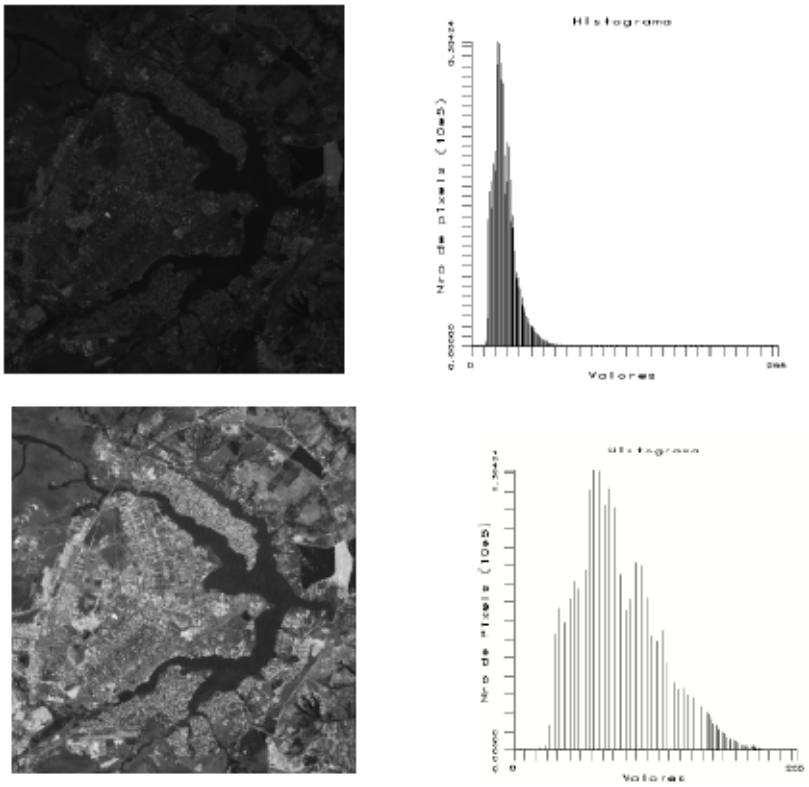

Figura 2.12: Histograma de imagens [Maria (2000)]

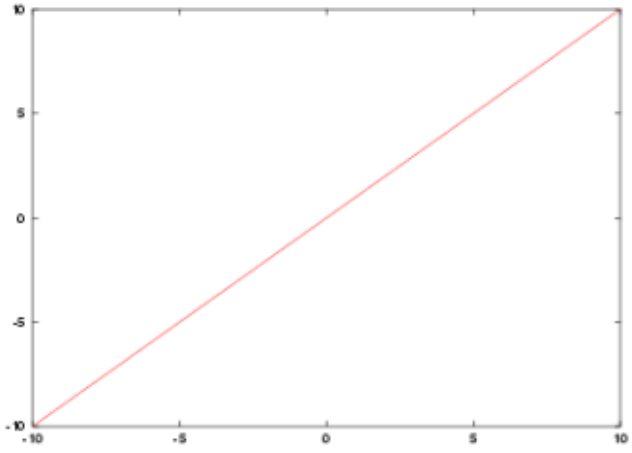

Figura 2.13: Função linear

- Raiz Quadrada => $g(x,y) = a * \sqrt{f(x,y)}$, onde o parâmetro a é o fator de ajuste da função que varia de 0 a 255. Esta função é utilizada para realçar as áreas escuras da imagem. Este comportamento é facilmente verificado no gráfico da Figura 2.14, note que a inclinação da função é maior próximo a áreas escuras da imagem;

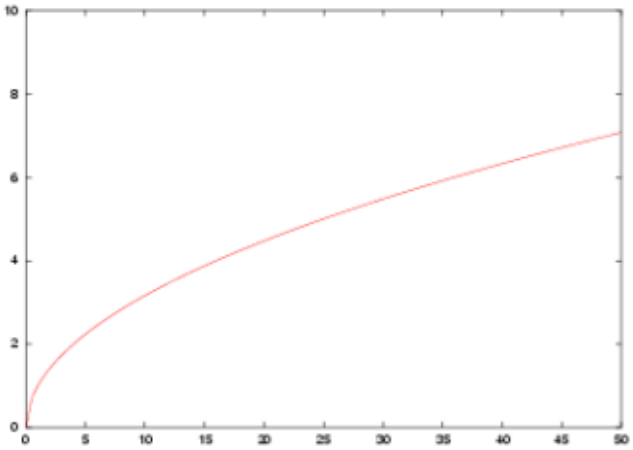

Figura 2.14: Função raiz quadrada

- Logaritima => $g(x,y) = a*log(f(x,y)+1)$, onde o parâmetro a é o fator de ajuste da função que varia de 0 a 255. Esta função tem um comportamento semelhante a raiz quadrada, porém realçando um intervalo menor de áreas escuras (Figura 2.15);

- Inversa => $g(x,y) = -(a*f(x,y)+b)$, onde os parâmetros a e b são valores de ajuste da função que variam de 0 a 255. Esta função inverte os níveis de cinza da imagem, ou seja, valores de *pixels* pretos são mapeados para brancos, valores brancos são mapeados para pretos e valores intermediários são mapeados para valores intermediários (figura 2.16);

•Filtragem

O processo de filtragem procura extrair informações como as bordas da imagem ou corrigir algumas degradações na imagem, tais como: borrões, ruídos inseridos pelo processo de imageamento ou na trasmissão da imagem. Os filtros são divididos em duas categorias: filtros no domínio do espaço e filtros no domínio da freqüência. Detalhes sobre filtragem no domínio da freqüência em [Gonzales & Woods (1992)].

A filtragem no espaço é considerada uma operação local, ou seja, o nível de cinza de um ponto depende do original e de sua vizinhança. O princípio de funcionamento de tal filtro está baseado em máscaras de deslocamento as quais são

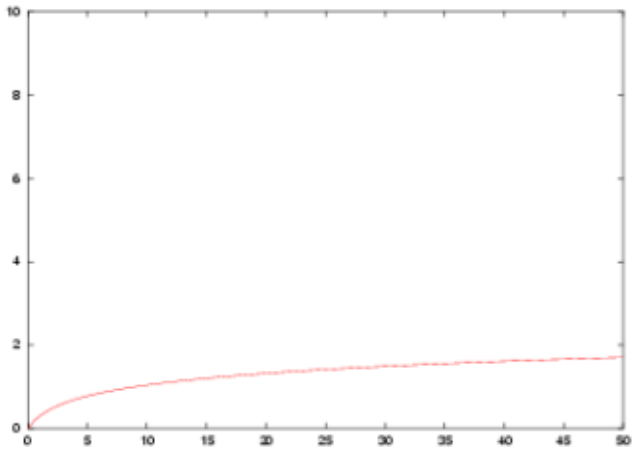

Figura 2.15: Função logaritmica

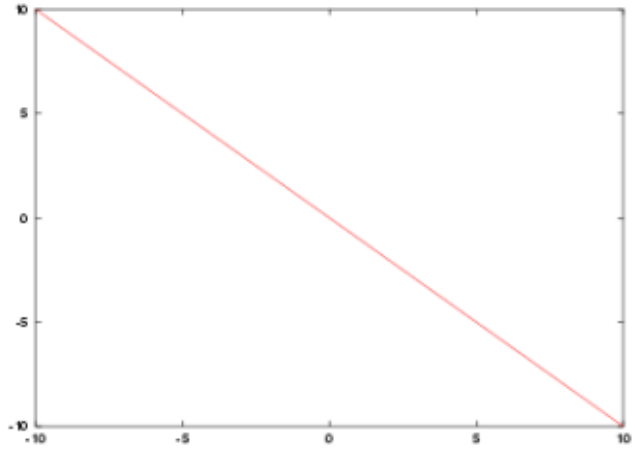

Figura 2.16: Função inversa

matrizes com pesos associados em cada posição. A máscara com centro na posição (x, y) é colocada sobre o *pixel* a ser modificado na imagem. O *pixel* correspondente na imagem é substituído por um valor que considera os *pixels* vizinhos e os pesos correspondentes na máscara. A soma de todos os produtos dos pesos da máscara pelos *pixels* correspondente na imagem resulta em um novo valor de cinza que substituirá o *pixel* central.

•Filtros Detectores de Bordas

Segundo [Mascarenhas & Velasco (1989)], a borda é caracterizada por mudanças a-bruptas de descontinuidade na imagem. Através do processo de detecção de bordas pode-se localizar e distinguir os objetos presentes na imagem, bem como descobrir algumas de suas propriedades, tais como forma e tamanho.

Os detectores de bordas se baseiam na idéia de derivadas. A derivada mede a taxa de variação de uma função. Em imagens esta variação é maior nas bordas e menor em áreas constantes. Ao percorrer a imagem marcando os pontos onde a derivada possui uma variação maior, ou seja pontos de máximo, no fim todas as bordas terão sidos marcadas [Marta (1998)].

Como as imagens são bidimensionais, usa-se derivadas parciais nas direções vertical y e horizontal x, que são representadas por um vetor gradiente.

- Operador gradiente de Roberts
  O operador gradiente de Roberts utiliza uma matriz (2 X 2), nas direções horizontal Gx e vertical Gy [Marta (1998)].

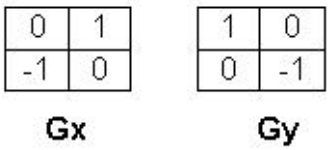

Figura 2.17: Mascara de Roberts

O gradiente é calculado pela seguinte formula: $GR = \sqrt{Gx^2 + Gy^2}$

- Operador gradiente de Sobel
  O operador gradiente de Sobel utiliza matriz (3 X 3) [Marta (1998)].

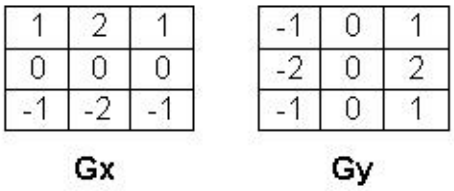

Figura 2.18: Mascara de Sobel

O gradiente é calculado pela seguinte formula: $GR = \sqrt{Gx^2 + Gy^2}$

•Filtros de Ruídos

Para [Marta (1998)] os ruídos são um conjunto de *pixels* aleatórios diferentes dos dados da imagem.

- Filtro da Média
  O filtro da média utiliza uma máscara que percorre a imagem e substitui cada *pixel* da imagem pela média de seus vizinhos. O objetivo desse filtro não é de eliminar o ruído e sim suavizá-lo. A mascara utilizada é uma matriz (3 X 3) baseada na vizinhança 8 [Facon (2002)].

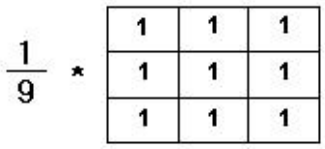

Figura 2.19: Mascara da média

- Filtro da mediana
  O objetivo do filtro da mediana é substituir o *pixel* central de uma matriz, geralmente 3 X 3, pelo valor que estiver na mediana desses *pixels* ordenados em ordem crescente de nível de cinza [Facon (2002)].

  Este filtro pode ser facilmente implementado da seguinte forma:
  1) Para todos os pontos da imagem;
  2) Escolha um ponto e o atribua a x;
  2.1) Coloque em ordem crescente x e sua vizinhança 8, guarde esses valores em um conjunto X;
  2.2) O novo valor do ponto é o valor que está na mediana do conjunto X;

  Este filtro baseia-se no fato de que se o ponto for um ruído, ao colocar em ordem crescente este ponto e sua vizinhança, o ruído tende a ficar em um dos extremos do conjuto X, sendo substituído pelo valor mediano do conjunto X.

2.2.3 Segmentação de Imagens

Por segmentação de uma imagem entende-se a extração ou identificação dos objetos contidos na imagem, onde o objeto é toda característica com conteúdo semântico relevante para a aplicação desejada [Mascarenhas & Velasco (1989)]. A segmentação faz parte de um passo maior que é a classificação da imagem. A etapa

de segmentação divide a imagem em regiões, sem considerar o processo de classificação. Para [Mascarenhas & Velasco (1989)], a região em uma imagem é um conjunto de pontos ligados, ou seja, de qualquer ponto da região pode-se chegar a qualquer outro ponto por um caminho inteiramente contido na região, baseado na vizinhança 4 ou vizinhança 8.

Segmentação Baseado em Crescimento de Regiões

A segmentação baseada no crescimento de regiões pode ser feito de duas formas: por varredura ou por busca em todas as direções. Ambos os métodos baseiam-se no fato das regiões a serem detectadas apresentarem propriedades locais aproximadamente constantes como, por exemplo, o nível de cinza.

- Crescimento de regiões por busca em todas as direções.[1]
  No crescimento por busca, em todas as direções uma única região é crescida por vez.

  Algoritmo:
  Inicialmente região$(p) = 0$ para todo o ponto p da imagem.
  1) $k = 0$;
  2) escolha um ponto x, tal que região $(x) == 0$, se não existe ponto nesta condição, então pare: fim;
  3) inicie nova região: $k = k + 1$; região$(x) = k$;
  4) ache todos os pontos p da vizinhança de x, tal que região $(p) == 0$ e p possa ser adicionado à região de x sem violar o critério de homogeneidade: faça região(p) = região(x) e guarde estes pontos em um conjunto X;
  5) se X está vazio, vá para 2; caso contrário, escolha e extraia um ponto x de X;
  6) vá para 4;
  Onde:
  x é o ponto (x, y);
  os vizinhos de P é a sua vizinhança 4; região (p) indica a região do ponto p.

- Crescimento de regiões segundo a varredura[2]
  Neste método, a imagem é percorrida da esquerda para direita e de cima para baixo comparando um ponto inicial da imagem (x, y) com todos os outros pontos da imagem, adicionando ou não esses pontos a região do ponto inicial.

[1] Retirado de [Mascarenhas & Velasco (1989)]
[2] Modificado de [Mascarenhas & Velasco (1989)]

Algoritmo:
1) Percorra a imagem segundo a varredura;
2) Escolha um ponto x na imagem e o rotule;
3) Compare x com todos os outros pontos da imagem;
3.1) Adicione a região de x os pontos que não violarem o critério de homogeneidade;

2.2.4 Classificação de Imagem

A classificação é o processo de extração de informações da imagem. Consiste na divisão da imagem em classes, ou seja, segmentação e posterior identificação destas classes. Esta pode ser feita de forma não automática, onde um especialista humano extrai informações baseando-se na inspeção visual da imagem ou de forma automática, feita por computador [Maria (2000)]. A classificação de imagem por computador pode ser realizada usando técnicas de redes neurais. A rede é treinada de forma que ela possa identificar e agrupar todos os *pixels* em classes. O aprendizado da rede pode ser feito de duas formas: aprendizado supervisionado e aprendizado não supervisionado.

- Aprendizado supervisionado
  Para [Mascarenhas & Velasco (1989)] quando existem amostras disponíveis de classificação conhecida, define-se que o problema de aprendizado é supervisionado. A rede, tendo esta amostra conhecida inicialmente, tenta encontrar na imagem todos os *pixels* que podem ser agrupados a esta amostra conhecida, formando uma classe. Os *pixels* podem ser agrupados a uma mesma classe se por um critério de similaridade eles são iguais ou semelhantes. Este critério é chamado de limiar, podendo ser a intensidade luminosa do *pixel*.

- Aprendizado não supervisionado
  No aprendizado sem supervisão as amostras na área de treinamento não são rotulados [Mascarenhas & Velasco (1989)]. Neste caso a rede tem que aprender sozinha quais *pixels* podem fazer parte de uma mesma classe, sem nenhum conhecimento inicial.

Capítulo 3

Metodologia

Este capítulo apresenta a metodologia utilizada neste trabalho.

Este projeto implementou alguns dos principais algoritmos para processamento digital de imagens. Esses algoritmos formaram uma biblioteca, que tornou possível a implementação do aplicativo **wxPDI**, um *software* de código aberto para processamento digital de imagens.

3.1 Algoritmos Implementados

Foram escolhidos algoritmos clássicos da literatura, baseados nas seguintes etapas do processamento digital de imagens:

- **Realce de Imagem**
  Técnicas de realce buscam melhorar a qualidade visual da imagem ou enfatizar alguma característica de interesse como bordas. Os algoritmos implementados foram:

  - Algoritmos de melhoramento de contraste, os quais trabalham de forma pontual na imagem através de funções de transferência; A saber.

    – Função de Transferência Linear;
    – Função de Transferência Raiz Quadrada;
    – Função de Transferência Logaritma;
    – Função de Transferência Inversa;

  - Algoritmos de filtragem, cujos filtros trabalham de forma local, utilizando máscara de deslocamento. Os algoritmos são:

  - Filtro detectores de bordas
    - Operador gradiente de Roberts
    - Operador gradiente de Sobel

  - Filtros de ruídos
    - Filtro da média
    - Filtro da mediana

- Segmentação de Imagem
  Técnicas de segmentação buscam encontrar regiões para extração ou identificação dos objetos presentes na imagem. Os algoritmos implementados são baseados em crescimento de regiões, a saber.

  - Crescimento de regiões por busca em todas as direções;
  - Crescimeto de regiões segundo a varredura;

3.2 Linguagem Utilizada

A linguagem escolhida para execução do projeto foi C++. Dentre as principais características que motivaram tal escolha estão: código multiplataforma, paradigma de orientação à objeto e incorporação da biblioteca gráfica *wxWindows*.

A *wxWindows* é uma biblioteca gráfica para C++. A facilidade dessa biblioteca em trabalhar com imagens constitui a principal motivação de sua utilização. Ela consegue ler vários formatos de imagens e convertê-los para um tipo genérico *wxImage*, permitindo a manipulação dos dados da imagem. Mais detalhes sobre a *wxWindows* pode ser visto em [wxWindows.org].

O diagrama de classes baseado na linguagem *UML* foi a forma proposta para modelagem do sistema. Mais detalhes sobre *UML* em [BOOCH et al.].

3.3 Equipamentos Utilizados

Os equipamentos utilizados foram computadores com microprocessadores Duron (AMD) 1,0 GHz, 128 mb RAM, 40 Gb, com os sistemas operacionais *Windows 2000 professional (Windows ®) e Red Hat 8 com microprocessadores Pentium III (Intel) 500 MHz, 128 mb RAM, 30 Gb, com o sistema operacional Red Hat 7.2*. Os computadores pertencem ao Laboratório de Computação do Departamento de Ciência da Computação da Universidade Federal de Lavras (DCC/UFLA).

3.4 Teste dos Algoritmos

A avaliação dos algoritmos implementados foi de forma subjetiva e visual, ou seja, dada uma imagem como entrada, verifica-se após processamento se a imagem de saída é a desejada.

Capítulo 4

Resultados e Discussões

Este capítulo apresenta os resultados e as discussões relativos ao projeto, apresentando as classes que compõe a biblioteca, a modelagem do *software* e a disussão sobre o funcionamento dos algoritmos.

4.1 A Biblioteca

Nesta seção está a descrição de todas as classes que compõem a biblioteca para processamento digital de imagens. A descrição engloba os arquivos que devem ser incluídos, bem como os métodos que o usuário pode utilizar.

4.1.1 Classe *clIContrast*

Esta classe contém as funções de tranferências para melhoramento de contraste.
Arquivo a ser incluído: *clIContrast.h*

Métodos da Classe

- *clIContrast::clIContrast()*
  Construtor *default* da classe *clIContrast*, esse construtor inicializa os atributos da classe deixando os objetos em seu estado consistente.

- *clIContrast::clIContrast(unsigned char *data, int _size)*
  Construtor parametrizado da classe, iniciliza os objetos com valores passados.

  data = Ponteiro para unsingned char, contém os dados (pixels) da imagem.

  _size = Tamanho de data

- *clIContrast::c̃lIContrast()*
  Destrutor da classe.

- *unsigned char *clIContrast::PDILinearFunction(unsigned char a, unsigned char b)*
  Método que aplica a função de transferência linear nos dados da imagem.

  a - Fator de ajuste da função

  b - Fator de ajuste da função, onde estes valores estão no intervalo de [0 a 255]

- *unsigned char *clIContrast::PDISquareRootFunction(unsigned char a)*
  Método que aplica a função de transferência raiz quadrada nos dados da imagem.

  a - Fator de ajuste da função, valores no intervalo de [0 a 255]

- *unsigned char *clIContrast::PDILogarithmFunction(unsigned char a)*
  Método que aplica a função de transferência logarítma nos dados da imagem.

  a - Fator de ajuste da função, valores no intervalo de [0 a 255]

- *unsigned char *clIContrast::PDIInverseFunction(unsigned char a, unsigned char b)*
  Método que aplica a função de transferência inversa nos dados da imagem,

  a - Fator de ajuste da função

  b - Fator de ajuste da função, onde estes valores estão no intervalo de [0 a 255]

4.1.2 Classe *clPhilter*

Esta classe contém os algoritmos de filtragem no espaço, algoritimos para detecção de bordas e algoritmos para filtragem de ruidos.

Arquivo a ser incluído: *clPhilter.h*

Métodos da Classe

- *clPhilter::clPhilter()*
  Construtor *default* da classe, inicializa os atributos da classe com valores *default* deixando-os em seu estado consistente.

- *clPhilter::clPhilter(unsigned int width, unsigned int height, unsigned char *data, unsigned char *newData)*
  Construtor parametrizado da classe, inicializa os dados da classe com valores passado.

  width - Largura da imagem em *pixels*

height - Altura da imagem em *pixels*

data - Estrutura que contém os dados da imagem (*pixels*), esta estrutura é um vetor simulando uma matriz, cada posição do vetor está codificada pela seguinte fórmula $x * Altura + y$, ou seja, $f(x, y) = data[x * altura + y]$

newData - Estrutura onde será armazenado os dados de saída, codificado da mesma forma que *data*

- clPhilter::c̃lPhilter()
  Destrutor da classe.

- *unsigned char *clPhilter::PDIOperatorGradientRoberts()*
  Algoritmo detector de bordas, utilizando operador gradiente de Roberts.

- *unsigned char *clPhilter::PDIOperatorGradientSobel()*
  Algoritmo detector de bordas, utilizando operador gradiente de Sobel.

- *unsigned char *clPhilter::PDIAveragePhilter()*
  Filtro que suaviza ruído utilizando o filtro da média.

- *unsigned char *clPhilter::PDIMediumPhilter()*
  Filtro que suaviza ruído utilizando o filtro da mediana.

4.1.3 Classe *clGRSweeping*

Classe que contém o algoritmo de segmentação de imagens segundo a varredura.
Arquivo a ser incluído: *clGRSweeping.h*

Métodos da Classe

- *clGRSweeping::clGRSweeping()*
  Construtor *default* da classe, inicializa os atributos com valores *default* deixando-os em seu estado consistente.

- *clGRSweeping::clGRSweeping(unsigned int _threshold, unsigned int width, unsigned int height, unsigned char *data, unsigned int *cPtr)*

  _threshold - Limiar de similaridade entre os pixels.

  width - Largura da imagem em *pixels*

  height - Altura da imagem em *pixels*.

  data - Estrutura que contém os dados da imagem (*pixels*), esta estrutura é um vetor simulando uma matriz, cada posição do vetor está codificada pela seguinte fórmula $x * Altura + y$, ou seja, $f(x, y) = data[x * altura + y]$.

cPtr - Estrutura que contém as classes dos pixels da imagem, codificada da mesma forma que *data*.

- *clGRSweeping::~clGRSweeping()*
  Destrutor da classe.

- *unsigned int *clGRSweeping::PDISweeping(unsigned int x, unsigned int y)*
  Algoritmo que cresce uma região a partir de uma coordenada espacial inicial (x, y).

4.1.4 Classe *clGRAllDirections*

Esta classe contém o algoritmo que identifica todas as regiões, através da busca por todas as direções.

Arquivo a ser incluído: *clGRAllDirections*

Métodos da Classe

- *clGRAllDirections::GRAllDirections()*
  Construtor *default* da classe, inicializa os atributos com valores *default* deixando-os em seu estado consistente.

- *clGRAllDirections::clGRAllDirections(int _threshold, unsigned int width, unsigned int height, unsigned char *data, unsigned int *cPtr)*

  _threshold - Limiar de similaridade entre os *pixels*.

  width - Largura da imagem em *pixels*.

  height - Altura da imagem em *pixels*.

  data - Estrutura que contém os dados da imagem (*pixels*). Esta estrutura é um vetor simulando uma matriz, cada posição do vetor está codificada pela seguinte fórmula $x * Altura + y$, ou seja, $f(x, y) = data[x * altura + y]$

  cPtr - Estrutura que contém as classes dos pixels da imagem, codificada da mesma forma que *data*.

- *clGRAllDirections::~clGRAllDirections()*
  Destrutor da classe.

- *unsigned int *clGRAllDirections::PDISerchAllDirections()*
  Algoritmo que cresce regiões fazendo busca em todas as direções.

- *unsigned int clGRAllDirections::PDIGetK() const*
  Função que retorna a maior classe encontrada na imagem.

4.2 Algoritmos de Contraste

Esta seção mostra os resultados dos algoritmos para melhoramento de contraste.

4.2.1 Função de Mapeamento Linear

A função linear pode ser usada para redistribuir o histograma da imagem de forma linear. Esta função apresentou bons resultados em imagens escuras, nas quais torna-se difícil o discernimento dos objetos em cena. O resultado dessa função é uma melhoria no contraste (conforme exemplificado na Figura 4.1), que é explicada pelo comportamento da função linear. Todos os pontos da imagem são mapeados para outros valores, seguindo o crescimento da reta linear.

4.2.2 Função de Mapeamento Raiz Quadrada

A função raiz quadrada é utilizada para melhorar o contraste em áreas escuras da imagem. A (Figura 4.2) mostra o resultado da função raiz quadrada com fator de ajuste $a = 16$. Esta função aumenta a diferença entre os *pixels* que estão no intervalo onde o gráfico da função apresenta um maior crescimento, ou seja, áreas escuras da imagem.

4.2.3 Função de Mapeamento Logarítmica

A função logaritmica apresenta comportamento semelhante a raiz quadrada, porém realçando um intervalo menor de áreas escuras da imagem, pois o intervalo de crescimento desta função é menor que a raiz quadrada. A medida que a imagem se aproxima do branco, a função logaritma não encontra diferenças significativas entre os pixels. A (Figura 4.3) mostra o resultado da função logaritma com fator de ajuste $a = 105$, para a mesma figura processada pela raiz quadrada.

4.2.4 Função de Mapeamento Inversa

A função inverte os valores dos *pixels* da imagem de entrada (Figura 4.4). Os *pixels* de valor 0 são mapeados para 255 e vice-versa. Os valores intermediários são mapeados para outros valores intermediários, seguindo o comportamento da reta da função inversa.

4.3 Algoritmos de Filtragem

Esta seção mostra os resultados dos filtros espaciais.

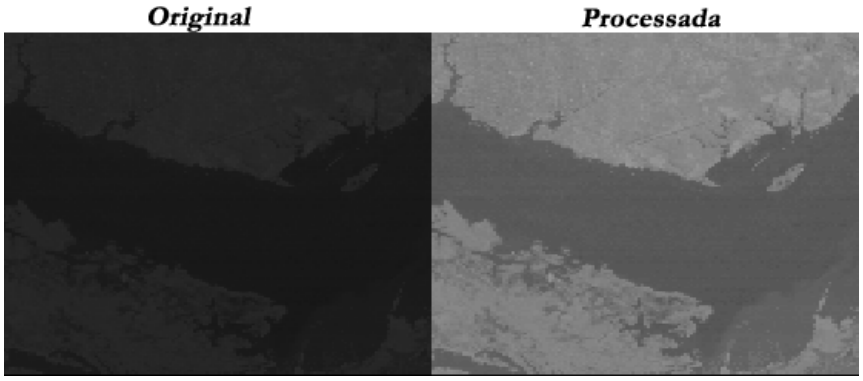

Figura 4.1: Resultado da função linear

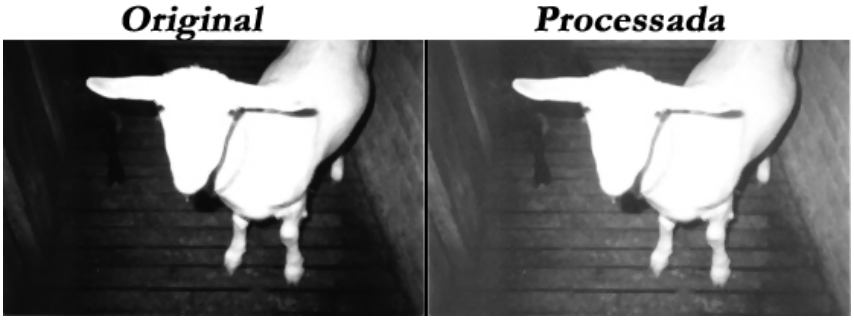

Figura 4.2: Resultado da função raiz quadrada

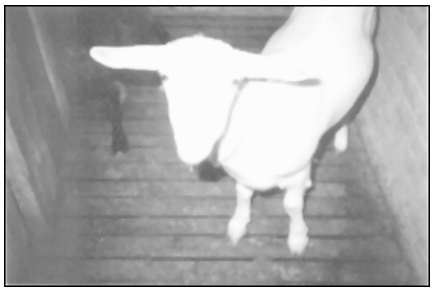

Figura 4.3: Resultado da função logaritmica

4.3.1 Filtro de Roberts

Este é um dos filtros mais antigos e simples usados para detectar bordas na imagem utilizando operador gradiente (Figura 4.5). O filtro de Roberts usa o operador

Figura 4.4: Resultado da função inversa

gradiente para detectar bordas. O operador marca áreas constantes da imagem com valores baixos e áreas onde a taxa de variação da derivada é maior com valores altos. Originalmente áreas constantes da imagem são marcadas com preto e as bordas com branco, porém, no exemplo citado, a imagem foi invertida para dar maior ênfase nas bordas.

4.3.2 Filtro de Sobel

O filtro de sobel é mais sofisticado do que o de Roberts. O princípio de funcionamento do operador de Sobel é igual ao de Roberts, porém a máscara de deslocamento é maior. Ele consegue encontrar mais detalhes na imagem. É possível perceber que este marca as bordas com uma intensidade maior (Figura 4.6).

4.3.3 Filtro da Média

O filtro da média apenas suaviza os ruídos da imagem. Ele redistribui os ruídos para os pixels vizinhos (Figura 4.7). A grande desvantagem desse filtro é a suavização de detalhes como as bordas da imagem. Este filtro substitui um ponto da imagem por um valor que é a média aritmética desse ponto e sua vizinhança 8. Se esse ponto for um ruído, ele será redistribuído entre os vizinhos.

4.3.4 Filtro da Mediana

O filtro da mediana é melhor aplicado em imagens onde os ruídos ocorrem de forma aleatória. Esse filtro depende de boas informações dos *pixels* vizinhos a um ponto com ruído (Figura 4.8). Este filtro coloca em um conjunto um ponto da imagem e sua vizinhança 8 e posteriormente os ordena em ordem crescente de nível de cinza. O novo valor do ponto é o valor que está na mediana deste conjunto. Se o ponto escolhido for um ruído ele tenderá a ficar em um dos extremos do conjunto, sendo eliminado ao ser substituído pelo valor da mediana.

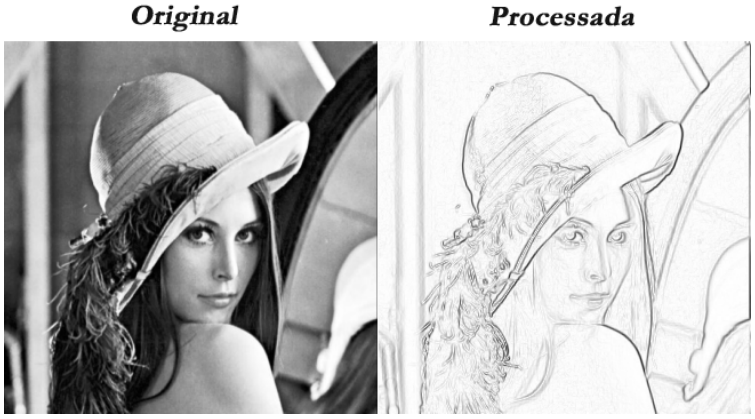

Figura 4.5: Resultado do operador gradiente de Roberts

Figura 4.6: Resultado do operador gradiente de Roberts

4.4 Algoritmos de Segmentação

Esta seção apresenta os métodos de segmentação baseado em crescimento de regiões.

4.4.1 Segmentação Segundo a Varredura

O método de segmentação segundo a varredura cresce uma região a partir de um ponto inicial da imagem. Dado um limiar (critério de similaridade entre *pixels*) e a coordenada espacial do ponto inicial (xi, yi) o algoritmo tenta encontrar na imagem todos os pontos que podem ser incorporados à região do ponto inicial. O limiar adotado neste projeto é a intensidade em escala de cinza. Assim, um ponto pertence a região do ponto inicial se $|f(x, y) - f(xi, yi)|$ for menor ou igual ao limiar passado. Esse algoritmo é melhor aplicado quando o objetivo é encontrar

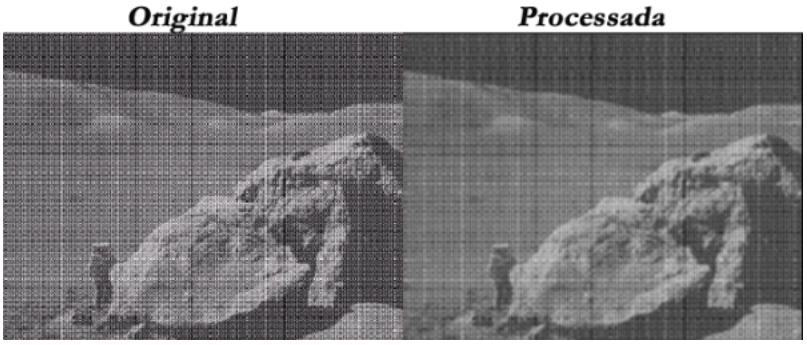

Figura 4.7: Resultado do filtro da média

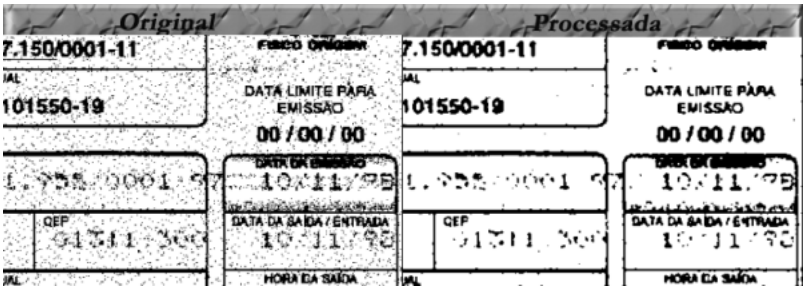

Figura 4.8: Resultado do filtro da mediana

regiões semelhantes não ligadas na imagem. Este algoritmo rotula o ponto inicial e percorre toda a imagem, colocando o mesmo rótulo nos pontos que são similares ao ponto inicial. Ao final do processo pinta-se esses pontos. A (Figura 4.9) mostra a segmentação de uma imagem de satélite a partir da coordenada espacial $(xi = 224, yi = 126)$ e $limiar = 30$, observa-se que o algoritmo encontrou todas as regiões aquáticas na imagem.

4.4.2 Segmentação por Busca em Todas as Direções

A segmentação por busca em todas as direções encontra todas a regiões que estão ligadas na imagem. Este algoritmo escolhe um ponto na imagem, o rotula e começa a crescer a região a partir desse ponto. Quando não mais conseguir expandir a região, ele escolhe outro ponto e repete o processo até não houver mais pontos para serem rotulados. A figura 4.10 ilustra o resultado da segmentação com $limiar = 30$.

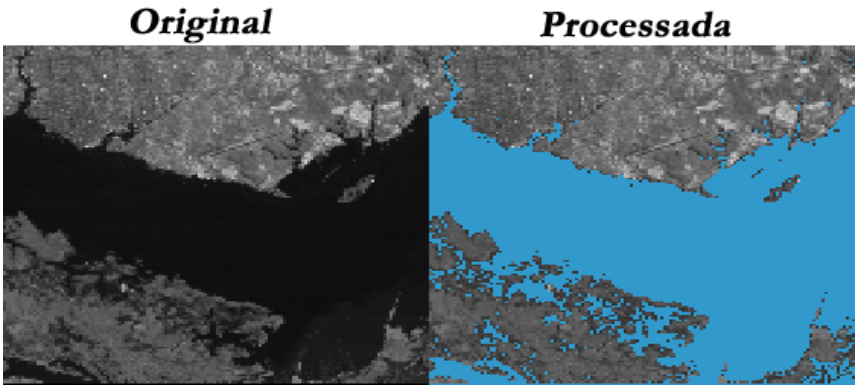

Figura 4.9: Resultado da segmentação segundo a varredura

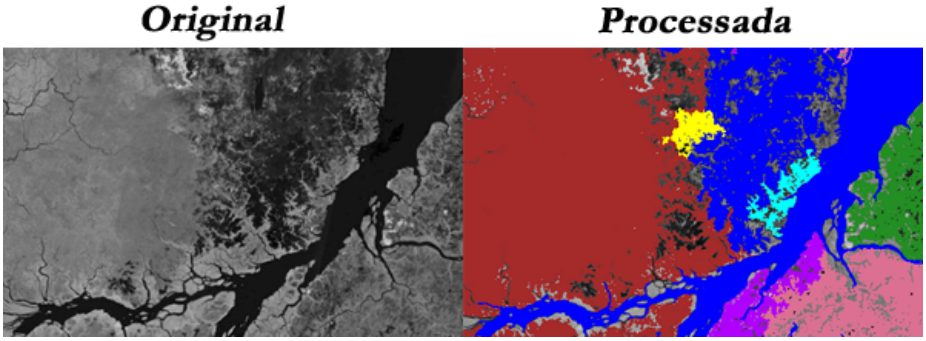

Figura 4.10: Resultado da segmentação por todas as direções

4.5 O *software*

A figura 11 mostra o diagrama de classes do *software* **wxPDI** por questões de legibilidade, a descrição do diagrama encontra-se no apêndice A. A (Figura 4.12) apresenta a tela principal do *software*.

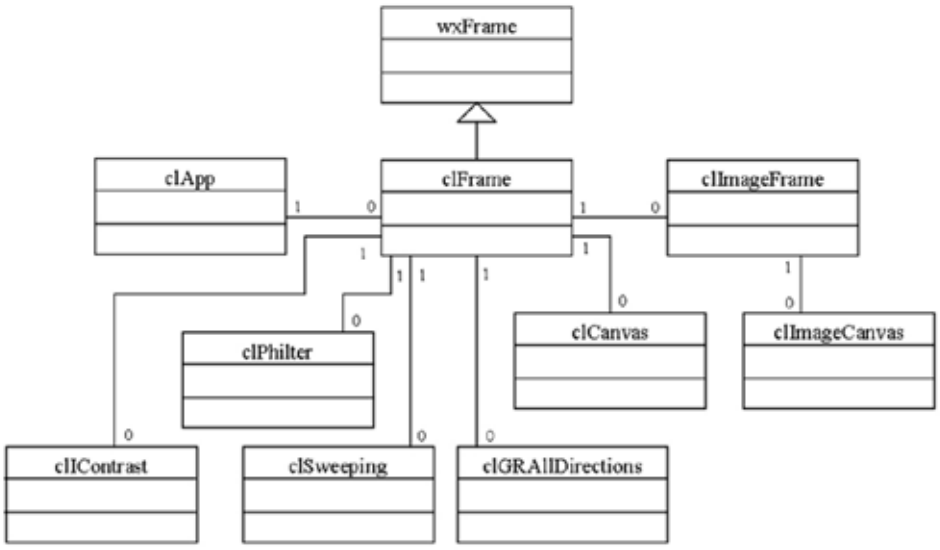

Figura 4.11: Diagrama de classes

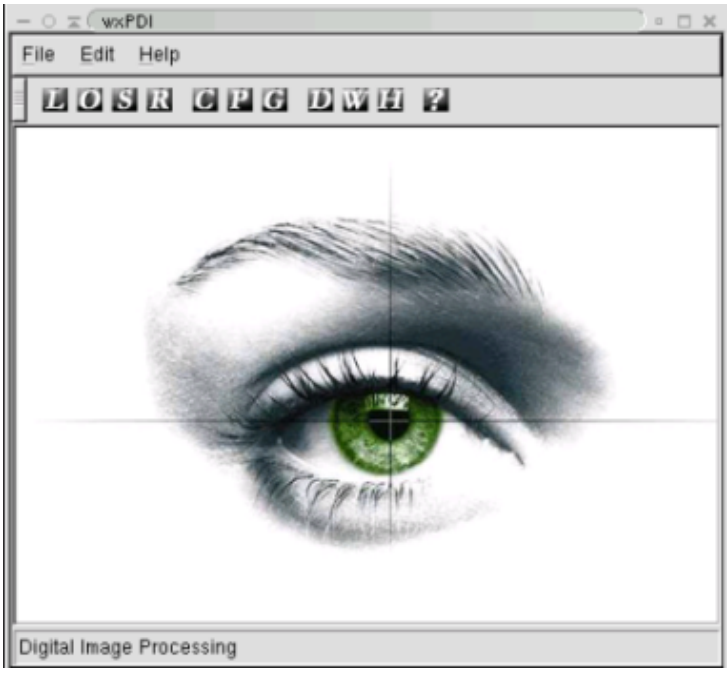

Figura 4.12: Tela principal do wxPDI

Capítulo 5

Conclusão

O presente projeto confirmou o grande potencial da aplicação de técnicas de processamento digital de imagens com diferentes finalidades, sejam elas o realce através do melhoramento de contraste e filtragem ou a segmentação.

Como foi visto no Capítulo Resultados e Discussões os algoritmos mostraram-se bastante eficientes em relação ao seu funcionamento, confirmando a revisão de literatura. Pode-se perceber também que a utilização dos mesmos depende dos resultados que deseja-se obter na imagem. Em muitos casos esta análise é empírica e subjetiva.

Terminando este trabalho espera-se que tanto a biblioteca quanto o *software* *wxPDI* sejam de grande utilidade.

Capítulo 6

Trabalhos Futuros

Dando continuidade ao projeto, ficam como sugestões de trabalhos futuros:

- Implementação de histograma;
- Métodos de filtragem e domínio da freqüência;
- Métodos de segmentação baseados em perseguição de contorno e pesquisa em grafo;
- Métodos de classificação;

Referências Bibliográficas

[Banon (2000)] BANON, G. J. F. *Formal Introduction to digital image processing*. INPE, São José dos Campos JULHO 2000.

[Bastos] TÉCNICAS DE SEGMENTAÇÃO DE IMAGENS PARA RECUPERAÇÃO DE INFORMAÇÕES VISUAIS. *Disponível em http://http://atlas.ucpel.tche.br/ vbastos/*. Visitado em 05/11/2003.

[BOOCH et al.] BOOCH. G. & RUMBAUCH J. & JACOBSON I. *UML - Guiado usuário*. Rio de Janeiro, Campus, 2000.

[Brito & Carvalho (1998)] BRITO, S. F. & CARVALHO, J. M. *Sistemas de Processamento digital de imagens para Fins didáticos/ciêntificos: Estudo, seleção e implementação de algoritmos de segmentação*. Relatorio de iniciação ciêntífica. Agosto de 1998.

[Darsa (1994)] DARSA, L. *Deformação e Metarmofose de Objetos Gráficos*. Departamento de Informática. Dissertação de Mestrado pela Pontificia Universidade Católica do Rio de Janeiro. 14 de junho 1994.

[Escuri (2002)] ESCURI, A. E. *Fundamentos da Imagem Digital*. TecGraf /PUC-Rio. Setembro de 2002.

[Escuri (2004)] ESCURI, A. E. *Filtros Interativos para Imagens Digitais no Domínio da Frequência*. Dissertação de Mestrado Pontificia Universidade Catolica do Rio de Janeiro. 14 de setembro de 1994.

[Facon (2002)] FACON, J. *Processamento e Análise de Imagens*. Pontificia Universidade Católica do Paraná. Curso e Mestrado em Informatica Aplicada. FEVEREIRO 21, 2002.

[Facon (2002)] FACON, J. *Princípios Básicos da Visão por Computador e do Processamento de Imagens*. Pontificia Universidade Católica do Paraná. Curso e Mestrado em Informatica Aplicada, 2002.

[Ferraz (1998)] FERRAS, M.C. *Codificação de Imagens*. IMPA, Rio de janeiro. 28 de junho de 1998.

[Goldenstein (1997)] Goldenstein, S. K. *Metarmorfose de Sons e Aplicações*. 25 de agosto de 1997.

[Gonzales & Woods (1992)] GONZALES, R. C. & WOODS, R. E. *Digital Image Processing*. University of Tennessee Perceptics Corporation, 1992.

[Hadad (2000)] HADAD, R. M. *Identificação de Padrões em Imagens de Satélite - Formas circulares*. Tese de doutorado, Universidade Federal de Minas Gerais. 20 de setembro de 2000.

[Maria (2000)] MARIA, L. G. F. *Processamento Digital de Imgens*. INPE, Junho de 2000.

[Marta (1998)] MARTA. D. S. *Algoritmos para detecção de bordas*. UFSC - dezembro de 1998.

[Mascarenhas & Velasco (1989)] MASCARENHAS, N. A. & VELASCO, F. R. D. *Processamento Digital de Imagens*. Ministerio da ciência e Tecnologia - MCT. Instituto de Pesquisa Espaciais - INPE. Janeiro de 1989.

[Mota (1999)] MOTA, C. *Processamento Geométrico de Imagens*. IMPA, 14 de outubro de 1999.

[Oliveira & Carvalho (1998)] OLIVEIRA, J. J. J & CARVALHO, J. M. *Sistemas de Processamento digital de imagens para Fins didáticos/ciêntificos: Estudo, seleção e implementação de algoritmos de Filtragem Espacial*. Relatorio de iniciação ciêntífica. Julho de 1998.

[Oliveira (2001)] OLIVEIRA, C. J. S. *Classificação de Imagens coletadas na Web*. Dissertação de Mestrado, Universidade Federal de Minas Gerais. 20 de agosto de 2001.

[Ramos (2000)] RAMOS, O. T. H. *Análise Comparativa entre os principais formatos de armazenamento de imagens*. Dissertação de Mestrado pela Universidade Presbiteriana Mackenzie. São Paulo, dezembro de 2000.

[Schneider (2001)] SCHNEIDER, B. O. *Apostila de Computação Gráfica*. Departamento de Ciência da Computação. Universidade Federal de Lavras. 9 de Agosto de 2001.

[Silva (1994)] SILVA, B. C. *Deformação e Metamorfose de Imagens*. Departamento de Informática, Dissertação de Mestrado pela Pontificia Universidade Católica do Rio de Janeiro. 7 de junho de 1994.

[Sobreiro (1998)] SOBREIRO, M. V. R. *Quantização de Imagens*. 22 de setembro de 1998.

[Young & Gerbrands & Van Vliet (1998)] YOUNG, I. T. & GERBRANDS, J. J. & Van Vliet, L. J. *Fundamentals of Image Processing*. Delft University of Tecnology, 1998.

[INPE] ISTITUTO NACIONAL DE PESQUISA ESPACIAIS. *Disponível em http://www.dpi.inpe.br/pdi.html* . Visitado em 13/06/03.

[NPDI UFMG] NÚCLEO DE PROCESSAMENTO DIGITAL DE IMAGENS. *Disponível em http://www.npdi.dcc.ufmg.br/*. Visitado em 13/06/03.

[wxWindows.org] WXWINDOWS. *Disponível em http://www.wxwindows.org/*. Visitado em 13/11/03.

Apêndice A

Diagrama de Classes

Este anexo apresentará a descrição de todas as classes do *wxPDI*.

==========

A.1 Classe *clContrast*

A.1.1 Atributos

- dataPtr : unsigned char * = NULL
- size : int = 0

A.1.2 Métodos

+ clIContrast()
+ clIContrast(unsigned char *data, int _size)
+ ~clIContrast()
+ PDILinearFunction(unsigned char a, unsigned char b) : unsigned char *
+ PDISquareRootFunction(unsigned char a) : unsigned char *
+ PDILogarithmFunction(unsigned char a) : unsigned char *
+ PDIInverseFunction(unsigned char a, unsigned char b) : unsigned char *

==========

A.2 Classe *clPhilter*

A.2.1 Atributos

- nWidth : unsigned int = 0
- nHeight : unsigned int = 0
- dataPtr : unsigned char * = NULL
- newDPtr : unsigned char * = NULL

A.2.2 Métodos

+ clPhilter()
+ clPhilter(unsigned int width, unsigned int height, unsigned char *data, unsigned char *newData)
+ ~clPhilter()
+ PDIOperatorGradientRoberts() : unsigned char *
+ PDIOperatorGradientSobel() : unsigned char *
+ PDIAveragePhilter() : unsigned char *
+ PDIMediumPhilter() : unsigned char *
- GetPosition(unsigned int x, unsigned int y) : unsigned int
- InitialArray() : void
- Average(unsigned int x, unsigned int y) : unsigned char
- Medium(unsigned int x, unsigned int y) : unsigned char

A.3 Classe *clGRSeeping*

A.3.1 Atributos

- threshold : int = 0
- nWidth : unsigned int = 0
- nHeight : unsigned int = 0
- dataPtr : unsigned char * = NULL
- classPtr : unsigned int * = NULL
- k : unsigned int = 0

A.3.2 Métodos

+ clGRSweeping()
+ clGRSweeping(unsigned int _threshold, unsigned int width, unsigned int height, unsigned char *data, unsigned int *cPtr)
+ ~clGRSweeping()
+ PDISweeping(unsigned int x, unsigned int y) : unsigned int *
- InitialArray() : void
- GetPositon(int x, int y) : unsigned int
- VThreshold(unsigned int x, unsigned int y, unsigned int xi, unsigned int yi) : bool
- VNeighbor(unsigned int x, unsigned int y) : bool

A.4 Classe *clGRSeeping*

A.4.1 Atributos

- threshold : int = 0
- nWidth : unsigned int = 0
- nHeight : unsigned int = 0
- dataPtr : unsigned char * = NULL
- classPtr : unsigned int * = NULL
- k : unsigned int = 0

A.4.2 Métodos

+ clGRSweeping()
+ clGRSweeping(unsigned int _threshold, unsigned int width, unsigned int height, unsigned char *data, unsigned int *cPtr)
+ ~clGRSweeping()
+ PDISweeping(unsigned int x, unsigned int y) : unsigned int *
- InitialArray() : void
- GetPositon(int x, int y) : unsigned int
- VThreshold(unsigned int x, unsigned int y, unsigned int xi, unsigned int yi) : bool
- VNeighbor(unsigned int x, unsigned int y) : bool

A.5 Classe *clGRAllDirecitions*

A.5.1 Atributos

- threshold : int = 0
- nWidth : unsigned int = 0
- nHeight : unsigned int = 0
- dataPtr : unsigned char * = NULL
- classPtr : unsigned int * = NULL
- pixel : unsigned int = 0
- setX : deque< int > * = NULL
- setY : deque< int > * = NULL
- k : unsigned int = 0

A.5.2 Métodos

+ clGRAllDirections();
+ clGRAllDirections(int _threshold, unsigned int width, unsigned int height, unsigned char *data, unsigned int *cPtr)

+ ~clGRAllDirections()
+ PDISerchAllDirections() : unsigned int *
+ PDIGetK() : unsigned int
- InitialArray() : void
- VThreshold(unsigned int x, unsigned int xi) : bool
- VNeighbor(unsigned int x) : bool
- VLabel(unsigned int x) : bool
- VClass(unsigned int x, unsigned int xi) : void
- ClassifyNeighborhood4(unsigned int x) : void ==========

A.6 Classe *clCanvas*

A.6.1 Atributos

- imgBGround : wxBitmap * = NULL
- mather : wxFrame * = NULL
- image : wxImage * = NULL
- width : int = NULL
- height : int = NULL
- x : int = 0
- y : int = 0

A.6.2 Métodos

+ clCanvas()
+ clCanvas(wxWindow *parent, wxWindowID id, const wxPoint &pos, const wx-
Size &size, const wxBitmap &bitmap)
+ ~clCanvas()
+ OnPaint(wxPaintEvent &event) : void
+ SetBitmap(const wxBitmap &bitmap, bool k) : void
+ OnEventMouse(wxMouseEvent &event) : void ==========

A.7 Classe *clImageCanvas*

A.7.1 Atributo

imgBGround : wxBitmap = image

A.7.2 Métodos

+ clImageCanvas()
+ clImageCanvas(wxWindow *parent, wxWindowID id, const wxPoint &pos,

const wxSize &size, const wxBitmap &bitmap)
+ ~clImageCanvas()
+ OnPaint(wxPaintEvent &event) : void
==========

A.8 Classe *clImageFrame*

A.8.1 Atributo

- canvas : clImageCanvas * = NULL

A.8.2 Métodos

+ clImageFrame()
+ clImageFrame(wxWindow *parent, wxWindowID id, const wxString &title, int
xpos, int ypos, int width, int height, const wxBitmap &bitmap, bool up)
+ ~clImageFrame()
==========

A.9 Classe *clFrame*

A.9.1 Atributos

- menuBar : wxMenuBar * = NULL
- menuFile : wxMenu * = NULL
- menuEdit : wxMenu * = NULL
- menuHelp : wxMenu * = NULL
- m_tbar : wxToolBar * = NULL
- m_smallToolbar : bool = false
- m_horzToolbar : bool = false
- width : int = 0
- height : int = 0
- data : unsigned char * = NULL
- data2 : unsigned char * = NULL
- imageLoad : wxImage * = NULL
- imageBG : wxImage * = NULL
- canvas : clCanvas * = NULL
- cls : unsigned int * = NULL
- k : unsigned int = 0
- newData : unsigned char * = NULL
- red : unsigned char = 128
- green : unsigned char = 128

- blue : unsigned char = 128
- name : wxString * = NULL

A.9.2 Métodos

+ clFrame()
+ clFrame(wxWindow *parent, wxWindowID id, const wxString &title, int xpos, int ypos, int width, int heigth)
+ ~clFrame()
+ OnLoadImage(wxCommandEvent &event) : void
+ OnFileOpen(wxCommandEvent &event) : void
+ OnFileSave(wxCommandEvent &event) : void
+ OnRefresh(wxCommandEvent &event) : void
+ OnFileExit(wxCommandEvent &event) : void
+ OnContrast(wxCommandEvent &event) : void
+ OnPhilter(wxCommandEvent &event) : void
+ OnSegmentation(wxCommandEvent &event) : void
+ OnDrawClass(wxCommandEvent &event) : void
+ OnWriteClassFile(wxCommandEvent &event) : void
+ OnChooseColour(wxCommandEvent &event) : void
+ OnHelpAbout(wxCommandEvent &event) : void
+ RecreateToolbar() : void
- AllocateData(int choose) : void
- GetPosition(int x, int y) : int
- DrawImage(unsigned int k) : void
- DrawImagePhilter() : void

A.10 Classe *clApp*

A.10.1 Atributo

frame : wxFrame * = NULL

A.10.2 Métodos

+ OnInit() : bool