Campus de São Carlos

"DETECÇÃO DE FORMAS CIRCULARES
UTILIZANDO A TRANSFORMADA DE HOUGH,
DESENVOLVIDA NO AMBIENTE WINDOWS"

Autor:

Eng. Paulo César Lucena Bentes

Orientadora:

Prof^a. Dr^a. M^a Stela Veludo de Paiva

UNIVERSIDADE DE SÃO PAULO

**ESCOLA DE ENGENHARIA
DE SÃO CARLOS**

✓

DETECÇÃO DE FORMAS CIRCULARES UTILIZANDO A TRANSFORMADA DE HOUGH, DESENVOLVIDA NO AMBIENTE WINDOWS

Eng. PAULO CÉSAR LUCENA BENTES

Dissertação de Mestrado apresentada à Escola de Engenharia de São Carlos, da Universidade de São Paulo, como parte dos requisitos para obtenção do Título de Mestre em Engenharia Elétrica.

ORIENTADORA: Prof^a. Dr^a. M^a Stela V. de Paiva

DEDALUS - Acervo - EESC

31100016951

São Carlos

1997

Class.	TESE
Curr.	0668
Tombo	T078/97

Área: Engenharia Elétrica.

✱ 0747789

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca - EESC-USP

B475d Bentes, Paulo César Lucena
Detecção de formas circulares utilizando a
Transformada de Hough, desenvolvida no ambiente
Windows / Paulo César Lucena Bentes. -- São
Carlos, 1997.

Dissertação (Mestrado). -- Escola de Engenharia
de São Carlos-Universidade de São Paulo, 1997.
Área: Engenharia Elétrica
Orientador: Prof^a. Dr^a. Maria S. V. de Paiva

1. Detecção de formas circulares. 2.
Transformada de Hough. I. Título

FOLHA DE APROVAÇÃO

Dissertação defendida e aprovada em 18-3-1997
pela Comissão Julgadora:

Prof.ª Doutora **MARIA STELA VELUDO DE PAIVA (Orientadora)**
(Escola de Engenharia de São Carlos - Universidade de São Paulo)

Prof. Doutor **ADILSON GONZAGA**
(Escola de Engenharia de São Carlos - Universidade de São Paulo)

Prof. Doutor **MARCOS LUIZ MUCHERONI**
(Universidade Federal de São Carlos - UFSCar)

Prof. Doutor. **GERALDO ROBERTO M. DA COSTA**
Coordenador da Área de Engenharia Elétrica

Prof. Assoc. **JOSE CARLOS A. CINTRA**
Presidente da Comissão de Pós-Graduação

A meu pai e em memória de minha mãe.

AGRADECIMENTOS

À Professora Maria Stela Veludo de Paiva pela orientação fornecida durante a elaboração deste trabalho.

Aos Professores Adilson, Evandro e Murilo. Aos Técnicos de Laboratório Mateus e José Luis, à Especialista de Laboratório de Nível Superior Roseli, e aos colegas Celso, Ernany, Evelina, Dulce, Stilante, Sebastião Sérgio, Mary Glória, Mauro, Waldemir, Tânia, Hélio Glauco, Clébia, Monaco e Manoel pela ajuda e amizade.

À minha namorada Maria pela ajuda e apoio.

SUMÁRIO

LISTA DE FIGURAS.....	i
LISTA DE TABELAS.....	viii
RESUMO.....	ix
ABSTRACT.....	x
1 INTRODUÇÃO.....	1
2 MÉTODOS PARA IDENTIFICAÇÃO DE FORMAS.....	3
2.1 Introdução.....	3
2.2 A Transformada de Hough.....	4
2.3 Método do <i>Template Matching</i>	6
2.4 Métodos para detectar formas circulares.....	7
2.5 Métodos para detectar formas circulares como caso particular das elípticas.....	19
2.6 Métodos para detecção de arcos de circunferências e simetrias rotacionais.....	23
2.7 Métodos que utilizam a lógica Fuzzy.....	24
2.8 Métodos que utilizam dispositivos especiais.....	27
2.9 Conclusão.....	28
3 O AMBIENTE DE PROCESSAMENTO.....	29
3.1 Introdução.....	29
3.2 Disposição da imagem na memória.....	29
3.3 As estruturas <i>headers</i> do arquivo MBID.....	34
3.4 Manipulação do arquivo de entrada da imagem.....	36
3.4.1 Leitura e gravação do arquivo MBID.....	36
3.4.2 Funções que extraem as características da imagem.....	38

3.5 A Visualização da Imagem.....	40
3.5.1 As mudanças de <i>palette</i>	41
3.5.1.1 Mudança da <i>palette</i> da janela.....	41
3.5.1.2 Mudança da <i>palette</i> para níveis de cinza.....	41
3.5.2 Um tratamento diferenciado para imagens de 16(dezesseis) cores.....	42
3.6 Conclusão.....	44
 4 O PRÉ-PROCESSAMENTO.....	 45
4.1 Introdução.....	45
4.2 A suavização da imagem.....	45
4.2.1 Suavização através da média.....	45
4.2.2 Suavização através da Gaussiana.....	46
4.3 Detectores de borda.....	48
4.3.1 Detecção de bordas por gradientes espaciais.....	49
4.3.2 O operador Sobel.....	52
4.3.3 O operador OU-exclusivo.....	53
4.4 A binarização da imagem.....	53
 5 DETECTORES	 56
5.1 Introdução.....	56
5.2 Utilizando a Transformada de Hough.....	57
5.3 Utilizando o método de <i>Template Matching</i>	59
5.4 A visualização do acumulador.....	60
 6 OS FILTROS PARA O ACUMULADOR.....	 62
6.1 Introdução.....	62
6.2 O Filtro Comparativo.....	62
6.3 O Filtro Equalizador.....	64
6.4 O Filtro Fuzzy.....	65
6.5 A visualização das formas circulares encontradas.....	68
6.6 A visualização dos dados das prováveis formas circulares.....	69

7 APRESENTAÇÃO DOS RESULTADOS.....	71
7.1 Resultados obtidos na representação das imagens contidas nos arquivos MBID.....	71
7.2 Mudança da <i>palette</i> para tons de cinza.....	71
7.3 Aplicação dos métodos de suavização na imagem original.....	72
7.4 Determinação das bordas na imagem.....	74
7.5 A detecção de formas circulares.....	80
7.5.1 Utilizando a Transformada de Hough.....	80
7.5.1.1 Aplicando apenas o Filtro Comparativo.....	81
7.5.1.2 Utilizando o Filtro Fuzzy antes do Filtro Comparativo.....	86
7.5.1.3 Aplicando o Filtro Equalizador antes do Comparativo.....	91
7.5.2 Utilizando o método de <i>Template Matching</i>	97
7.5.3 Obtendo os tempos de execução.....	98
CONCLUSÕES.....	110
REFERÊNCIAS BIBLIOGRÁFICAS.....	112
Apêndice 1 Programa fonte para detecção de formas circulares.....	119
Apêndice 2 Lista de Recursos para o Scanc.....	157
Apêndice 3 Definições para o Scanc.....	162
Apêndice 4 Algoritmo de Bresenham para desenhar círculos implementado em C	164
Apêndice 5 Programa fonte para obter os valores para formar o <i>template</i> da Gaussiana.....	165
Apêndice 6 Programa fonte para obter os valores para formar o <i>template</i> do filtro Fuzzy.....	167
Apêndice 7 Programa que calcula o número de pontos de um círculo construído a partir do Algoritmo de Bresenham para círculos.....	168
Seção 1.1 Declarações Globais.....	119

Seção 1.2 Criação da Janela.....	119
Seção 1.3 Funções para extrair informações do ponteiro criado a partir da leitura do arquivo de imagem.....	120
Seção 1.4 Funções de controle de caixas de diálogo.....	122
Seção 1.5 Funções para fazer ajustes na imagem.....	123
Seção 1.6 Funções de leitura e gravação.....	123
Seção 1.7 Função de mudanças e adaptações da palette.....	126
Seção 1.8 Funções usadas para suavizar a imagem.....	127
Seção 1.9 Funções para detecção de bordas na imagem.....	131
Seção 1.10 Funções que binarizam a imagem.....	134
Seção 1.11 Funções que fornecem a faixa de valores para os raios que se pretende detectar.....	135
Seção 1.12 Funções que realizam o acumulador da Transformada de Hough.....	137
Seção 1.13 Funções que realizam o acumulador para o Template Matching.....	139
Seção 1.14 Funções para filtrar o acumulador.....	142
Seção 1.15 Funções que permitem a visualização dos resultados na tela.....	147
Seção 1.16 Procedimento de Janela.....	149
Apêndice 8 Programa usado para contar o número de pontos de borda existentes numa imagem.....	169
Apêndice 9 Transformada de Hough para retas.....	175

LISTA DE FIGURAS

Fig. 2.1 - Forma circular contida numa imagem.....	5
Fig. 2.2- Os círculos paramétricos interceptam-se no centro da forma circular da imagem.....	6
Fig. 2.3 - Cruzamento entre cones no espaço de parâmetros que determinam a existência de uma forma circular na imagem	7
Fig. 2.4 - a) O <i>template</i> contendo a forma procurada que deve ser convoluído com a imagem; b) imagem contendo entre outras a forma procurada.....	8
Fig. 2.5 - Grade acumuladora quadrangular onde um grande número de círculos passa na célula correspondente ao centro da forma circular da imagem.....	9
Fig. 2.6 - Planos constituídos de grades acumuladoras quadrangulares onde um grande número de círculos passa na célula correspondente ao centro da forma circular da imagem.....	10
Fig. 2.7 - O centro é localizado através de um espaço acumulador bidimensional...	15
Fig. 2.8 - O valor do raio é encontrado através de um histograma obtido a partir da medida entre os pontos de borda da imagem e o valor do centro.....	15
Fig. 2.9 - a) Imagem contendo formas circulares concêntricas; b) espaço acumulador dos parâmetros <i>a</i> e <i>b</i> ; c) espaço acumulador dos parâmetros <i>a</i> e <i>r</i>	17
Fig. 2.10 - Gráfico de desempenho de cada método em relação ao número de formas circulares detectadas.....	18
Fig. 2.11 - Gráfico Comparativo do tempo gasto pelos diversos métodos.....	19
Fig. 2.12 - Geometria básica para encontrar o centro da elipse através de tangentes não paralelas.....	21
Fig. 3.1 - Diagrama de Blocos do Scanc.....	30
Fig. 3.2 - a) Byte dividido por oito <i>pixels</i> ; b) dividido por dois <i>pixels</i> ; c) byte correspondente a um <i>pixel</i>	31
Fig. 3.3 - Menu do Scanc acessando a mensagem de abertura de arquivo.....	36
Fig. 3.4 - Caixa de diálogo padrão do Windows para leitura.....	37

Fig. 3.5 - Diagrama do mapa de bits de uma imagem 6x1x16.....	39
Fig. 3.6 - Conteúdo do arquivo MBID pintado na janela.....	40
Fig. 3.7 - a) Seção da memória de uma imagem de 16 cores ou tons de cinza; b) Os bits correspondentes aos <i>pixels</i> dessa imagem passaram a ocupar apenas os bytes menos significativos.....	43
Fig. 3.8 - a) <i>Template</i> de aquisição dos quatro bits mais significativos em cada byte e, b) <i>template</i> de aquisição dos quatro bits menos significativos em cada byte.....	43
Fig. 4.1 - <i>Templates</i> circulares contendo Gaussianas em 2D com a) desvio padrão igual a 0,2; b) desvio padrão igual a 0,4 e c) desvio padrão igual a 1	47
Fig. 4.2 - Menu de escolha do desvio padrão para a Gaussiana.....	47
Fig. 4.3 - Escolha do detetor de borda.....	49
Fig. 4.4 - A Aproximação Derivativa não está centrada num ponto.....	51
Fig. 4.5 - Imagem de gradientes resultante da aplicação do operador Sobel.....	54
Fig. 4.6 - O programa requisita a inserção do valor do <i>threshold</i>	55
Fig. 5.1- Grade acumuladora onde os círculos paramétricos são traçados.....	56
Fig. 5.2 - Escolha da faixa de raios a ser pesquisada.....	59
Fig. 5.3 - Visualização da sobreposição do acumulador.....	61
Fig. 6.1 - Tela de escolha de faixa permissiva para o Filtro Comparativo através do menu.....	63
Fig. 6.2 - a) Forma circular com bordas grossas; b) forma do acumulador correspondente numa visão bidimensional.....	65
Fig. 6.3 - a) Forma circular com rotas alternativas; b) forma do acumulador correspondente numa visão bidimensional.....	66
Fig. 6.4 - Função $g(r)$	67
Fig. 6.5 - <i>Template</i> circular usado para implementar o Filtro Fuzzy.....	67
Fig. 6.6 - a) Forma do acumulador de uma forma circular com rota alternativa prestes a ser convoluído com o <i>template</i> , b) Forma do acumulador modificado após a convolução.....	68
Fig. 6.7 - Requisição por parte do usuário da visualização da forma circular encontrada.....	69

Fig. 6.8 - Representação da forma circular encontrada através do desenho de um círculo.....	70
Fig. 6.9 - Exemplo da visualização dos dados de uma forma circular encontrada.....	70
Fig. 7.1 - Exemplo da visualização através do Scanc de uma imagem de 256 cores.....	71
Fig. 7.2 - Exemplo da visualização através do Scanc de uma imagem de 16 cores.....	72
Fig. 7.3 - Imagem com a <i>palette</i> original.....	73
Fig. 7.4 - Imagem com <i>palette</i> trocada para tons de cinza.....	73
Fig. 7.5 - Imagem original com <i>palette</i> trocada par tons de cinza.....	74
Fig. 7.6 - Imagem suavizada pela média.....	74
Fig. 7.7 - Imagem suavizada através de uma função Gaussiana.....	75
Fig. 7.8 - Imagem obtida pela aplicação dos <i>templates</i> de Sobel.....	75
Fig. 7.9 - Imagem binarizada depois de aplicado os <i>templates</i> de Sobel.....	76
Fig. 7.10 - Imagem já binarizada obtida pela aplicação do operador OU-exclusivo.....	76
Fig. 7.11 - Imagem de borda - OU-exclusivo - suavizada com uma Gaussiana.....	76
Fig. 7.12 - Imagem de borda - OU-exclusivo - suavizada com a média.....	77
Fig. 7.13 - Imagem de borda suavizada com a média após a binarização com <i>threshold</i> = 28.....	77
Fig. 7.14 - Imagem de borda suavizada com a média binarizada e invertida.....	77
Fig. 7.15 - Imagem de borda através de Sobel obtida a partir de uma imagem suavizada pela Gaussiana.....	78
Fig. 7.16 - Imagem de borda através do OU-exclusivo obtida a partir de uma imagem suavizada pela Gaussiana.....	79
Fig. 7.17 - Imagem original contendo objetos encostados entre si.....	79
Fig. 7.18 - Imagem contendo as falhas na borda devido a falta de separação entre as peças.....	80
Fig. 7.19 - Gráfico da sobreposição do acumulador obtido pela aplicação da Transformada de Hough para detectar as prováveis formas circulares numa imagem.....	81
Fig. 7.20 - Resultado da detecção pelo Filtro Comparativo visualizado na imagem de bordas.....	82

Fig. 7.21 - Resultado da detecção pelo Filtro Comparativo visualizado na imagem original.....	82
Fig. 7.22 - Resultado da detecção pelo filtro na razão de 85%.....	83
Fig. 7.23 - Sobreposição do acumulador para raios na faixa 65 - 70.....	84
Fig. 7.24 - Formas circulares detectadas pelo Filtro Comparativo.....	84
Fig. 7.25 - Gráfico obtido a partir da sobreposição do acumulador alterado pelo Filtro Fuzzy.....	87
Fig. 7.26 - Forma circular encontrada pelo Filtro Comparativo a 98% depois de aplicado o Filtro Fuzzy no acumulador.....	87
Fig. 7.27 - Formas circulares encontradas pelo Filtro Comparativo a 90% depois de aplicado o Filtro Fuzzy no acumulador.....	88
Fig. 7.28 - Sobreposição do acumulador modificado pelo Filtro Fuzzy.....	89
Fig. 7.29 - Formas circulares encontradas pelo Filtro Comparativo na razão de 98% a partir do acumulador alterado pelo Filtro Fuzzy desenhadas na imagem original.....	89
Fig. 7.30 - Formas circulares encontradas pelo Filtro Comparativo na razão de 98% a partir do acumulador alterado pelo Filtro Fuzzy desenhadas na imagem de borda....	90
Fig. 7.31 - Formas circulares encontradas pelo Filtro Comparativo na razão de 98% a partir do acumulador original desenhadas na imagem de borda.....	90
Fig. 7.32 - Imagem original de uma mamografia.....	91
Fig. 7.33 - Imagem de bordas da mamografia.....	92
Fig. 7.34 - Sobreposição do acumulador obtida pela aplicação da Transformada de Hough na imagem de bordas da mamografia.....	92
Fig. 7.35 - Visualização na imagem original da forma circular encontrada quando usado o Filtro Comparativo na razão de 98%.....	93
Fig. 7.36 - Visualização na imagem original das formas circulares encontradas quando aplicado o Filtro Comparativo na razão de 95%.....	93
Fig. 7.37 - Visualização na imagem original das formas circulares encontradas quando aplicado o Filtro Comparativo na razão de 75%.....	94
Fig. 7.38 - Sobreposição do acumulador modificado pelo Filtro Equalizador.....	95
Fig. 7.39 - Formas circulares detectadas pelo Filtro Comparativo, na razão de 98%, aplicado no acumulador modificado pelo Filtro Equalizador.....	96

Fig. 7.40 - Formas circulares detectadas pelo Filtro Comparativo, na razão de 90%, aplicado no acumulador modificado pelo Filtro Equalizador.....	96
Fig. 7.41 - Gráfico da sobreposição do acumulador obtido pela aplicação da técnica de <i>Template Matching</i> para detectar as prováveis formas circulares na imagem.....	97
Fig. 7.42 - Imagem - com dimensões 378x283 <i>pixels</i> - com bordas obtidas através do OU-exclusivo a partir de uma imagem suavizada pela Gaussiana.....	98
Fig. 7.43 - Gráfico referente aos tempos medidos na execução do Arranjo Acumulador usando a Transformada de Hough na imagem (320x240x256) de bordas grossas. Os eixos do gráfico são dadas pelo início da faixa de raios que deverá ser usada para detectar a forma circular, a quantidade de raios a serem pesquisados menos um e o tempo em segundos que o detector consumiu para realizar a detecção.....	100
Fig. 7.44 - Gráfico referente aos tempos medidos na execução do Arranjo Acumulador usando o método do <i>Template Matching</i> na imagem (320x240x256) de bordas grossas.....	101
Fig. 7.45 - Gráfico referente aos tempos medidos na execução do Arranjo Acumulador usando a Transformada de Hough na imagem (320x240x256) de bordas finas.....	101
Fig. 7.46 - Gráfico referente aos tempos medidos na execução do Arranjo Acumulador usando o método do <i>Template Matching</i> na imagem (320x240x256) de bordas finas.....	102
Fig. 7.47 - Gráfico referente aos tempos medidos na execução do Arranjo Acumulador usando a Transformada de Hough na imagem (378x283x256) de bordas finas.....	102
Fig. 7.48 - Gráfico referente aos tempos medidos na execução do Arranjo Acumulador usando o método do <i>Template Matching</i> na imagem (378x282x256) de bordas finas.....	103
Fig. 7.49 - Gráfico referente aos tempos medidos na execução do Filtro Fuzzy após a construção do Arranjo Acumulador pela Transformada de Hough na imagem (320x240x256) de bordas grossas.....	103

Fig. 7.50 - Gráfico referente aos tempos medidos na execução do Filtro Fuzzy após a construção do Arranjo Acumulador pela técnica do <i>Template Matching</i> na imagem (320x240x256) de bordas grossas.....	104
Fig. 7.51 - Gráfico referente aos tempos medidos na execução do Filtro Fuzzy após a construção do Arranjo Acumulador pela Transformada de Hough na imagem (320x240x256) de bordas finas.....	104
Fig. 7.52 - Gráfico referente aos tempos medidos na execução do Filtro Fuzzy após a construção do Arranjo Acumulador pela técnica do <i>Template Matching</i> na imagem (320x240x256) de bordas finas.	105
Fig. 7.53 - Gráfico referente aos tempos medidos na execução do Filtro Fuzzy após a construção do Arranjo Acumulador pela Transformada de Hough na imagem (378x282x256) de bordas finas.....	105
Fig. 7.54 - Gráfico referente aos tempos medidos na execução do Filtro Fuzzy após a construção do Arranjo Acumulador pela técnica do <i>Template Matching</i> na imagem (378x282x256) de bordas finas.....	106
Fig. 7.55 - Gráfico referente aos tempos medidos na execução do Filtro Equalizador após a construção do Arranjo Acumulador pela Transformada de Hough e <i>Template Matching</i> nas imagens (320x240x256) de bordas grossas e finas.....	106
Fig. 7.56 - Gráfico referente aos tempos medidos na execução do Filtro Equalizador após a construção do Arranjo Acumulador pela Transformada de Hough e <i>Template Matching</i> na imagem (378x282x256) de bordas finas.....	107
Fig. 7.57 - Gráfico referente aos tempos medidos na execução do Filtro Comparativo após a construção do Arranjo Acumulador pela Transformada de Hough e <i>Template Matching</i> nas imagens (320x240x256) de bordas grossas e finas.....	107
Fig. 7.58 - Gráfico referente aos tempos medidos na execução do Filtro Comparativo após a construção do Arranjo Acumulador pela Transformada de Hough e <i>Template Matching</i> na imagem (378x282x256) de bordas finas.....	108
Fig. 1- Gráfico da reta na imagem.....	176
Fig. 2 - Gráfico do ponto correspondente no espaço de parâmetros.....	176
Fig. 3 - Gráfico da interseção entre duas retas no espaço de parâmetros.....	177

Fig. 4 - Gráfico da interseção entre muitas retas no espaço de parâmetros indicando a presença de uma reta na imagem com parâmetros a e b respectivamente iguais a 1,3 e 8.....	177
Fig. 5 - Gráfico da reta. A reta faz um ângulo φ com o eixo das abcissas.....	178
Fig. 6 - Gráfico da reta com o ângulo θ e amplitude ρ representados.....	179
Fig. 7 - Gráfico das cossenoides correspondentes aos pontos (x_i, y_i)	182

LISTA DE TABELAS

Tabela 3.1 - Descrição dos campos da estrutura BITMAPFILEHEADER.....	34
Tabela 3.2 - Descrição dos campos da estrutura BITMAPINFOHEADER.....	34
Tabela 3.3 - Descrição dos campos da estrutura RGBQUAD.....	35
Tabela 7.1 - Dados correspondentes às formas circulares com raios entre 30-35 <i>pixels</i> , encontradas através da Transformada de Hough quando aplicado o Filtro Comparativo na razão de 85%.....	86
Tabela 7.2 - Dados correspondentes às formas circulares com raios entre 65-70 <i>pixels</i> encontradas através da Transformada de Hough quando aplicado o Filtro Comparativo na razão de 98%.....	86
Tabela 7.3 - Tempos medidos na detecção de formas circulares numa imagem (320x240x256) de bordas grossas através da TH (Transformada de Hough) e TM (<i>Template Matching</i>) usando os Filtros Fuzzy, Equalizador e Comparativo. A medição dos tempos dos filtros é independente.....	99
Tabela 7.4 - Tempos medidos na detecção de formas circulares numa imagem (320x240x256) de bordas finas através da TH (Transformada de Hough) e TM (<i>Template Matching</i>) usando os Filtros Fuzzy, Equalizador e Comparativo. A medição dos tempos dos filtros é independente.....	99
Tabela 7.5 - Tempos medidos na detecção de formas circulares numa imagem (378x283x256) de bordas finas através da TH (Transformada de Hough) e TM (<i>Template Matching</i>) usando os Filtros Fuzzy, Equalizador e Comparativo. A medição dos tempos dos filtros é independente.....	100

RESUMO

Neste trabalho é descrito um *software* denominado **Scanc** implementado no ambiente Windows, para a identificação de formas circulares em imagens digitais do tipo BMP. Dois métodos foram implementados para esse fim: a Transformada de Hough e *Template Matching*.

Incluídos neste *software* encontram-se recursos para detecção de bordas e binarização da imagem, entre outros.

O *software* tem aplicação nas mais diversas áreas, que requeiram a identificação de formas circulares.

Palavras-chave: Detecção de formas circulares; Transformada de Hough

ABSTRACT

The present work describes a software named Scanc, developed for circular pattern identification in digital bitmap images (BMP).

Scanc was developed for Microsoft Windows environment and it takes advantage of two methods: Hough Transform and *Template Matching*.

Image tools such as border detection and binarization are also available among other resources.

The software finds several applications wherever circular forms identification is required.

Key words: circular forms identification; Hough Transforms.

1 - Introdução

A Transformada de Hough é um método largamente empregado na identificação de formas que podem, ou não, ser descritas através de parâmetros, em imagens digitais. Sua grande vantagem é a robustez na presença de ruído e a precisão na identificação das formas.

Durante um certo tempo, a implementação da Transformada de Hough ficou restrita aos sistemas de médio e grande porte devido à grande quantidade de memória requerida e ao longo tempo de processamento necessário. Estas restrições dificultavam a disseminação do processamento de imagens para reconhecimento de formas através da Transformada de Hough, pois o custo elevado de um sistema de médio e grande porte inviabiliza a utilização deste procedimento para usuários de menor poder aquisitivo.

Este trabalho investiga um sistema de reconhecimento de padrões, através do processamento de imagens, de baixo custo, para que possa ser aproveitado num maior número de atividades nas mais diversas áreas.

A implementação dessa técnica em microcomputadores pessoais do tipo PC só foi possível com o grande avanço tecnológico recente, que introduziu maior velocidade e mais memória disponível nessa arquitetura, e também com o advento do ambiente *Windows*, devido à possibilidade de manipulação da memória de forma contínua e ao fácil acesso à memória estendida que a programação nesse ambiente proporciona. A vantagem do uso do microcomputador pessoal do tipo PC é a sua popularidade e baixo custo, tornando os programas que são desenvolvidos para este tipo de computador acessíveis a um número maior de usuários.

**Scanc*, o *software* descrito neste trabalho, foi desenvolvido no ambiente *Windows* para identificar formas circulares numa imagem digital. A Transformada de Hough é um dos métodos que foi implementado no *Scanc*, para esse fim. Além desse, foi implementado a técnica de *Template Matching*, técnica empregada, em geral, em processos industriais para reconhecimento de formas.

* *Scanc* é um nome fantasia usado para identificar o programa desenvolvido neste trabalho

O objetivo primeiro deste trabalho foi desenvolver uma ferramenta para estudo e análise das imagens utilizadas nos grupos de Visão Computacional e Processamento de Imagens Médicas do Departamento de Engenharia Elétrica, onde a Transformada de Hough tem sido muito utilizada.

O segundo objetivo foi avaliar a viabilidade da implementação desse método num computador pessoal do tipo PC, e a possibilidade de utilizá-lo futuramente em aplicações em tempo real.

O presente trabalho está dividido em 6(seis) capítulos:

- No Capítulo 2 são descritas as técnicas descritas na literatura para implementação da Transformada de Hough
- O Capítulo 3 justifica a necessidade de se utilizar o ambiente *Windows* para realizar a detecção de formas.
- No Capítulo 4 são descritas as técnicas escolhidas para realizar o pré-processamento da imagem, bem como a forma como foram implementadas.
- No Capítulo 5 é descrita a forma como foram implementados a Transformada de Hough e a técnica do *Template Matching*
- No Capítulo 6 são descritos os filtros utilizados para permitir a separação das formas circulares mais prováveis na imagem a partir da informação fornecida pela Transformada de Hough ou pela técnica do *Template Matching*.
- No Capítulo 7 são mostrados e analisados os resultados do trabalho.

2 - Métodos para Identificação de Formas

2.1- Introdução

A necessidade de se obter processos automáticos para se inspecionar objetos como foi feito em ¹⁷HORAUD & SOSSA (1995) e em ²⁶LEUNG & HUANG (1991), condições do tempo como em ³¹PEKINPAUGH & HOLYER (1994), ou mesmo partes do corpo humano como fizeram ³⁰NIXON & HAMES (1993), ¹¹GATH & HOORY (1995), e ³³PHILIP et alii (1994), contribuiu para o desenvolvimento de técnicas computacionalmente viáveis para este fim.

Inspecionar visualmente uma linha de produção é um procedimento largamente utilizado na manufatura de objetos como foi comentado em ³⁶SUN & TSAI (1992). Inspeções realizadas através da visão humana, estão sujeitas a falhas, pois, como argumentaram ¹⁰FOSTER et alii (1990), num processo industrial as tolerâncias tendem a ser bem pequenas e a razão de produção a ser alta .

Para resolver esse problema busca-se o desenvolvimento de técnicas computacionais que automatizem o processo de inspeção visual. Com vistas à simplificação, eficiência e minimização de custos, estes processos tendem a ser altamente dedicados à inspeção desejada [⁴⁴BATCHELOR et alii (1985)].

Quanto maior a complexidade da forma que se precisa inspecionar através de processos automáticos, mais dimensões serão necessárias para representá-la, aumentando a quantidade de processamento, como descreve ⁰²BALLARD (1981), necessitando assim de mais tempo e memória, como prova ¹⁹HSU & HUANG (1990).

Se o objetivo for detectar retas, a fim de inspecionar objetos retangulares, ou triangulares, deve-se identificar pontos, pois dois deles definem uma reta. Um ponto pode ser definido por apenas dois parâmetros da imagem (x,y), porém circunferências os são pelo seu ponto central (x,y) e seu valor de raio, ou seja, são necessários três parâmetros de imagem para definir uma circunferência. Para uma elipse precisaríamos de cinco parâmetros de imagem: seu ponto central (x,y), seu raio menor, seu raio maior e sua inclinação em relação aos eixos da imagem como está descrito em ⁴¹YIP et alii (1992).

Formas não geométricas também podem ser inspecionadas. ²⁸MERLIN & FARBER (1975) propuseram um método para detectar toda e qualquer curva, apresentando a vantagem de poder ser implementado aproveitando características de paralelismo. Esse método não necessita de que os pontos de borda sejam conectados, e a direção do gradiente não é usada. Dessa maneira, o método possui grande robustez a ruídos. Porém, segundo ³⁸WATANABE et alii (1991), esse método pode detectar apenas padrões de translações paralelas, e métodos auxiliares são necessários para detectar padrões rotacionados e mudanças de escala. Uma forma de se fazer isto é repetir o método ²⁸MERLIN & FARBER (1975) para diferentes rotações da forma.

Segundo ⁰⁷COSTABILE & PIERONI (1986) formas não geométricas também podem ser tratadas como uma combinação de formas geométricas e podem ser processadas separadamente, e ⁰²BALLARD (1981) estendeu o método de ²⁸MERLIN & FARBER (1975) de maneira a usar a informação do gradiente, generalizando assim o método de Hough a fim de detectar formas arbitrárias.

2.2 - A Transformada de Hough

Foi através de uma patente que ¹⁸HOUGH (1962) propôs um método para detecção de padrões complexos, mais precisamente para estudar o comportamento de partículas sub-atômicas através do rastro deixado quando estas passavam através de um campo visual.

A Transformada de Hough (HT) , é um método de parametrização onde cada ponto da imagem deve ser representado por uma reta ou por uma senóide, num espaço de parâmetros.

Uma explanação mais detalhada sobre a Transformada de Hough para retas está contida no Apêndice 9.

A Transformada de Hough também pode ser usada para detectar formas circulares. Da mesma forma que na detecção de retas, é possível utilizar um espaço parametrizado para indexar os 3(três) parâmetros que definem um círculo: as

coordenadas **a** e **b** do centro do círculo e o **raio**. Com esses três parâmetros ficam definidas as posições dos pontos que pertencem ao círculo.

Fig. 2.1 - Forma circular contida numa imagem.

Para se detectar, por exemplo, a presença de uma forma circular de raio r numa imagem como a da figura 2.1, pode-se usar os N pontos de borda da forma como centros (x_i, y_i) de círculos de raio r , sendo $i = 1, 2, 3, \dots, N$ (onde N é o número de pontos de borda que pertencem à forma circular na imagem). Estes círculos são traçados no espaço de parâmetros e se a forma na imagem for circular, eles interceptarão o centro (a, b) da forma circular da imagem como ilustrado na figura 2.2.

Geralmente o valor exato do raio não é conhecido, e se estabelece então um intervalo de busca para esse parâmetro. Quanto maior esse intervalo, mais valores de raio serão pesquisados. Porém este procedimento provoca um grande aumento do número de valores que precisariam ser calculados e acumulados e também aumenta o tempo de busca. O que se faz então para obter maior eficiência, é escolher o menor intervalo possível para que o processo de detecção se concretize num tempo mínimo, ocupando o menor espaço possível em memória. Por isso, é interessante conhecer com antecedência a faixa de valores possíveis para o raio.

Fig. 2.2- Os círculos paramétricos interceptam-se no centro da forma circular da imagem.

Quando se faz a variação do valor do raio, o espaço de parâmetros deixa de ser planar e passa a ser espacial. Cada valor de raio será pesquisado num dos planos paramétricos. Desta forma, ao incrementarmos o valor do raio estaremos formando, para cada ponto de borda da imagem, um cone no espaço de parâmetros como ilustrado na figura 2.3. O ponto de cruzamento de um maior número destes cones nos mostrará o centro de uma provável forma circular na imagem.

2.3- Método do *Template Matching*

Template Matching é um método de segmentação baseado apenas na comparação entre níveis de cinza para realizar o reconhecimento de formas, e utiliza uma sub-imagem, conhecida como *template*, contendo a forma a ser pesquisada.

O processo de reconhecimento consiste em deslocar o *template* sobre toda a imagem buscando similaridades entre o *template* e a imagem, sendo o objeto identificado quando um predeterminado grau de casamento é alcançado [54SCHALKOFF (1989)].

Fig. 2.3 - Cruzamento entre cones no espaço de parâmetros que determinam a existência de uma forma circular na imagem.

A forma procurada pode estar contida em qualquer posição na imagem. Para encontrá-la então, é preciso que o *template* seja sobreposto em todas as posições possíveis na imagem. A figura 2.4a ilustra o exemplo de um *template* contendo uma forma circular que ao ser convoluído com a imagem, proporciona a detecção dessa forma presente na imagem ilustrada na figura 2.4b.

2.4 - Métodos para detectar formas circulares

A fim de desenvolver o método de Hough para detectar padrões comuns ⁰⁹DUDA & HART (1972) estabeleceram um erro aceitável para θ e ρ (veja apêndice 9) num plano θ - ρ colocado sobre em uma grade acumuladora quadrangular. Esta quantização pode ser confinada à região $0 \leq \theta < \pi$, $-R \leq \rho \leq R$. Desta forma, pontos fora desta região correspondem a linhas no plano da imagem que não atravessam o campo visual.

A região de quantização é tratada como um campo bidimensional de acumuladores. Para cada ponto (x_i, y_i) no plano da imagem, a curva correspondente no

plano paramétrico irá incrementar a contagem de cada célula da grade contida ao longo da curva. Outrossim, uma dada célula do acumulador bidimensional grava o número total de curvas passando através dela.

Fig. 2.4 - a) O template contendo a forma procurada que deve ser convoluído com a imagem; b) imagem contendo entre outras, a forma circular procurada.

Depois que todos os pontos tiverem sido considerados, o campo é inspecionado a fim de se encontrar células com altas contagens, obtendo picos no acumulador, qualificando assim retas ou círculos no plano da imagem.

Segundo ²³KIMME et alii. (1975), o algoritmo de ⁰⁹DUDA & HART aborda o método para encontrar formas circulares numa imagem. Nesse método, o raio é limitado pelo uso de apenas valores discretos, restringindo assim a faixa e os passos de quantização da forma circular contida na imagem digitalizada. Um exemplo da atuação da grade acumuladora está ilustrado na figura 2.5.

Podemos visualizar o campo acumulador como planos formados por grades acumuladoras, sendo um plano para cada valor de raio. Círculos de raio r são traçados numa grade sendo os seus centros os pontos de borda da imagem. A cada intersecção destes círculos incrementa-se de 1(um) o valor acumulado da célula correspondente na grade acumuladora. As células incrementadas um maior número de

vezes corresponderão a picos no acumulador nas posições mais prováveis para os centros das formas circulares na imagem.

Fig. 2.5 - Grade acumuladora quadrangular onde um grande número de círculos passa pela célula correspondente ao centro da forma circular da imagem.

Na detecção de círculos expressa por ⁰⁹DUDA & HART (1972), se um círculo é parametrizado pelas coordenadas do centro e o raio (a,b,r), então estas coordenadas indicam a posição dos pontos de borda (x,y), através da equação 2.1.

$$(x-a)^2 + (y-b)^2 = r^2 \quad (2.1)$$

Esta equação também indica que um dado ponto de borda (x_i,y_i) deve estar incluído em algum círculo cujos parâmetros estão contidos na superfície de um certo cone circular no espaço de parâmetros (a,b,r). Se no cone houver vários pontos de borda interceptando-se no mesmo ponto, então todos os pontos de borda da imagem que contribuem para este ponto estarão no círculo definido por esses três parâmetros. A figura 2.6 ilustra a atuação dos vários planos formados por grades acumuladoras.

Fig. 2.6 - Planos constituídos de grades acumuladoras quadrangulares onde um grande número de círculos passa na célula correspondente ao centro da forma circular da imagem.

⁰⁸DAVIES (1992) fez um estudo dos picos obtidos no acumulador da Transformada de Hough. Para fazer o estudo, ele usou um acumulador para detecção de círculos e concluiu que a complexidade do acumulador depende da qualidade da imagem que está sendo verificada. Além disso, a acuidade com a qual o objeto pode ser localizado depende fundamentalmente da forma do pico.

²³KIMME et alii (1975) descrevem um método para detectar formas circulares através da direção do gradiente de cada ponto de borda da imagem. Quando as direções do gradiente destes pontos apontam para uma mesma célula acumuladora, esta célula é então incrementada. A posição do centro é escolhida passando-se um *threshold* proporcional ao raio, de forma que a razão entre o valor do *threshold* e do raio seja sempre constante. O uso da direção do gradiente, segundo eles, é justificado pela sensível diminuição do tempo de processamento.

Esta idéia também é usada por ⁴³YUEN et alii (1990) para detectar o centro de uma possível forma circular na imagem. O centro do círculo paramétrico deve estar contido na linha que passa através do ponto de borda e ao longo da direção do gradiente. Como resultado, em vez de incrementar todo o cone circular, apenas

segmentos do cone precisam ser incrementados. Este procedimento foi denominado de Transformada de Hough Padrão (SHT).

O acumulador deve ser incrementado em 3D (a,b,r) , que pode ser eficientemente indexado usando as direções do gradiente que irão definir o centro (a,b) e a equação do círculo a fim de se calcular o parâmetro r . Uma escolha natural é ter cada célula da grade acumuladora representada por um ponto no espaço de parâmetros. Desta forma, para procurar por uma forma circular numa imagem de dimensão N^2 , cujo raio é menor ou igual a N , é necessário um acumulador 3D de tamanho N^3 .

Uma complementação a se fazer a respeito do uso da direção do gradiente é que, se por um lado permite uma redução de memória requerida, por outro, se a imagem for muito ruidosa ou complexa, este método tende a ser ineficaz, detectando falsas formas circulares.

²⁰ILLINGWORTH & KITTLER (1988) fizeram um levantamento sobre a Transformada de Hough e concluíram que esse método possui grande capacidade de detecção de formas, mesmo sob condições de ruído, porém a necessidade de mais memória e tempo de processamento cresce com o aumento da dimensionalidade do acumulador.

Na tentativa de resolver o problema da dimensionalidade do acumulador ⁰⁵CASASENT & KRISHNAPURAM (1987) assumiram que um objeto curvo pode ser considerado como uma composição de pequenos segmentos de retas tangenciais à borda desse objeto. Cada um desses segmentos corresponde a um ponto (ρ,θ) no espaço paramétrico. Deste modo, irá existir um grande número de pequenos segmentos de retas e estes irão gerar pequenos picos no espaço paramétrico.

Picos menores ainda, geralmente de altura igual a dois, irão ser gerados. Esses picos são provenientes de retas que possuem apenas dois pontos na imagem. Essas retas geralmente não são tangenciais à curva e portanto, não devem contribuir para a detecção da curva na imagem e deverão ser retirados através de um *threshold*.

Depois de retirados os picos muito pequenos, os pontos (ρ,θ) incrementados maior número de vezes, irão cada um caracterizar a coordenada polar de um ponto na

imagem. Assim, através de uma Transformada inversa de Hough, será possível detectar a curva procurada.

Este método pode ser usado para detectar qualquer objetos curvos na imagem, tais como: objetos circulares, elípticos e parabólicos, mesmo que estejam parcialmente ocultos ou seccionados.

Uma grande vantagem deste método é a redução da quantidade de memória requerida, já que trata objetos curvos (que na HT convencional requerem um acumulador multi-dimensional) como segmentos de retas, que requerem acumuladores bidimensionais. Porém este método possui um grande inconveniente na escolha do *threshold*. Para uma seleção razoável é necessário conhecer a resolução da entrada e a do espaço paramétrico. Em imagens muito complexas, este método deverá falhar, pois os picos referentes à imagem real, por serem pequenos, tendem a ter a mesma amplitude dos picos referentes às retas que não são tangenciais à curva, sendo assim impossível separá-los utilizando um *threshold*.

¹⁵HANAHARA & HIYANE (1990), baseados na analogia da Transformada de Hough com o Princípio de Huygens's, apresentaram um algoritmo de detecção de formas circulares que resolve numericamente uma equação de onda bidimensional usando apenas operações com bases adjacentes, que são, Laplaciano, adição de imagem e multiplicação de constantes com o conteúdo da imagem. Pelo fato de não usar extração de borda, o algoritmo detecta formas circulares em imagens de baixo contraste.

Este método usa o Princípio de Huygens's, - uma regra matemática para descrever o comportamento da luz - este princípio estabelece que a superposição de todas as ondas esféricas elementares com mesmo raio, emitidos a partir de posições puntiformes numa frente de onda, formam um estado de onda futuro.

A Transformada de Hough é muito sensível à complexidade da imagem e deve falhar nas imagens de baixo contraste. O problema não é propriamente da transformada de Hough, porém da extração da borda (usada para simplificar a imagem e diminuir a quantidade de processamento), e mais especificamente da escolha do *threshold*.

O conceito básico deste método vem das ondas na água, com a altura das ondas correspondendo à escala de cinza. Distúrbios simultâneos na circunferência de um círculo irão se propagar na direção do centro, onde formarão um pico. Desta maneira, simulando-se a propagação de onda, identificam-se picos que podem ser usados para detectar formas circulares.

Algoritmos que se valem da Transformada de Hough e do Princípio de Huygens's desenham ambos círculos ou esferas com o mesmo raio e usam o princípio da superposição. Na Transformada de Hough, o princípio da superposição aparece durante o armazenamento, e no Princípio de Huygens's esta propriedade é inerente à equação da onda.

A desvantagem desse método é o fato de possuir um custo computacional maior do que o método baseado na Transformada de Hough. Enquanto neste último a quantidade de processamento é proporcional a $R^2 N_{\text{borda}}$, no método proposto por ¹⁵HANAHARA & HIYANE (1990) esta proporcionalidade fica em RN^2 sendo R o número de raios pesquisados, N uma das dimensões de uma imagem $N \times N$ e N_{borda} o número de pontos de borda da imagem.

Um dos problemas do SHT de ⁴³YUEN et alii (1990) é a grande quantidade de dados a serem armazenados se o intervalo de valores para o raio for muito grande. ¹²GERIG e KLEIN (1986) então reordenaram o cálculo da HT para refazer o acumulador 3D, de dimensão N^3 , com três acumuladores 2D, ordenados em tamanho N^2 . Este método é chamado de Transformada de Hough de Gerig (GHT). Ao contrário do SHT, onde a determinação do pico é feita usando todos os parâmetros, a Transformada de Hough de Gerig realiza a HT em estágios.

Uma deficiência do método de ¹²GERIG e KLEIN (1986) é que devido ao fato dos picos serem gravados em posições espaciais bidimensionais, os valores serão reescritos se dois círculos paramétricos interceptarem uma posição comum. Por essa razão, numa implementação simples não é possível acumular todos os detalhes de círculos concêntricos.

A eficiência desse método pode aumentar se neste for incorporado a direção do gradiente, ao invés de se acumular todo o círculo paramétrico. Esta modificação da Transformada de Hough de Gerig é denominado de Transformada de Hough de

Gerig com Gradiente (GHTG). Se a direção do gradiente for usada, então apenas a linha do cone que recai sobre a direção do gradiente de um ponto de borda será incrementada.

Se a direção do gradiente dos pontos de borda é disponível, uma outra forma de reduzir o armazenamento e demanda computacional na procura pela forma circular é decompor o problema em dois estágios como mencionado em ²⁰ILLINGWORTH & KITTLER (1988). O método consiste num acumulador 2D para encontrar o centro da forma circular, seguido de um HT de 1D para determinar o raio. Este método é por eles chamado de Transformada de Hough 2-1 (21HT). O centro da forma circular deve estar ao longo da posição do gradiente de cada ponto de borda. Então o ponto de interseção comum desses gradientes identificam seu centro.

O primeiro estágio é usado para acumular os valores de HT indexando com os valores do centro (a,b) num espaço 2D e o segundo estágio usa os parâmetros do centro do primeiro estágio e pela equação do círculo, constrói e analisa um histograma de possíveis valores do raio.

Um histograma do raio é feito para cada candidato a centro do círculo, e picos no histograma do raio indicam evidência de círculos. O processo é ilustrado nas figuras 2.7 e 2.8.

Um grande problema do método 21HT é o fato do acumulador 2D usado para encontrar o centro ser mais difícil de se analisar que o acumulador 3D, o qual vem a ser uma projeção completa daquele espaço. Em adição, um falso pico ou inacuidade na estimativa do parâmetro no primeiro estágio irá afetar a performance do segundo estágio.

²⁷LI et alii (1986) sugeriram um método denominado de Transformada Hough Rápida (FHT) que usa uma estrutura multi-dimensional para simultaneamente acumular e detectar picos, necessitando de menos memória que o SHT para seu armazenamento.

O parâmetro original de volume é considerado como um hipercubo simples, se o número de pontos da imagem que incrementa os parâmetros de um certo hipercubo for maior do que um valor limite. Então assume-se que no espaço de parâmetros

correspondente existe um pico, e o volume do hipercubo é investigado com maior detalhe pela decomposição do hipercubo em sub-hipercubos.

Fig. 2.7 - O centro é localizado através de um espaço acumulador bidimensional.

Fig. 2.8 - O valor do raio é encontrado através de um histograma obtido a partir da medida entre os pontos de borda da imagem e o valor do centro.

Este é um processo iterativo de acumulação, análise e divisão contínua até um limite de resolução dos parâmetros, ou o hipercubo falha e seus incrementos ficam abaixo do limite que define um pico.

Uma modificação apropriada para o método de ²⁷LI et alii (1986) denominada de Transformada de Hough Rápida Modificada (MFHT) utiliza a direção do gradiente de cada ponto da imagem.

²¹ILLINGWORTH & KITTLER (1987) introduziram a Transformada de Hough Adaptativa (AHT). A implementação do AHT usa um pequeno acumulador ordenado (9x9x9). Realiza uma análise grosseira de cada parâmetro assim como no FHT de ²⁷LI et alii (1986), e interativamente focaliza os parâmetros das regiões de maiores densidades do acumulador. A escolha da resolução de cada parâmetro para a próxima interação é independente entre si, e é feita de maneira dinâmica e depende da distribuição do número de incrementos no acumulador.

Isto supera o principal problema associado com a parametrização polinomial da elipse, que é o de grande armazenamento de dados devido às grandes larguras de faixa dos parâmetros.

Por outro lado, problemas com o AHT incluem sua complexa implementação. Além disso assim como no método de ²⁷LI et alii (1986) células incrementadas pelas regiões de maiores densidades de pontos de borda tendem a ser mais incrementadas gerando assim falsos picos no acumulador. Deste modo o método tende a falhar quando a imagem a ser analisada for complexa.

Também com o objetivo de diminuir a dimensão do acumulador e aumentar a velocidade de processamento, ⁰⁶CHAN & SIU (1991) apresentaram um método que usa apenas um par de acumuladores bidimensionais ordenados. Este método é capaz de discriminar múltiplas (incluindo concêntricas) formas circulares em imagens complexas.

As duas dimensões do primeiro acumulador representam as coordenadas **a** e **b** dos centros das formas circulares e as do segundo acumulador representam a coordenada **a** do centro e o raio. Cada ponto de borda irá ter uma reta correspondente neste último espaço acumulador obtida a partir da equação 2.1. A

intersecção entre estas retas determinará o valor do raio para um determinado valor do parâmetro a .

A figura 2.9 mostra que a capacidade de encontrar formas circulares concêntricas através deste método é mantida, enquanto que os acumuladores utilizados são bidimensionais reduzindo a necessidade de memória.

Fig. 2.9 - a) Imagem contendo formas circulares concêntricas; b) espaço acumulador dos parâmetros a e b ; c) espaço acumulador dos parâmetros a e r .

Por se tratar de um método de particionamento do acumulador este método apresenta os mesmos problemas relatados para o método 21HT.

⁴³YUEN et alii (1990) compararam vários métodos de detecção de formas circulares. Os métodos considerados são: a Transformada de Hough Padrão (SHT), a Transformada de Hough Rápida Modificada (MFHT) de ²⁷LI et alii (1986), o método que executa Hough em dois estágios (21HT) mencionado em ²¹ILLINGWORTH & KITTLER (1987), o método que utiliza apenas acumuladores bidimensionais (GHT), e por fim, aquele que além disso utiliza a direção do gradiente (GHTG) descritos por ¹²GERIG & KLEIN (1986).

A fim de comparar os vários métodos ⁴³YUEN et alii (1990) usaram uma imagem contendo muitas formas circulares de diversos raios. Porém a imagem usada praticamente não possuía ruído.

O gráfico da razão entre o número de formas circulares detectadas e o número de formas circulares presentes na imagem em função dos métodos utilizados está ilustrado na figura 2.10.

Fig. 2.10 - Gráfico de desempenho de cada método em relação ao número de formas circulares detectadas.

O resultado em termos de tempo de processamento e quantidade de armazenamento necessário do estudo comparativo está no gráfico ilustrado na figura 2.11, obtido a partir do processamento de uma imagem num computador μ Vax-2. Pode-se perceber por este gráfico que o SHT requer muito espaço para armazenamento de dados enquanto o GHT necessita de um longo tempo de processamento. Tanto o 21HT quanto o GHTG conseguem ter um bom compromisso entre o armazenamento e o tempo computacional. A performance do MFHT depende da complexidade espacial da imagem que está sendo analisada. A comparação entre o GHT e o GHTG ilustra a vantagem computacional do uso da direção do gradiente quando a imagem é isenta de ruído.

Nessas condições o GHTG aparece como o melhor método, pois requer pequeno armazenamento, é rápido, preciso e tem boa relação entre as formas detectadas e as totais, na imagem. Seu problema é sua incapacidade de detectar círculos concêntricos.

O SHT requer enorme armazenamento. O 21HT é razoavelmente bom e requer pequeno armazenamento, mas tem problemas na detecção de formas circulares pequenas. A Transformada de Hough de Gerig é considerada ineficiente pela falta do uso da direção do gradiente e o MFHT é comparativamente grande e depende dos detalhes da estrutura da imagem.

Fig. 2.11 - Gráfico comparativo do tempo gasto para armazenamento dos diversos métodos.

2.5 - Métodos para detectar formas circulares como caso particular das elípticas

Formas circulares também podem ser vistas como caso particular de formas elípticas. Desta forma pode-se usar métodos de detecção de formas elípticas, e estabelecer a condição onde os dois focos da elipse coincidem.

³⁷TSUJI & MATSUMOTO (1978) apresentaram uma Transformada de Hough para elipse com cinco parâmetros. A técnica através da qual os focos são encontrados é baseada numa propriedade geométrica da elipse. Se um par de pontos do contorno possui tangentes iguais, ou quase iguais, então o ponto equidistante destes dois pontos será o centro da elipse. Um acumulador bidimensional de pontos médios de cada par de pontos extraídos é então construído. Picos neste histograma definem os centros dos candidatos a elipses na imagem. A posição dos pontos que

contribuem para a extração do centro é então identificada e o método dos mínimos quadrados é aplicado para obtenção dos parâmetros restantes.

Para identificar a existência de elipses concêntricas, um teste geométrico é aplicado. A técnica dos mínimos quadrados é aplicada separadamente para cada elipse concêntrica, se ela existir. Porém esta técnica de encontrar centros de elipses tem sérias limitações. Se devido a oclusões de parte das elipses, insuficientes pares de pontos com tangentes iguais restarem no contorno dessas elipses, esta técnica irá recorrer em erro. Retas na imagem devem ser localizadas e removidas antes das elipses serem procuradas. Elas podem gerar falsos picos durante o processo de busca do centro.

⁴²YUEN et alii (1989) implementaram uma técnica que pode tolerar maiores níveis de oclusão de uma elipse, que pode ser estendida para um círculo, caso particular da elipse.

O procedimento para encontrar o centro da elipse se faz a partir da escolha de dois pontos P e Q, como ilustrado na figura 2.12, com tangentes não paralelas que se interceptam num ponto T, sendo M o ponto médio entre P e Q. Então o centro da elipse está contido na reta paramétrica definida por T e M. O centro de uma elipse, ou mesmo de um círculo é encontrado quando, na implementação de um acumulador, for detectado um pico no espaço de Hough. A Transformada de Hough Adaptativa (AHT) de ²¹ILLINGWORTH & KITTLER (1987) é usada para extrair os três parâmetros remanescentes. Desta forma este método traz consigo os problemas do AHT.

⁴¹YIP et alii (1992) apresentaram uma técnica para introdução de eficiência e redução no tamanho da memória do acumulador. Baseado nessa técnica, apenas um campo bidimensional é necessário para detecção de formas circulares e elípticas. As posições dos quatro pontos da elipse que determinam seus eixos, são calculados e acumuladas num campo bidimensional, através de 5(cinco) propriedades geométricas das elipses

Posições de máximo no acumulador bidimensional irão indicar a posição desses pontos para o círculo ou elipse correspondente. O método mais comum é determinar um *threshold* global. As células acumuladas com contagens superiores ao

threshold indicam a provável existência de um desses pontos. Desta forma, vários conjuntos de quatro picos relativos no acumulador que satisfaçam essas condições irão indicar a posição de uma forma circular ou elíptica.

Fig. 2.12 - Geometria básica para encontrar o centro da elipse através de tangentes não paralelas.

Pode-se aumentar a robustez do algoritmo usando-se apenas três pontos da elipse que determinam seus eixos ao invés dos quatro, porém isto aumenta bastante o tempo de processamento, busca e comparação de resultados de diferentes combinações das posições dos vértices.

Outra maneira de encontrar formas elípticas e circulares a partir de suas propriedades geométricas é descrita por ¹⁶HO & CHEN (1995). O método por eles proposto localiza tais formas em duas fases: na primeira, apenas o centro da forma é procurado e seus pontos característicos são classificados em diferentes sub-imagens; enquanto que na segunda fase, cada sub-imagem é analisada em relação à simetria geométrica dos objetos detectados.

Como neste método a direção do gradiente não é usada, a acuidade não é perdida. Desta forma, este método é mais rápido (pelo fato de usar apenas Transformada de Hough para retas) e mais preciso que os demais algoritmos de detecção de formas circulares e elípticas baseadas na Transformada de Hough. Porém, através de implementações práticas eles observaram que se a forma elíptica na imagem tiver 25% ou mais de sua borda apagada, o algoritmo perderá a capacidade de montar os paralelogramos perdendo sua capacidade de detecção.

Os modos de detecção de formas elípticas tradicionais assumem que os contornos a serem detectados são conectados. Portanto eles têm a desvantagem de não poderem detectar elipses cujo contorno é consideravelmente particionado em razão da presença de ruído e sobreposição.

Para lidar com este problema, ³⁸WATANABE et alii (1991) apresentaram um método baseado em ²⁸MERLIN & FARBER (1975), que pode lidar com imagens onde existe um certo número de formas elípticas onde a conectividade do contorno é ausente devido ao ruído e parte do contorno é perdida devido a sobreposições. Este método não depende muito das propriedades geométricas da elipse e a detecção é feita utilizando-se imagens multi-resolucionais e o método dos mínimos quadrados. Porém, resultados experimentais mostraram que a precisão da detecção começa a baixar se 20% dos *pixels* do contorno forem perdidos devido ao ruído e é completamente comprometida se 50% dos *pixels* do contorno forem perdidos devido à sobreposição.

²²KÄLVIÄINEN et alii (1995) fizeram uma comparação entre métodos probabilísticos e não probabilísticos. A idéia básica dos métodos probabilísticos é escolher de forma aleatória pontos a serem analisados na imagem. Isto diminui muito a quantidade de pontos pesquisados e deste modo, o custo computacional é bastante atenuado, diminuindo assim a quantidade de memória necessária e o tempo para realizar a detecção. É previsto que este método deva apresentar vários problemas quando for usado em imagens complexas ou ruidosas.

Segundo ²⁵LEAVERS (1992), a Transformada de Hough é uma poderosa ferramenta na análise de formas. A maior falha desta técnica é a excessiva necessidade de acúmulo de dados e complexidade computacional. Usando a Transformada de Hough Padrão (SHT), cada ponto da imagem vota independentemente para todas as posições da forma sob detecção. Destarte, há uma grande redundância de evidências concernentes à imagem gerada.

Os Métodos Probabilísticos são uma tentativa de reduzir a geração de informação redundante amostrando os dados da imagem de várias maneiras. A Transformada de Hough Generalizada Dinâmica (DGHT) proposta por ²⁵LEAVERS (1992) pertence a essa família. Se T é a resolução do espaço da transformada e n é o

número de parâmetros sob detecção, então o uso do DGHT reduz a memória requerida de T^n a nT e introduz uma oportunidade para fazer de forma paralela os cálculos e armazenamento de parâmetros.

Essencial para o uso eficiente de técnicas probabilísticas, é um critério para assegurar uma amostragem adequada no que diz respeito à detecção resultante desejada. Um critério robusto é deduzido para o DGHT que é aplicado para detecção de círculos e elipses concorrentes usando uma imagem real sob condições de uma larga faixa e variedades de ruído. Isto mostra que o DGHT é muito bom quando a imagem de entrada possui oclusões e ruídos. O método consegue melhorar a relação sinal-ruído através da segmentação da imagem, porém esta segmentação tem que ser suficientemente grande para abrigar toda a forma que se quer detectar sob pena de aumentar a incerteza na detecção.

A segmentação é possível se forem considerado os pequenos segmentos de retas detectados a partir da conectividade entre seus pontos. Estes segmentos aparecem na digitalização da imagem, tanto nas formas circulares quanto elípticas, e basicamente demarcam o topo, a base, as extremidades da direita e esquerda tanto dos círculos quanto das elipses.

A robustez da detecção de formas elípticas e circulares depende também do seu grau de oclusão, bem como do número de formas que se quer detectar.

Todos estes métodos tem como principal objetivo diminuir a quantidade de dados acumulados porque são feitos originalmente para detectar formas elípticas. Para conseguir isso estes métodos tendem a comprometer a robustez da detecção, principalmente quando a imagem é ruidosa. Portanto, quando o objetivo é detectar formas circulares, deve-se utilizar métodos específicos para esse fim, eliminando a priori, parte do problema da multi-dimensionalidade do acumulador.

2.6 - Métodos para detecção de arcos de circunferências e simetrias rotacionais

Ao contrário de técnicas como Fourier e Momento [⁴⁶GONZALES (1987)] que são capazes de captar apenas contornos fechados, a Transformada de Hough tem

a capacidade de reconhecer formas a partir de pontos desconectados, isto proporciona a possibilidade de perceber simetrias rotacionais numa imagem.

³²PEI & HORNG (1995) desenvolveram um método capaz de detectar arcos de circunferências usando a Transformada de Hough para identificação do ponto focal do arco.

Pontos de borda fora da forma circular que não satisfaçam o foco encontrado são eliminados e uma segunda Transformada de Hough para analisar apenas os pontos do arco é realizada, encontrando assim seu comprimento. O ponto médio do arco é determinado através de identidades geométricas.

⁴¹YIP et alii (1994) introduziram uma técnica para detectar simetrias rotacionais calculando o centro de rotação e a ordem de simetria, que é o número de vezes que os pontos rotacionalmente simétricos se repetem. Isto é feito através das propriedades geométricas entre os pontos rotacionalmente simétricos e também utilizando a Transformada de Hough para determinar o centro mais provável da simetria rotacional.

2.7 - Métodos que utilizam a lógica Fuzzy

¹⁴HAN et alii (1994) desenvolveram um método que exclui os círculos extras que surgem quando é usada a Transformada de Hough convencional na detecção de formas circulares.

A detecção de bordas numa imagem digitalizada gera pontos extras àqueles necessários para definir apenas uma reta ou um círculo, isto é, as bordas geralmente não são finas o suficiente para evitar que o HT convencional detecte várias retas ou formas circulares onde na verdade só existe uma reta ou uma forma circular. O uso da lógica Fuzzy foi proposto para resolver este problema.

Um valor Fuzzy é representado por um par $\langle x, \mu \rangle$, onde o x é o elemento do conjunto Fuzzy, e μ seu grau de pertinência que varia do valor zero a um.

Um valor mínimo α de grau de pertinência, pode ser escolhido de forma que $\mu(x) \geq \alpha$ onde x pertence ao conjunto Fuzzy. Se $\alpha = 1$, então o grau de pertinência é

máximo, ou seja, o único valor possível será x . Porém se $\alpha = 0$, então o grau de pertinência é mínimo e pode haver outros valores dentro de um intervalo limitante.

Se α for representado por um círculo, de forma que seus pontos sejam

$$\begin{aligned}x &= x_0 + r \cos \psi \quad \text{e} \\y &= y_0 + r \sin \psi\end{aligned}\tag{2.2}$$

então a equação da reta ficará

$$(x_0 + r \cos \psi) \cos \theta + (y_0 + r \sin \psi) \sin \theta = \rho \tag{2.3}$$

e sua derivada

$$-r \sin \psi \cos \theta + r \cos \psi \sin \theta = 0 \tag{2.4}$$

Desenvolvendo a igualdade, percebe-se que $\tan \theta = \tan \psi$, então $\theta = \psi$ ou $\psi + \pi$, então pode-se reescrever a equação 2.3 como sendo:

$$\rho = (x_0 \pm r \cos \theta) \cos \theta + (y_0 \pm r \sin \theta) \sin \theta \tag{2.5}$$

$$\rho = x_0 \cos \theta + y_0 \sin \theta \pm r \tag{2.6}$$

Desta forma, o intervalo onde um dado ponto do espaço da imagem pode pertencer a uma reta, que no espaço de parâmetros equivale a uma senóide, é de $-r$ até $+r$ na direção de ρ no espaço de parâmetros.

Para os círculos o procedimento é equivalente. A equação do círculo é

$$\rho^2 = (a - x)^2 + (b - y)^2 \tag{2.7}$$

substituindo x e y pelos seus valores, considerando o círculo representativo de α

$$\rho = \pm \sqrt{(a - x_0 - r \cos \psi)^2 + (b - y_0 - r \sin \psi)^2} \quad (2.8)$$

portanto,

$$\rho = \pm \left(r \pm \sqrt{(a - x_0)^2 + (b - y_0)^2} \right) \quad (2.9)$$

Mais uma vez o intervalo possível é de -r a +r na direção de ρ .

Pode-se implementar a Transformada de Hough com Fuzzy (FHT), convoluindo a integral de uma forma de onda de densidade linear $g(d)$ com a Transformada de Hough da imagem na direção ρ . A forma de onda de densidade linear em questão pode ser descrita como dois segmentos de retas formando um triângulo com a largura da base valendo $2r$. O resultado da integral desta forma de onda linear será uma função exponencial, e seu valor será dado por

$$g(d) = k \exp\left(-d^2/\sigma^2\right) ; \text{ se } d < r \quad \text{ou} \quad (2.10)$$

$$g(d) = 0 ; \quad \text{se } d \geq r$$

As constantes k , σ e r são empíricas. A constante k só deve assumir valores positivos inteiros e $\sigma = r$ (também inteiro para aumentar a velocidade de processamento). São usados apenas valores inteiros para $g(d)$.

¹¹GATH & HOORY descrevem um método desenvolvido para detectar os parâmetros de grupos de anéis de forma elíptica. Usaram este método para fazer reconstrução de imagem tridimensional a partir de imagens obtidas por ressonância magnética de seções transversais do tórax humano a fim de observar o ventrículo esquerdo de um coração.

Eles denominaram o algoritmo desenvolvido de "Fuzzy K-Ellipse" (FKE). A elipse é recuperada a partir da localização de seus focos e do comprimento do eixo

maior da elipse. A lógica Fuzzy e aproximação por mínimos quadrados são usados para determinar os pontos que pertencem ao contorno da elipse.

⁰³BHANDARKAR (1994) desenvolveu um método denominado de Transformada de Hough Generalizada Utilizando Pesos (WGHT), que consiste em estabelecer pesos que são vistos como valores-membro de uma função Fuzzy e têm como finalidade a redução das interpretações espúrias que aparecem quando é aplicado o GHT na detecção das formas desejadas. Para reduzir a dimensão do acumulador, os votos gerados durante o processo de Hough não são todos acumulados, mas apenas uma parte deles, escolhidos de forma aleatória, são acumulados.

2.8 - Métodos que utilizam dispositivos especiais

Segundo ³⁴RHODES et alii (1988), um algoritmo que utilize a Transformada de Hough pode ser implementado num *chip* VLSI. A maior vantagem detectada neste tipo de implementação é a grande redução do tempo de processamento necessário para realizar o algoritmo.

Quando se dispõe de pouca memória existe a necessidade de desenvolver uma técnica que diminua a quantidade de informação a ser armazenada. É interessante também reduzir o custo de processamento. Conforme ⁰¹ALBANESI & FERRETI (1991), tal resultado pode ser conseguido através de uma implementação paralela da geração dos incrementos e subsequente análise do espaço de parâmetros com redução de sua complexidade. Eles apresentaram uma versão de um algoritmo que usa uma quantidade mínima de memória possível, mantendo sua eficiência quanto à confiabilidade das formas detectadas e também quanto à fácil integração em baixo custo e alta velocidade. Tudo isto é proposto na forma de um dispositivo VLSI especial para estas finalidades.

O uso de uma arquitetura especial também é proposto por ²⁴KUMAR et alii (1994). Eles apresentaram duas técnicas para realizar a detecção de círculos através da Transformada de Hough utilizando uma arquitetura do tipo SIMD que executa um conjunto de dados bidimensionais, pois esta arquitetura inclui entre suas

características: uma matriz bidimensional de processadores interconectados, e memória local associada a cada processador, permitindo que todos os processadores executem a mesma instrução simultaneamente [⁴⁹KITTLER & DUFF (1986)].

Na primeira técnica foi implementado um algoritmo que realiza a Transformada de Hough a partir da magnitude do gradiente.

O método consiste em realizar a Transformada de Hough para detectar formas circulares utilizando o SIMD como uma malha capaz de armazenar o número de vezes que círculos de mesmo raio, traçados a partir dos pontos de borda (que têm magnitude do gradiente diferente de zero), passam pela célula correspondente da malha acumuladora. Cada célula é incrementada paralelamente às outras, pois cada uma corresponde a um processador do SIMD.

A segunda técnica realiza a detecção acrescentando a direção do gradiente.

A grande vantagem deste método em relação à primeira é a economia de tempo de processamento porém traz consigo a desvantagem da perda de robustez, característica do uso da direção do gradiente.

2.9 - Conclusão

As variações em torno do método de Hough procuram fundamentalmente resolver o problema da grande quantidade de dados que precisa ser armazenada quando este método é usado, principalmente quando o número de parâmetros necessários para definir o objeto a ser identificado aumenta. Porém estes métodos diminuem a capacidade de detecção quando a imagem possui ruído ou ocultação de parte do objeto. Como a quantidade de parâmetros necessários para definir uma forma circular não é tão crítica quanto à necessária para definir formas mais complexas (como a elíptica, por exemplo), é feita uma opção, neste trabalho, de utilizar a Transformada de Hough sem variações, mantendo assim a grande capacidade deste método de realizar a detecção em imagens ruidosas, ou quando o objeto a ser detectado estiver parcialmente oculto.

3 - O Ambiente de Processamento

3.1 - Introdução

A fim de realizar a detecção de formas circulares desenvolveu-se um programa denominado **Scanc**, que dispõe dos seguintes recursos:

- Representação num monitor de vídeo, tipo varredura, de imagens de 16(dezesesseis) ou 256(duzentos e cinquenta e seis) cores.
- Mudança de *palette*, contida no ponteiro de imagem, para tons de cinza.
- Execução de duas técnicas de suavização de uma imagem: média entre 9(nove) e convolução com uma função Gaussiana.
- Detecção de bordas por dois métodos: Sobel e OU-exclusivo.
- Binarização da imagem pela técnica do *Threshold*.
- Detecção de formas circulares através de duas técnicas: Transformada de Hough e *Template Matching*.
- Filtragem, através de três métodos, para seleccionar possíveis formas circulares na imagem: Filtro Equalizador; Filtro Comparativo e Filtro Fuzzy.
- Permitir a visualização da presença de formas circulares através de quatro formas: desenhando círculos nas posições correspondentes; mostrando os dados dos círculos numa janela de diálogo; visualizando no plano a constituição da sobreposição dos sub-espacos acumuladores e finalmente exportando para o Matlab ou equivalente, os valores referentes à sobreposição dos sub-espacos acumuladores a fim de visualizá-la tridimensionalmente.

A Fig 3.1 ilustra um Diagrama de Blocos do **Scanc**.

3.2 - Disposição da imagem na memória

O número de cores ou tons de cinza da imagem que irá ser efetivamente mostrada no monitor de vídeo determina a profundidade do *pixel* em termos de bits necessários para representá-lo. Esta descrição através da profundidade do *pixel* é chamada de Mapa de Bits. Uma imagem binária pode ser representada por apenas um

Fig. 3.1 - Diagrama de Blocos do Scanc.

bit por *pixel*. Já uma imagem de 16 (dezesesseis) cores ou níveis de cinza precisa de 4(quatro) bits para cada *pixel*, pois sua combinação irá fornecer as 16(dezesesseis) possibilidades. Por isso, para uma imagem de 256(duzentos e cinquenta e seis) cores ou níveis de cinza, serão necessários 8(oito) bits por *pixel*, isto é, a quantidade de *pixels* por byte depende do número de tons de cinza ou do número de cores na imagem. A figura 3.2a ilustra um byte ocupado por 8(oito) *pixels* caracterizando assim uma imagem binária. Já as figuras 3.2b e 3.2c caracterizam imagens de 16(dezesesseis) e 256(duzentos e cinquenta e seis) cores respectivamente.

Fig 3.2 - a) Byte dividido por oito pixels; b) dividido por dois pixels; c) byte correspondente a um pixel.

A quantidade de memória, em bits, requerida por uma imagem é dada por:

altura (nº de linhas) x largura (em pixels) x nº bits/pixel

Enquanto que, a quantidade de memória requerida em bytes é dada por:

altura (nº de linhas) x largura (em pixels) x nº bytes/pixel

Por exemplo, uma imagem de 300(trezentos) pixels de largura, com 240(duzentos e quarenta) linhas e 16(dezesseis) cores, requer:

$$320 \times 240 \times 4 = 307.200 \text{ bits, e}$$

$$320 \times 240 \times 0,5 = 38.400 \text{ bytes}$$

Uma imagem descrita desta forma deve ser armazenada num Mapa de Bits Independente do Dispositivo (MBID); no **Scanc** serão usados os arquivos MBID com extensão BMP. Esse tipo de arquivo foi o escolhido para figurar nesse trabalho pela sua simplicidade de manipulação. É constituído por um cabeçalho denominado de *header*, que contém as informações necessárias para utilização do arquivo, seguido pela imagem propriamente dita.

No **Scanc**, apenas arquivos de extensão BMP contendo imagens de 16(dezesseis) ou 256(duzentos e cinquenta e seis) cores serão admitidos.

A imagem contida no arquivo pode ser tratada de duas maneiras durante seu processamento: pode ser colocada na memória convencional ou diretamente na memória de vídeo do PC.

A memória de vídeo apresenta o inconveniente de estar sempre sujeita a intervenção do controlador de vídeo, o que torna difícil seu uso como memória de imagem, já que essa memória pode, também, ser comandada por outro programa qualquer (caso de Screen Saver do Windows, por exemplo).

Já a memória convencional do PC está sujeita apenas ao programa que comanda aquela área de memória, e por isso, foi escolhida para ser usada no programa que figura neste trabalho, o **Scanc**. Para a visualização da imagem foi incluído um comando no programa que a transfere para a memória de vídeo.

A Transformada de Hough para círculos requer que um espaço em memória seja reservado para o armazenamento de valores num acumulador. Tais valores devem ser tomados como inteiros não sinalizados (16 bits) que podem acumular valores entre 0(zero) e 65.535(sessenta e cinco mil, quinhentos e trinta e cinco).

Como a Transformada de Hough requer grande quantidade de memória para o acumulador, o **Scanc** pode não funcionar adequadamente se o PC onde ele for instalado não tiver pelo menos 8(oito) Mbytes disponível em memória convencional.

Se fizermos uma varredura na imagem utilizando a Transformada de Hough para 5(cinco) valores de raio apenas, precisaremos de um espaço na memória para um acumulador de números inteiros (dois bytes), de 76.800(setenta e seis mil e oitocentos) bytes vezes 5(cinco) e vezes 2(dois) já que precisaremos deste número de bytes em inteiros (cada inteiro equivale a dois bytes). O resultado é um acumulador que necessita de 768.000(setecentos e sessenta e oito mil) bytes disponíveis, ou podemos dividir esse número por 1.024 e obter 750 Kbytes.

O problema de se desenvolver o software no sistema operacional MS-DOS é que a memória baixa do PC é de apenas 640 Kbytes, onde deverá estar presente pelo menos o sistema operacional (normalmente o MS-DOS) e o programa que fará o processamento da imagem a fim de detectar as formas circulares. Ou seja, teremos disponível menos de 640 Kbytes para o acumulador enquanto que este precisaria de 750(setecentos e cinquenta) Kbytes para varrer apenas 5(cinco) valores de raio.

O problema acima poderia ser contornado diminuindo o tamanho da imagem, o que reduziria muito sua qualidade, pois menos *pixels* seriam usados para representá-la. Além disso, ficaríamos limitados a uma varredura muito pequena em termos de número de valores de raios pesquisados.

Outra solução é a alocação, para o acumulador, da memória estendida do PC, pois esta memória dispõe de alguns Mbytes extras. O problema aqui é a complexidade do processo de alocação e mais ainda da liberação da memória que precisa ser feita quando esta não for mais necessária.

Uma forma segura e simples de se ter acesso à memória estendida é usar o ambiente Windows, que proporciona ao programador a possibilidade de poder reservar e liberar um espaço na memória estendida da mesma maneira como o faz para a memória baixa. Para a elaboração do programa **Scanc** foi usado o software Borland C++ 4.0 para esse ambiente.

3.3 - As estruturas *headers* do arquivo MBID

O cabeçalho do arquivo MBID começa com uma seção definida pela estrutura `BITMAPFILEHEADER` cujos campos estão descritos na tabela 3.1 [⁴⁸KAY & LEVINE (1992)].

Campo	Tamanho	Descrição
<i>bfType</i>	UINT	Os bytes “BM” (de “mapa de bits”)
<i>bfSize</i>	DWORD	Tamanho total do arquivo
<i>bfReserved1</i>	UINT	Em 0(zero)
<i>bfReserved2</i>	UINT	Em 0(zero)
<i>bfOffBits</i>	DWORD	Distância em bytes entre fim desta estrutura e o começo dos bits de imagem

Tabela 3.1 - Descrição dos campos da estrutura `BITMAPFILEHEADER`.

Este cabeçalho é seguido por outro definido pela estrutura `BITMAPINFOHEADER` cujos campos estão descritos na tabela 3.2.

Campo	Tamanho	Descrição
<i>biSize</i>	DWORD	Tamanho dessa estrutura em bytes
<i>biWidth</i>	LONG	Largura do mapa de bits em pontos
<i>biHeight</i>	LONG	Altura do mapa de bits em pontos
<i>biPlanes</i>	WORD	Número de planos da imagem, em 1
<i>biBitCount</i>	WORD	Bits de cor por ponto (1, 4, 8, ou 24)
<i>biCompression</i>	DWORD	Tipo de compressão (0 para nenhuma)

<i>biSizeImage</i>	DWORD	Tamanho do mapa de bits em bytes (necessário somente se a compressão está sendo usada)
<i>biXPelsPerMeter</i>	LONG	Resolução horizontal em pontos por metro
<i>biYPelsPerMeter</i>	LONG	Resolução vertical em pontos por metro
<i>biClrUsed</i>	DWORD	Número de cores usadas na figura
<i>biCtrImportant</i>	DWORD	Número de cores importantes na figura

Tabela 3.2 - Descrição dos campos da estrutura BITMAPINFOHEADER.

Todos os campos que seguem *biBitCount* podem ser deixados em 0(zero) para os valores padrões (ou podem nem mesmo aparecer no arquivo). Assim, a estrutura pode ter apenas 16(dezesseis) bytes de tamanho.

Essa estrutura é seguida então por uma tabela de cores construída a partir de estruturas RGBQUAD, que definem um valor de cor RGB(vermelho, verde e azul), com campos descritos na tabela 3.3.

Campo	Tamanho	Descrição
<i>rgbBlue</i>	BYTE	Intensidade do azul
<i>rgbGreen</i>	BYTE	Intensidade do verde
<i>rgbRed</i>	BYTE	Intensidade do vermelho
<i>rgbReserved</i>	BYTE	Em 0(zero)

Tabela 3.3 - Descrição dos campos da estrutura RGBQUAD.

O número de estruturas RGBQUAD é normalmente determinado pelo campo *biBitCount*, pois serão necessários $2^{(biBitCount)}$ dessas estruturas para montar a tabela. Porém se o campo *biClrUsed* for diferente de 0(zero), então este contém o número de estruturas RGBQUAD.

3.4 - Manipulação do arquivo de entrada da imagem

3.4.1 - Leitura e gravação do arquivo MBID

No **Scanc** a imagem de entrada é sempre um arquivo MBID com extensão BMP. Essa imagem é obtida através da função de leitura, acessada através do menu ilustrado na figura 3.3 (que é organizado na lista de recursos visualizada no apêndice 2) a partir de uma outra função denominada procedimento de janela - na programação C++ para o ambiente Windows, as funções são chamadas a partir do procedimento de janela que executa uma tarefa baseando-se na mensagem enviada e em seguida retorna o controle ao ambiente - que evoca a caixa de diálogo padrão para leitura, vista na figura 3.4[apêndice 1 seção 1.16].

Fig. 3.3 - Menu do Scanc acessando a mensagem de abertura de arquivo.

Fig. 3.4 - Caixa de diálogo padrão do Windows para leitura.

Depois que o arquivo MBID é selecionado na caixa de diálogo, o nome do arquivo é passado como argumento para a função de leitura [apêndice 1 seção 1.6] que irá abri-lo e ler seu início numa estrutura `BITMAPFILEHEADER` (`bmfh`). Esta estrutura é declarada como uma variável global, e pode ser chamada diretamente de qualquer lugar no programa.

Os dois primeiros bytes (o campo *bfType*) são verificados, em busca da assinatura “BM” que indica que o arquivo aberto é mesmo um MBID. Em seguida o campo *bfSize* é usado para calcular o tamanho do arquivo menos o tamanho da estrutura `BITMAPFILEHEADER`. A função usa então o resultado dessa subtração para reservar um bloco de memória que conterá a estrutura `BITMAPINFOHEADER`, as estruturas `RGBQUAD` e os bytes da imagem.

A variável `lpMbid` usada para armazenar o ponteiro referente à memória de imagem é definida como um ponteiro do tipo *huge*, já que muitos MBIDS são maiores do que 64 KB e o compilador precisa gerar um código para percorrer os múltiplos segmentos do MBID. Nesse ponto, o restante do arquivo é simplesmente gravado num bloco de memória previamente reservado e localizado pelo ponteiro `lpMbid` em blocos de 32 KB.

A função de leitura chama a função *TamCabecalhoInfoMbid*, que simplesmente extrai a primeira `DWORD` do bloco de memória, e verifica se o cabeçalho tem 12 bytes de comprimento (caso em que o programa está lidando com um arquivo de mapa de bits do OS/2 1.x) ou pelo menos 16 bytes de tamanho (em

cujo caso é um MBID do Windows). Essa função (leitura) retorna o ponteiro lpMbid ao procedimento de janela, onde estarão nessa ordem, as estruturas BITMAPINFOHEADER, RGBQUAD e os bytes da imagem.

O procedimento de janela pode, posteriormente, também acessar uma função de gravação [apêndice 1 seção 1.6] chamando antes a caixa de diálogo padrão do Windows, para gravação. Nessa caixa é escolhido ou escrito um nome para o arquivo, que será passado como argumento a esta função juntamente com o ponteiro lpMbid. A estrutura bmfh (BITMAPFILEHEADER), que é global, é então escrita no arquivo aberto para gravação, seguida pelo conteúdo atual do ponteiro lpMbid.

3.4.2 - Funções que extraem as características da imagem

Durante a leitura do arquivo MBID é criado um ponteiro (lpMbid) que aponta para a primeira posição do espaço da memória, onde serão situadas as estruturas BITMAPINFOHEADER, RGBQUAD e os bytes da imagem.

No **Scanc** várias funções são criadas a partir desse ponteiro a fim de extrair as características da imagem [apêndice 1 seção 1.3]. Como o BITMAPINFOHEADER é uma estrutura, seus campos podem ser diretamente apontados a partir do endereço do ponteiro lpMbid. Com esse procedimento são extraídos o tamanho da própria estrutura BITMAPINFOHEADER, o número de cores da imagem (que também pode ser o número de níveis de cinza) e a largura e altura da imagem em pontos.

Uma característica inerente aos arquivos formadas por mapa de bits é o fato de terem sempre largura múltipla de quatro. Portanto quando o arquivo MBID é criado, se a largura da imagem não for múltipla de quatro, zeros são acrescentados à imagem até que esta possa ser arquivada como um MBID típico. Exemplo de uma imagem assim é mostrado na figura 3.5 onde a imagem é de 6x1x16, e dois *pixels* nulos são acrescentados.

Fig. 3.5 - Diagrama do mapa de bits de uma imagem 6x1x16.

Porém esta diferença não é incluída no campo *biWidth* da estrutura *BITMAPINFOHEADER*, que considera apenas a largura real da imagem. Um cuidado especial, então, deve ser tomado quando este campo não tiver valor múltiplo de quatro.

No *Scan*, isto é feito acrescentando esse deslocamento à largura lida em *biWidth*, todas as vezes que esse parâmetro for usado como limitador de índice. O deslocamento é calculado dividindo-se o conteúdo do campo *biWidth* por 4(quatro). Se o resto for 0(zero) não há deslocamento, mas se não for 0(zero), toma-se o deslocamento como sendo 4(quatro) subtraídos do resto [apêndice 1 seção 1.5].

As intensidades de cor (vermelho, verde e azul) de cada ponto da imagem também podem ser extraídas lendo-se as estruturas *RGBQUAD* do ponteiro *lpMbid* [apêndice 1 seção 1.3].

O tamanho da tabela de cores ou *palette* do arquivo (referente aos bytes das estruturas *RGBQUAD*) deve ser calculado multiplicando-se o número de cores da imagem pelo tamanho em bytes da estrutura *RGBQUAD*. Este comprimento deve ser somado ao tamanho encontrado para a estrutura *BITMAPINFOHEADER* e ao primeiro endereço do ponteiro *lpMbid*. O resultado obtido é o endereço do primeiro byte da imagem, que neste trabalho é denominado de endereço da imagem.

3.5 - A Visualização da Imagem

O ponteiro `lpMbid` é usado pelo procedimento de janela para mostrar na tela a imagem contida no arquivo MBID. No final do comando de abertura e leitura do arquivo MBID, é colocado na fila de mensagens um comando que invalida a janela, forçando o Windows a fazer sua atualização. Uma das duas funções seguintes que utilizam as informações retiradas do ponteiro `lpMbid` é então usada para mostrar o mapa de bits na tela: *SetDIBitsToDevice* para mostrar a imagem em seu tamanho real como ilustrado na figura 3.6 ou *StretchDIBits* para mostrar a imagem expandida na janela [apêndice 1 seção 1.16]. Essas funções permitem a exibição de qualquer retângulo do mapa de bits em uma determinada largura e altura lógicas na tela, definidas pela janela ativa do programa.

Fig. 3.6 - Conteúdo do arquivo MBID pintado na janela.

3.5.1 - As mudanças de *palette*

3.5.1.1 - Mudança da *palette* da janela

A janela criada pelo Windows, em algumas configurações de tela, está preparada para receber uma quantidade máxima de 16(dezesseis) cores, e se o arquivo MBID possuir 256(duzentos e cinquenta e seis) cores a imagem na tela parecerá borrada, pois apenas 16(dezesseis) serão usadas para mostrá-la. Portanto, se após a leitura do arquivo MBID for constatado que este representa uma imagem de 256(duzentos e cinquenta e seis) cores, haverá a necessidade de adequar a janela ao arquivo. Isso é feito igualando-se a *palette* da janela com a *palette* da estrutura RGBQUAD do arquivo MBID.

Na verdade apenas 236(duzentos e trinta e seis) cores podem ser utilizadas, pois as 20(vinte) restantes são reservadas ao Windows [apêndice 1 seção 1.7].

3.5.1.2 - Mudança da *palette* para níveis de cinza

Em diversas aplicações, a visualização de uma imagem em níveis de cinza é preferível àquela em cores, pela facilidade de análise dessa imagem, como acontece em imagens médicas. Por essa razão é importante possuir um dispositivo que mude a *palette* da imagem para tons de cinza.

No Windows não é suficiente mudar o que estiver sendo mostrado na tela, através da mudança do conteúdo na memória de vídeo, pois a qualquer momento pode haver na fila uma mensagem de atualização do conteúdo da janela; portanto, é necessário mudar o conteúdo das estruturas RGBQUAD, apontado por *lpMbid*, e chamar a função *SetDIBitsToDevice* ou mesmo a função *StretchDIBits*. Essa mudança consiste em percorrer todas as estruturas, igualando os valores do vermelho, verde e azul de cada estrutura [apêndice 1 seção 1.7]. Com essa modificação a imagem será mostrada em níveis de cinza.

3.5.2- Um tratamento diferenciado para imagens de 16(dezesseis) cores

As imagens de 16(dezesseis) cores possuem profundidade de *pixel* igual a 4(quatro) bits, ou seja, cada byte de imagem do arquivo contém 2(dois) *pixels*.

A dificuldade neste caso é endereçar os *pixels* contidos nos 4(quatro) bits menos significativos de cada byte de imagem, pois não existe identificador de tipo para esta quantidade de bits. Em linguagem C consegue-se endereçar um número mínimo de 8(oito) bits identificados como *char* ou BYTE. Se usarmos esse identificador para imagens de 16(dezesseis) cores ou tons de cinza, os quatro bits menos significativos de cada byte serão tomados como parte de um valor de 8(oito) bits fornecendo resultados incorretos.

Usando-se como exemplo uma imagem com dimensões 320x240x16, a memória endereçada pelo ponteiro *lpMbid*, deslocado ao endereço de imagem, está organizado como na figura 3.7a.

Se os *pixels* forem endereçados byte a byte, o primeiro *pixel* será lido como $11100101_2 = 229$ (duzentos e vinte e nove) e não como dois *pixels* contendo $1110_2 = 14$ (quatorze), e $0101_2 = 5$ (cinco), como deveria ser. Além disso, a imagem seria interpretada como contendo apenas 38400 (trinta e oito mil e quatrocentos) *pixels* resultando numa imagem sem sentido visual e cortada pela metade.

A solução escolhida para o **Scanc** foi reorganizar, numa função, a memória endereçada pelo ponteiro *lpMbid* de modo que todos os bits correspondentes aos *pixels* passem a ocupar apenas os quatro bits menos significativos de cada byte. Essa nova organização é ilustrada na figura 3.7b.

Apesar desse procedimento ocupar um espaço extra em memória, ele permite endereçar todos os *pixels* da imagem. Para realizar essa tarefa é preciso primeiramente reservar um espaço em memória para um ponteiro auxiliar onde deverão ser colocados os *pixels* de modo a ocuparem apenas os 4(quatro) bits menos significativos de cada byte de imagem. Portanto esse novo ponteiro tem comprimento em bytes igual ao número de pontos da imagem.

Fig. 3.7 - a) Seção da memória de uma imagem de 16 cores ou tons de cinza; b) Os bits correspondentes aos pixels dessa imagem passaram a ocupar apenas os bytes menos significativos.

Depois da alocação do ponteiro, os *pixels* são separados e colocados um em cada byte [apêndice 1 seção 1.5]. Assim as imagens de 16(dezesseis) cores poderão ser tratadas de maneira idêntica às de 256(duzentos e cinquenta e seis), pois cada vez que for detectada uma imagem de 16(dezesseis) cores ou tons de cinza, essa função será acionada.

Uma segunda forma de tratar esse tipo de imagem seria a utilização de máscaras para separar os *pixels*, como está ilustrado nas figuras 3.8a e 3.8b.

Fig. 3.8 - a) Template de aquisição dos quatro bits mais significativos em cada byte e, b) template de aquisição dos quatro bits menos significativos em cada byte.

Os *pixels* seriam separados apenas no momento em que fossem operados com um valor, ou convoluídos com um *template*, não necessitando assim de um ponteiro auxiliar para armazená-los. Todavia este método aumenta a quantidade de processamento necessário para qualquer convolução da imagem com um *template* pois o valor de cada *pixel* teria que ser separado todas as vezes que o *template* a ser convoluído fosse deslocado sobre esse *pixel* enquanto em que pelo método implementado, o valor de cada *pixel* só é calculado uma vez e acumulado antes da convolução.

3.6 - Conclusão

Um dos problemas encontrados na implementação da Transformada de Hough em microcomputadores pessoais é a quantidade de memória que a técnica requer.

O Ambiente Windows tornou viável a implementação dessa técnica pela facilidade de acesso à memória estendida.

4 - O Pré-processamento

4.1 - Introdução

A Transformada de Hough, e igualmente a técnica de *Template Matching* são aplicadas sobre imagens de bordas, isto é, imagens que contém apenas os contornos dos objetos nelas contidos. Esse procedimento, “reduz drasticamente a quantidade de dados a serem processados, e ao mesmo tempo preserva a informação estrutural sobre a forma do objeto analisado” [⁰⁴CANNY (1986)]. Por isso, é interessante salientar que essas operações que denominamos de pré-processamento terão grande influência na performance do processo de detecção [²⁵LEAVERS (1992)].

Com objetivo de auxiliar o usuário do **Scanc** foram então incluídos alguns recursos para binarização da imagem de entrada, os quais são descritos nos itens seguintes.

4.2 - A suavização da imagem

Muitas vezes a imagem obtida apresenta muito ruído. É preciso então suavizá-la, para que pontos isolados nessa imagem, que apresentem discrepâncias em relação a seus vizinhos, sejam melhorados, evitando o aparecimento de falsos pontos de bordas. Existem diversas técnicas para realizar a suavização de imagens [³⁵RITTER et alli (1990), ⁵⁰MARR (1980)]. Serão descritos a seguir aquelas que foram implementadas neste trabalho.

4.2.1 - Suavização através da média

Pode-se suavizar a imagem operando-a com um *template* unitário 3x3 (três *pixels* de largura por três de altura) que deve ser convoluído com a imagem. O conteúdo de suas posições deve ser somado e dividido por 9(nove) já que ele cobre este número de posições [⁴⁶GONZALES & WINTZ (1987)].

4.2.2 - Suavização através da Gaussiana

Outra maneira de suavizar a imagem é convoluindo-a com um *template* cujos valores são calculados através da Gaussiana obtida a partir da equação

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left[-\frac{(x^2 + y^2)}{2\sigma^2}\right] \quad (4.1)$$

que é chamada de Operador Gaussiana 2D. Esta equação também pode ser expressa em termos de coordenadas polares. Como

$$x^2 + y^2 = r^2 \quad (4.2)$$

então,

$$G(r) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{r^2}{2\sigma^2}\right) \quad (4.3)$$

Dessa maneira, podemos tratar o *template* como circular, variando o valor de r .

A convolução é uma operação entre funções definida como

$$i(x) * t(x) = \int_{-\infty}^{+\infty} i(\tau) t(x - \tau) d\tau \quad (4.4)$$

onde $i(x)$ corresponde à imagem e $t(x)$ corresponde ao *template*. O resultado instantâneo da convolução corresponde à área instantânea de intersecção entre as duas funções.

O tamanho do *template* a ser convoluído com a imagem foi escolhido empiricamente com dimensões 5x5. É necessário então calcular 3(três) valores do *template*. Os 22(vinte e dois) restantes são atribuídos por simetria já que se trata de um *template* circular.

O programa do apêndice 5, foi implementado para calcular o conteúdo desse *template*. Com esse programa pôde-se escolher os valores mais adequados para o desvio padrão fornecendo ao **Scanc** três Gaussianas com curvas características bem diferentes entre si, os *templates* são ilustrados na figura 4.1, e no **Scanc** podem ser selecionados pelo usuário a partir do menu, conforme é ilustrado na figura 4.2.

Fig. 4.1 - Templates circulares contendo Gaussianas em 2D com a) desvio padrão igual a 0,2; b) desvio padrão igual a 0,4 e c) desvio padrão igual a 1.

Fig. 4.2 - Menu de escolha do desvio padrão para a Gaussiana.

Nota-se que os *templates* obtidos são normalizados. A imagem também deve ser normalizada já que as multiplicações envolvidas na convolução não devem ter

produto maior do que 255(duzentos e cinquenta e cinco). Na seleção do *template* deve ser considerado o nível de ruído da imagem, lembrando que um desvio padrão maior irá proporcionar uma maior suavização, mas com redução do contraste.

A convolução do *template* 5x5 com a imagem consiste em posicionar o *template* no canto superior esquerdo da imagem, e deslocá-lo da esquerda para a direita e de cima para baixo, percorrendo assim todos os pontos da imagem.

A cada deslocamento cada elemento do *template* é multiplicado com o nível de cinza do *pixel* correspondente na imagem. Os termos produtos são somados e o resultado obtido corresponde ao nível de cinza do *pixel* da nova imagem.

4.3 - Detectores de borda

Como já foi mencionado anteriormente, para realizar a detecção de formas dos objetos pesquisados não é necessário analisar completamente a imagem, o que demandaria muito tempo e processamento, mas apenas os contornos das figuras contidas na imagem.

Diz-se que um *pixel* é considerado como borda na imagem, quando o contorno da figura passa sobre sua posição^[45]BOYLE & THOMAS (1988)]. Esse *pixel* pode ser detectado examinando-se as mudanças de intensidades próximas a ele. Essas mudanças, também chamadas de Diferenças do Gradiente são descritas como sendo a diferença da função imagem entre dois pontos vizinhos e pode ser equacionada como

$$\Delta_1 = f(i, j) - f(i + 1, j + 1) \quad (4.5)$$

No **Scanc** foram implementados dois detectores de bordas: o OU-exclusivo que foi escolhido por sua simplicidade e o operador Sobel que foi escolhido por ser usado com muita frequência nas literaturas que tratam de Processamento de Imagens. A escolha entre esses dois métodos no momento da aplicação do programa é feita, pelo usuário, através do menu, na forma indicada pela figura 4.3.

4.3.1 - Detecção de bordas por gradientes espaciais

A detecção de bordas pode ser feita convoluindo duas ondas quadradas e simétricas com a imagem. Porém, resultado melhor é obtido se a primeira derivada de uma Gaussiana for usada no lugar das ondas quadradas.

Fig. 4.3 - Escolha do detetor de borda.

A idéia de usar valores simétricos para fazer detecção de borda está presente no operador Sobel descrito por ³⁵RITTER et alii (1990), que convolui dois *templates* de valores discretos com a imagem.

Conforme ⁵⁴SCHALKOFF (1989) o gradiente de uma função imagem, $f(x,y)$ é definido como:

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} \quad (4.6)$$

Os operadores da matriz podem ser aproximados para

$$\begin{aligned}\frac{\partial f(x,y)}{\partial x} &\approx \frac{f(x+\Delta x,y) - f(x,y)}{\Delta x} \\ \frac{\partial f(x,y)}{\partial y} &\approx \frac{f(x,y+\Delta y) - f(x,y)}{\Delta y}\end{aligned}\quad (4.7)$$

Aplicar estes operadores numa imagem corresponde à convolução da função imagem discretizada com os seguintes operadores:

$$\begin{bmatrix} -1 & 1 \end{bmatrix} \quad \text{e} \quad \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

O primeiro operador corresponde ao gradiente na direção da variação de x e o segundo ao gradiente na direção da variação de y , e devem ser convoluídos com a função imagem discretizada. Porém o resultado dessas operações tendem a se posicionar no ponto médio entre os *pixels*, já que, expandindo a função $f(x+\Delta x)$ numa série de Taylor (para simplificar a expansão é feita numa dimensão apenas), ela ficará na seguinte forma:

$$f(x+\Delta x) = f(x) + \Delta x f'(x) + \frac{(\Delta x)^2}{2!} f''(x) + \dots \quad (4.8)$$

Usando o teorema do valor médio a série pode ser rescrita como

$$f(x+\Delta x) - f(x) = \Delta x f'(x) + \frac{(\Delta x)^2}{2} f''(\xi) \quad (4.9)$$

onde $\xi \in [x, x+\Delta x]$

Portanto, a Aproximação Derivativa será dada por

$$D_1(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x} \quad (4.10)$$

e estará localizada aproximadamente no ponto médio entre $f(x)$ e $f(x+\Delta x)$, que é equivalente a estar entre dois *pixels* vizinhos. Na figura 4.4 nota-se que este resultado interpolado também é verdade para a direção de variação de y .

Fig. 4.4 - A Aproximação Derivativa não está centrada num ponto.

Uma forma alternativa para a Aproximação Derivativa é dada pela equação

$$D_2(x) = \frac{f(x + \Delta x) - f(x - \Delta x)}{2(\Delta x)} \quad (4.11)$$

que fará com que o resultado fique sobre o ponto central x localizado entre $x+\Delta x$ e $x-\Delta x$. Isso equivale a convoluir a função imagem com os operadores

$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ para o gradiente na direção da variação de x

$\begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix}$ para o gradiente na direção da variação de y .

Pode-se transformar esses operadores em *templates* circulares obtendo-se

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{para o gradiente na direção da variação de x}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad \text{para o gradiente na direção da variação de y.}$$

4.3.2 - O operador Sobel

Preparado para enfatizar o *pixel* central, da operação entre o *template* e a imagem dando-lhe maior amplitude, o operador Sobel deu origem aos *templates* circulares

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{e} \quad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

É fácil perceber que o primeiro deve ser aplicado para obtenção do gradiente na direção da variação de x enquanto que o segundo é para a obtenção na direção da variação de y.

Esses *templates* são convoluídos paralelamente com a imagem, isto é, o mesmo *loop* no **Scanc** é aproveitado para operar a convolução de ambos os *templates* com a imagem [apêndice 1 seção 1.9].

No **Scanc** o resultado final das somas em cada posicionamento do *template* não é atribuído ao *pixel* central da parte da imagem que está casada com o *template* como era de se esperar, em vez disso o resultado é atribuído ao canto superior esquerdo desta seção da imagem. Este procedimento faz com que o erro inerente das operações com *templates* seja todo deslocado para os limites direito e inferior da imagem.

4.3.3 - O operador OU-exclusivo

Outra maneira simples e eficaz de se detectar as bordas de figuras contidas numa imagem é aplicar o operador OU-exclusivo entre pontos conectados da imagem como foi abordado por ¹³GONZAGA & FRANÇA (1992). Neste método o resultado será um nível lógico igual a 1(um) se, e somente se, os *pixels* tiverem intensidades de tons de cinza diferentes.

Para garantir que pontos de borda não sejam perdidos nessa detecção deve-se fazer varreduras tanto na horizontal como na vertical.

O **Scanc** realiza este tipo de detecção subtraindo no mesmo *loop* a magnitude de um *pixel* da magnitude de seus vizinhos à direita e abaixo [apêndice 1 seção 1.9]. Os dois resultados em cada interação são somados entre si. O *pixel* em questão é tomado como ponto de borda se esta soma exceder um valor mínimo pré-estabelecido. Nesse caso seu tom de cinza será atribuído como 0(zero), caso contrário como 255(duzentos e cinquenta e cinco) binarizando assim a imagem. Todos os *pixels* da imagem são rastreados.

4.4 - A binarização da imagem

Após a detecção de bordas por Sobel verifica-se que a imagem resultante não é binária como quando se faz a detecção pelo operador OU-exclusivo. O **Scanc** foi feito propositalmente dessa maneira para permitir que a imagem de magnitudes dos gradientes seja obtida, já que Sobel é um operador gradiente, e possa ser gravada em arquivo MBID, que pode ser aberto pelo programa a qualquer instante, não havendo assim a necessidade de refazer a imagem das magnitudes do gradiente se a escolha do *threshold*, usado para binarizar a imagem, não for adequado para a imagem em questão.

No **Scanc** foi implementada então uma função que binariza a imagem em nível 0(zero) e 255(duzentos e cinquenta e cinco) através da escolha de um valor limite (*threshold*) onde magnitudes do ponto maiores que este valor fazem com que a

posição de memória correspondente ao ponto em questão seja preenchida com nível de cinza igual a 0(zero) e magnitudes do ponto menores fazem com que o ponto seja preenchido com nível de cinza igual a 255 (duzentos e cinquenta e cinco)[apêndice 1 seção 1.10]. Uma imagem de gradientes resultante da aplicação do operador Sobel no momento em que o usuário requisita sua binarização, pode ser vista na figura 4.5. O valor do *threshold* é pedido conforme pode ser visto na figura 4.6.

Fig. 4.5 - Imagem de gradientes resultante da aplicação do operador Sobel.

A escolha do valor do *threshold* durante a binarização da imagem é feita diretamente pelo usuário de forma empírica testando e observando a imagem de bordas resultante. Também chamado de Método da Tentativa e Erro [⁵⁵YOUNG & FU (1986)] este método proporciona uma melhor escolha para cada caso, já que o **Scanc** tem aplicação em diferentes áreas.

Um valor fixo, ou mesmo um algoritmo para calcular este valor, poderia ser usado num processo dedicado, pois as características da forma inspecionada já seria previamente conhecida.

Fig. 4.6 - O programa requisita a inserção do valor do threshold.

É grande a importância do pré processamento para a detecção de formas circulares, mais precisamente da detecção de bordas, pois, na verdade, os pontos de borda constituem os dados a serem analisados pelo detector, portanto, quanto maior for a distorção das bordas, mais imprecisa será a detecção das formas circulares.

5 - Detectores

5.1 - Introdução

A fim de analisar o desempenho da Transformada de Hough na detecção de formas circulares foi implementado o método de ⁰⁹DUDA & HART (1972), onde círculos de raio r são traçados numa grade acumuladora conforme ilustrado na figura 5.1 e a forma circular é encontrada através de uma sucessão dessas grades, uma para cada valor de raio pesquisado. A este arranjo damos o nome de Arranjo Acumulador. Este método foi escolhido pela sua simplicidade de implementação. Além disso, possui grande robustez na detecção da forma, característica importante na avaliação de imagens quando se quer realizar um estudo. Porém, pode-se obter uma maior acuidade na detecção se cada célula da grade acumuladora for representada por um *pixel* no espaço de parâmetros como no SHT de ⁴³YUEN et alii (1990).

Fig 5.1 - Grade acumuladora onde os círculos paramétricos são traçados.

Em seguida foi implementada a técnica de *Template Matching*, de modo que a matriz acumuladora resultante fosse idêntica àquela da Transformada de Hough, para facilitar a comparação entre estes métodos.

5.2 - Utilizando a Transformada de Hough

O algoritmo de Bresenham foi utilizado como base para implementar a Transformada de Hough. Este algoritmo desenha um círculo a partir das coordenadas do centro e do valor do raio [apêndice 4]. A grande vantagem do algoritmo de Bresenham é o fato de usar apenas valores inteiros na montagem do contorno do círculo. Isso aumenta muito a velocidade de processamento, pois o número de instruções utilizadas para operar um número inteiro é menor que o número de instruções utilizadas para operar um número real.

Outra peculiaridade que faz o algoritmo de Bresenham obter melhor resultado do que o uso da equação da circunferência é o fato de não perder a conectividade entre seus pontos, que acontece quando a equação da circunferência é usada.

O algoritmo de Bresenham também tem a vantagem de calcular oito posições do círculo "ao mesmo tempo" a cada incremento do laço onde estão sendo determinadas as posições dos pontos do círculo no espaço de parâmetros, pois este algoritmo, a cada interação, calcula uma posição do arco que varia de 45° a 90° . As sete posições restantes encontradas para formar o círculo são calculadas por simetria dos pontos.

As bordas contidas na imagem podem ser obtidas a partir de Sobel, OU-exclusivo ou mesmo usando uma combinação destas com a Gaussiana. Pode-se obter também a média e a binarização da imagem, antes ou depois de usar um desses detectores.

Após a binarização, as bordas aparecem destacadas em nível zero, enquanto que o restante da imagem aparece em nível um. Ela é rastreada *pixel a pixel*, da esquerda para a direita, de cima para baixo, levando em consideração os zeros acrescentados nos arquivos BMP, quando estes não têm largura múltipla de quatro. Estes zeros acrescentados não são visíveis na imagem mostrada, mas estão presentes

no arquivo BMP, e precisam ser desprezados. Isto corresponde a rastrear o arquivo MBID do último ao primeiro endereço não *header* do arquivo de imagem com extensão BMP.

Quando um ponto de borda na imagem (nível zero) é encontrado, aplica-se neste ponto (x,y) o algoritmo de Bresenham para círculos, tomando como valor de raio, o primeiro valor dentro da faixa escolhida [apêndice 1 seção 1.12].

Os valores de x' , y' e o valor do raio r no ponto são passados como argumento para a função em C que constrói o Arranjo Acumulador com base em Bresenham. A partir desses valores, o algoritmo de Bresenham calcula na mesma interação as oito posições no espaço de parâmetros. Aqui, cada posição no espaço de parâmetros é tridimensional, pois deverá ser indexada pelo valor x' , y' e o valor do raio.

Para realizar tal procedimento em C, é necessária a alocação na memória de um ponteiro acumulador tridimensional, com capacidade de acumular números inteiros. A dimensão da memória alocada é igual ao tamanho da imagem, em *pixels*, vezes o tamanho da faixa de raios que se pretende analisar (também em *pixels*). Este cálculo fornecerá o número de inteiros (2 bytes) que precisa ser alocado.

Porém, para assegurar que as posições de memória alocadas não contenham nenhuma informação anterior e dessa maneira obter valores falsos no acumulador é necessário preencher todas as posições alocadas pelo ponteiro com o valor 0(zero).

As posições encontradas pelo algoritmo de Bresenham fazem parte de um círculo de raio r no espaço de parâmetros, com centro na posição (x,y) de borda encontrada na imagem que está sendo analisada, e devem ser incrementadas de 1(um).

Antes de passar para outra posição na imagem, o algoritmo de Bresenham é repetido para o próximo valor de raio até que a faixa de raios escolhida seja toda rastreada.

No **Scanc**, o raio pode ser escolhido numa faixa máxima de 25 (vinte e cinco) *pixels*. Com o mínimo de 5 (cinco) até 95 (noventa e cinco) *pixels* de raio conforme ilustrado na figura 5.2. O tamanho da faixa também pode ser variável. Ela pode ser escolhida entre 5 (cinco), 10 (dez), 15 (quinze), 20 (vinte) ou 25 (vinte e cinco) *pixels*.

Na medida em que forem sendo encontrados na imagem outros pontos de borda, o procedimento descrito acima será repetido, e as posições encontradas pelo algoritmo de Bresenham para todos os raios da faixa serão incrementados de 1(um).

O resultado é um Arranjo Acumulador formado por um espaço tridimensional, onde serão percebidas altas contagens nos pontos onde o centro de uma determinada forma circular é mais provável. É importante notar que a tridimensionalidade do acumulador permite que este defina completamente o círculo, pois todos os seus parâmetros (x , y e o seu raio) podem ser diretamente indexados.

Fig. 5.2 - Escolha da faixa de raios a ser pesquisada.

5.3 - Utilizando o método de *Template Matching*

O método de *Template Matching* consiste na translação de um *template* sobre a imagem, devendo ele conter a forma que se quer detectar a fim de realizar uma comparação com a imagem.

Quando houver um número suficiente de coincidências entre os pontos da imagem e os do *template*, existe uma forte evidência de que a imagem contém, naquela posição, a forma procurada.

Mais uma vez o Algoritmo de Bresenham é tomado como base, sendo usado na construção dos *templates* circulares, um para cada valor de raio, que tem no **Scanc** seleção idêntica àquela da Transformada de Hough [apêndice1 seção 1.13].

A translação é feita percorrendo o ponto central do *template* por toda a imagem e a cada mudança de sua posição, todos os raios do intervalo são pesquisados.

É feita então a comparação anteriormente descrita, quando os pontos coincidem e o centro do *template* circular é incrementado.

Assim como na Transformada de Hough, aqui também haverá a construção de uma matriz acumuladora de três dimensões indexadas pela posição (x,y) e o valor do raio *r*.

5.4 - A visualização do acumulador

A fim de obter uma maneira de analisar visualmente o acumulador foi implementado no **Scanc** uma função que o mostra na tela do monitor. É preciso então renovar o conteúdo do ponteiro de imagem - o *lpMbid* - com o conteúdo do acumulador.

Como o acumulador é tridimensional, pois cada valor de raio corresponde a uma malha acumuladora, e o ponteiro *lpMbid* permite mostrar o acumulador em apenas duas dimensões, as várias malhas acumuladoras são sobrepostas e cada ponto de imagem passado ao ponteiro *lpMbid* que conterà apenas a maior contagem entre os acumuladores para aquela posição.

Após o ponteiro *lpMbid* ser reescrito, é passado ao procedimento de janela uma mensagem de renovação de tela e o acumulador será representado em tons de cinza numa imagem visualizada na tela do monitor como ilustrado na figura 5.3. É possível salvar esta imagem da sobreposição dos acumuladores como um arquivo MBID normal, assim como também pode-se salvar em arquivo esta sobreposição como uma matriz de valores em código ASCII, e usá-lo para visualizar tridimensionalmente a sobreposição do acumulador, através do Matlab.

Fig. 5.3 - Visualização da sobreposição do acumulador.

É importante salientar que tanto a Transformada de Hough quanto o método de *Template Matching* implementado utilizam todos os parâmetros das formas circulares. Portanto a robustez em ambos os casos é mantida.

Também deve-se observar que a diferença básica entre os dois métodos é que no método de *Template Matching* implementado todos os pontos da imagem são analisados enquanto que na aplicação da Transformada de Hough a análise fica restrita apenas aos pontos de borda da imagem.

6 - Os filtros para o acumulador

6.1 - Introdução

Após a leitura de um arquivo de imagem MBID, detecção de suas bordas, e da construção do acumulador através da Transformada de Hough ou da técnica de *Template Matching* é necessário retirar do acumulador as informações necessárias à localização da ou das possíveis formas circulares contidas na imagem, se houverem. O acumulador, apenas mostra evidências de formas circulares mas não decide se um determinado pico local é ou não é uma forma circular na imagem.

No **Scanc** essa decisão é realizada através de um processo de filtragem do acumulador, onde os parâmetros das possíveis formas circulares na imagem devem ser obtidos através de um critério que forneça resultados considerados satisfatórios. Com esse objetivo foram desenvolvidos os filtros Comparativo, Equalizador e Fuzzy, que serão descritos a seguir.

6.2 - O Filtro Comparativo

O Filtro Comparativo do **Scanc** usa como base de comparação o pico mais elevado do acumulador.

O filtro prevê 5(cinco) faixas para aceitação de posições no acumulador como indexantes de formas circulares: a primeira faixa supõe que a posição no acumulador deva ter um mínimo de 75% do valor do maior pico, a segunda 85%, a terceira 90%, a quarta 95% e a quinta um mínimo de 98%. Uma dessas faixas deve ser escolhida pelo usuário durante a detecção como é ilustrada na figura 6.1.

Muitas vezes a faixa escolhida pelo usuário não é a mais adequada. Quando isso acontece o **Scanc** permite que se escolha uma nova faixa, pois o acumulador não é alterado neste processo, e o ponteiro do acumulador continua com todos os valores obtidos pelo detector de formas circulares.

As posições indexantes fornecem os parâmetros das possíveis formas circulares da imagem. Cada parâmetro (a, b, raio) deve ser armazenado

separadamente num vetor e a variação do raio é usada para indexar cada um desses parâmetros.

Fig. 6.1 - Tela de escolha de faixa permissiva para o Filtro Comparativo através do menu.

Um quarto vetor, também indexado pela variação do raio, deve ser criado para acumular o conteúdo das posições indexantes do espaço acumulador. Este procedimento teoricamente deve inibir a detecção de duas formas circulares de mesmo raio numa imagem. Porém, como os detectores de borda geralmente fornecem bordas grossas ou irregulares, e a varredura de raio é *pixel a pixel* o filtro deve funcionar satisfatoriamente mesmo se existirem duas formas circulares de mesmo raio na imagem.

As posições do acumulador, indexadas pelo raio, que não alcançarem uma determinada altura mínima em relação ao pico mais elevado não são escolhidas como parâmetros indexantes das prováveis formas circulares na imagem, mas posições de valor suficiente para estarem dentro da faixa determinada pelo usuário indexam os parâmetros da ou das formas circulares encontradas, que são passados para as funções de visualização [apêndice 1 seção 1.14].

Um valor mínimo para a posição do acumulador tomada como indexante precisa ser estabelecido para que o filtro não encontre falsos positivos - que é a

detecção de uma forma circular inexistente na imagem - numa imagem que não possua nem formas circulares ou mesmo arcos de circunferências. Ao mesmo tempo este valor mínimo não pode ser muito grande pois o programa deve estar preparado para detectar formas circulares ou arcos de circunferência com um raio muito pequeno (5 *pixels*) e um valor mínimo muito grande iria excluir estas formas circulares em qualquer que fosse a imagem.

O valor mínimo aceitável, portanto, deve ser função do raio. No **Scanc** é dado como a metade do raio - que varia na faixa escolhida. Considerando-se em termos de perímetro pode-se dizer que o valor escolhido é aproximadamente 1/12 (um sobre doze avos) do perímetro da forma circular rastreada. Este é um valor bastante tolerante, que pode levar a um número alto de falsos positivos quando a imagem for complexa (conter muitos objetos, ser ruidosa ou mesmo possuir bordas muito grossas). Porém este valor pode ser facilmente trocado, adequando o filtro a situações onde houver a necessidade de se detectar apenas formas circulares ou arcos grandes.

6.3 - O Filtro Equalizador

Como a faixa de valores para os raios, que pode ser pesquisada no programa, é relativamente grande (até vinte e seis *pixels* de raio), a detecção de uma forma circular pequena em relação a uma outra contida na imagem pode ser totalmente perdida, mesmo com razão de comparação de 75% no filtro comparativo.

O Filtro Comparativo tende a privilegiar as formas circulares de maior raio, pois é nelas que vão ser encontradas mais pontos de borda essenciais na formação do acumulador.

O problema aqui é resolvido com a implementação de um Filtro Equalizador, que divide o número de incrementos contidos em cada posição do acumulador pelo valor do perímetro correspondente ao seu raio indexante. Este procedimento iguala os picos das formas circulares com diferentes raios, e por conseguinte altera o conteúdo do acumulador. O resultado de cada posição é magnificado multiplicando-o por 255(duzentos e cinquenta e cinco), para que o novo acumulador obtido possua valores inteiros assim como o acumulador original.

Depois de equalizado por este filtro o acumulador deve sofrer uma nova filtragem comparativa, que terá agora condições de detectar várias formas circulares com raios absolutamente discrepantes, pois os picos no acumulador referentes a essas formas circulares estão agora no mesmo nível de amplitude, dependendo apenas do número de incrementos relativos ao seu próprio raio.

6.4 - O Filtro Fuzzy

Além das formas circulares com raios muito adversos outro problema que pode aparecer é a multidetecção para a mesma forma circular, que consiste em detectá-la mais de uma vez. Isto acontece devido principalmente à largura das bordas obtidas. Bordas grossas geram valores nas vizinhanças dos picos locais quase tão altos quanto eles, como ilustrado nas figuras 6.2a e 6.2b, e muitas vezes o filtro comparativo não é suficiente para isolar o pico de interesse desses outros. A solução nesse caso é melhorar as bordas obtidas tornando-as mais finas. Uma forma, não implementada neste trabalho, de se conseguir isso é aplicando um algoritmo de *thinning* como o de * ZANG & SUEN apud ⁴⁶GONZALES & WINTZ, 1987.

Fig. 6.2 - a) Forma circular com bordas grossas; b) forma do acumulador correspondente numa visão bidimensional.

* ZANG, T.Y.; SUEN, C.Y. A Fast Parallel Algorithm for Thining Digital Patterns, Comm. ACM, v.27, n.3,p.236-239 apud GONZALES, Rafael C.; WINTZ, Paul Digital Image Processing 2.ed., Addison-Wesley, 1987.

Outro problema que pode ocorrer são as rotas alternativas entre as bordas (de uma mesma forma circular) que o programa encontra quando na formação do acumulador gerando picos muito próximos um do outro. Essas rotas geralmente são causadas por irregularidades nas bordas das próprias formas circulares da imagem. Um exemplo desta situação está ilustrado na figura 6.3a e 6.3b.

Fig. 6.3 - a) Forma circular com rotas alternativas; b) forma do acumulador correspondente numa visão bidimensional.

Na tentativa de resolver este problema implementou-se o Filtro Fuzzy proposto por ¹⁴HAN et alii (1994) referido no Capítulo 2. Para isso foi desenvolvido um programa [apêndice 6] que calcula os valores de um *template* circular (5x5) a partir da equação dada abaixo 6.1.

$$g(r) = k \exp\left(-r^2/R^2\right) \quad (6.1)$$

onde R é a metade da dimensão do *template* circular e r varia de 0(zero) a R.

A forma gráfica desta função está ilustrada na figura 6.4, onde k é igual a 255(duzentos e cinquenta e cinco) para que depois de normalizada essa função tenha valor máximo igual a 1(um) e R é igual a 2(dois) já que o *template* a ser convoluído é de dimensão 5x5, e será construído simetricamente a partir dos três valores obtidos pela função.

Os valores obtidos por essa função devem ser normalizados e formam o *template* circular ilustrado na figura 6.5.

Cada posição do acumulador, que também deve ser normalizado é dividida por 255(duzentos e cinquenta e cinco) e por uma razão empírica (de 0,60) usada para diminuir o nível da normalização.

Fig. 6.4 - Função g(r) .

0,367918	0,367918	0,367918	0,367918	0,367918
0,367918	0,778821	0,778821	0,778821	0,367918
0,367918	0,778821	1,000000	0,778821	0,367918
0,367918	0,778821	0,778821	0,778821	0,367918
0,367918	0,367918	0,367918	0,367918	0,367918

Fig. 6.5 - Template circular usado para implementar o filtro Fuzzy.

Depois de normalizado o acumulador é convoluído com o *template*, o resultado corresponde à área instantânea entre o *template* e o acumulador, modificando assim o acumulador. Um exemplo desta operação está ilustrado na figuras 6.6a e 6.6b.

Após o acumulador ser modificado por esse filtro deve-se encontrar os parâmetros da provável forma circular através do filtro comparativo pois só ele é capaz de decidir se uma região de pico local no acumulador - agora alterado pelo Filtro Fuzzy - é ou não indexante de uma forma circular na imagem. Além disso o Filtro Fuzzy pode ou não ser aplicado em conjunto com o Filtro Equalizador dependendo da necessidade do usuário.

Fig. 6.6 - a) Forma do acumulador de uma forma circular com rota alternativa prestes a ser convoluido com o template, b) Forma do acumulador modificado após a convolução.

6.5 - A visualização das formas circulares encontradas

Após decidir quais são as posições do acumulador que indexam as prováveis formas circulares na imagem, esses valores indexantes são passados a uma função de visualização dessas formas circulares. Essa função então, deverá ser acessada pelo menu do programa como ilustrado na figura 6.7 e usará o algoritmo de Bresenham para círculos a fim de desenhar no ponteiro lpMbid ou os círculos referentes às formas circulares encontradas, e depois requisitar a atualização de janela como está ilustrado na figura 6.8. Uma peculiaridade desse procedimento é que tanto a imagem original quanto a imagem de borda podem ser usadas para mostrar os círculos referentes às formas circulares encontradas.

6.6 - A visualização dos dados das prováveis formas circulares

O **Scanc** contém uma função que permite visualizar os parâmetros do centro (a , b), e o raio da forma circular, encontrados através dos detectores de círculos além do

Fig. 6.7 - Requisição por parte do usuário da visualização da forma circular encontrada.

número original de incrementos no acumulador para cada possível forma circular encontrada. Ao contrário da visualização da imagem da sobreposição do acumulador a qual pode ser modificada na aplicação do Filtro Equalizador ou do Filtro Fuzzy, a informação do número de incrementos aqui não é modificada e mostra exatamente o número de pontos de borda que contribuíram para a detecção. Um exemplo dessa visualização é ilustrado na figura 6.9.

Além de ser visualizada esta imagem pode ser gravada tanto como um arquivo de imagem MBID quanto como um arquivo de caracteres em código ASCII, o qual pode ser exportável para o Matlab, a fim de obter o gráfico tridimensional da sobreposição do acumulador.

Portanto o Filtro Comparativo deve ser aplicado todas as vezes que o Arranjo Acumulador for criado ou alterado pelo Filtro Equalizador ou pelo Filtro Fuzzy.

Fig. 6.8 - Representação da forma circular encontrada através do desenho de um círculo.

Fig. 6.9 - Exemplo da visualização dos dados de uma forma circular encontrada.

7 - Apresentação dos resultados

7.1- Resultados obtidos na representação das imagens contidas nos arquivos MBID

Um Scanner Hewlett Packard ScanJet 4c foi usado para digitalizar a imagem da figura 7.1 com dimensões 320x200x256. Um aplicativo de tratamento de imagem, mais especificamente o Adobe Photoshop 3.0 foi usado para alterar as dimensões dessa imagem para 320x200x16. A visualização destas duas imagens, de 256 e de 16 níveis de cinza através do **Scanc**, estão ilustradas nas figuras 7.1 e 7.2 respectivamente.

Fig. 7.1 - Exemplo da visualização através do Scanc de uma imagem de 256 cores.

Pode-se perceber que a qualidade da visualização na imagem de 16 níveis é inferior à qualidade da visualização da imagem de 256 níveis. Apesar do **Scanc** “aceitar” os dois tipos de imagem, todos os resultados neste trabalho serão obtidos para imagens de 256 cores ou níveis de cinza.

7.2 - Mudança da *palette* para tons de cinza

A imagem da figura 7.3 bem como as demais no restante deste capítulo foram adquiridas através de uma câmera CCD Hitachi KP-110 preto e branco e digitalizada numa *High Resolution Frame Grabber* DT-IRIS 2851.

Fig. 7.2 - Exemplo da visualização através do Scanc de uma imagem de 16 cores.

A *palette* da imagem que é aplicada no **Scanc** nem sempre está em tons de cinza. A opção de transformação da *palette* para tons de cinza deve ser utilizada, para garantir que a imagem analisada contenha apenas tons de cinza. Para o exemplo em questão obteve-se como resultado da mudança de *palette* a imagem representada na figura 7.4.

7.3 - Aplicação dos métodos de suavização na imagem original

Aplicou-se a suavização através da média na imagem da figura 7.5 cuja *palette* foi trocada para tons de cinza e obteve-se como resultado a imagem da figura 7.6.

Nota-se que o contraste da imagem foi reduzido. Isto acontece porque o valor de cada *pixel* da imagem passa a ser uma média entre seus vizinhos e as diferenças entre eles é diminuída.

Fig. 7.3 - Imagem com a palette original.

Fig. 7.4 - Imagem com palette trocada para tons de cinza.

Outra forma de suavizar a imagem é através da aplicação de *templates* formados a partir da função Gaussiana. Isso foi feito na imagem da figura 7.5 e obteve-se a imagem da figura 7.7. Pode-se notar um alargamento das figuras na imagem que acontece porque as mudanças de níveis são atenuadas nesse processo devido ao resultado da convolução entre a função Gaussiana e a imagem.

7.4- Determinação das bordas na imagem

Utilizou-se a imagem da figura 7.5 para realizar a detecção das bordas através de Sobel, cuja imagem de gradientes resultante está ilustrada na figura 7.8 e a imagem binarizada com *threshold* igual a 100(cem) está ilustrada na figura 7.9.

Fig. 7.5 - Imagem original com palette trocada para tons de cinza.

Fig. 7.6 - Imagem suavizada pela média.

A mesma imagem foi usada para realizar esta separação através do operador OU-exclusivo. O resultado desta operação é ilustrado na figura 7.10, que é uma

imagem pouco aproveitável para realizar a detecção de formas circulares, pois o operador detectou contornos extras àqueles que realmente existem na imagem.

Fig. 7.7 - Imagem suavizada através de uma função Gaussiana.

Fig. 7.8 - Imagem obtida pela aplicação dos templates de Sobel.

Uma maneira eficaz de resolver este problema é suavizar essa imagem de borda, retirando assim as bordas consideradas como ruído. Foi aplicado então uma Gaussiana com desvio padrão igual a 1(um) obtendo-se a imagem ilustrada na figura 7.11.

Fig. 7.9 - Imagem binarizada depois de aplicado os templates de Sobel.

Fig. 7.10 - Imagem já binarizada obtida pela aplicação do operador OU-exclusivo.

Fig. 7.11 - Imagem de borda - OU-exclusivo - suavizada com uma Gaussiana.

Resultado semelhante acontece quando a suavização é feita através da média, porém tem a desvantagem de necessitar de uma binarização, com valor de *threshold* bem ajustado, e uma inversão na imagem com outra binarização após a suavização como pode ser observado nas figuras 7.12, 7.13 e 7.14.

Fig. 7.12 - Imagem de borda - OU-exclusivo - suavizada com a média.

Fig. 7.13 - Imagem de borda suavizada com a média após a binarização com threshold = 28.

Fig. 7.14 - Imagem de borda suavizada com a média binarizada e invertida.

Outra maneira de obter as bordas de uma imagem é aplicar uma Gaussiana antes de aplicar o detetor de bordas. As figuras 7.15 e 7.16 mostram o resultado de uma operação desse tipo, utilizando respectivamente o operador Sobel e o operador OU-exclusivo. A imagem suavizada é a apresentada na figura 7.7.

Fig. 7.15 - Imagem de borda através de Sobel obtida a partir de uma imagem suavizada pela Gaussiana.

O problema aqui é a perda de informação, já que, além de parte da imagem ter sido perdida com a suavização, qualquer forma circular encontrada terá seu valor de raio alterado em relação a imagem original pois a imagem dos objetos foi dilatada.

Uma imagem, representada na figura 7.17, com objetos encostados entre si foi usada para realizar a detecção de suas bordas. O resultado é ilustrado na figura 7.18

onde pôde-se notar falhas na borda nas posições onde peças de contrastes aproximadamente iguais se tocaram.

O detector de bordas a ser usado depende das características do processo, do tipo de objeto a ser inspecionado e o objetivo da detecção. Se, por exemplo, apenas a detecção da existência da forma circular for importante, um filtro utilizando a Gaussiana poderia ser usado, seguido de um detector de bordas OU-exclusivo, este processo fornece bordas finas, que evita dubiedades na detecção. Porém, se o valor do raio da forma circular a ser detectada for relevante, esta maneira de detectar bordas é ineficaz, pois modifica o valor do raio.

Fig. 7.16 - Imagem de borda através do OU-exclusivo obtida a partir de uma imagem suavizada pela Gaussiana.

Fig. 7.17 - Imagem original contendo objetos encostados entre si.

Fig. 7.18 - Imagem contendo as falhas na borda devido a falta de separação entre as peças.

Através dos exemplos acima verifica-se que além do detector pode-se usar junto com este uma combinação de ferramentas gráficas mudando assim as características da borda detectada. Portanto a escolha do detector ou da combinação de ferramentas com o detector deve ser feita pelo usuário.

Neste trabalho a escolha de qual detector a ser usado é irrelevante por se tratar de um estudo do método. Comparações entre imagens de bordas finas obtidas a partir do detector OU-exclusivo, precedida da aplicação do filtro que utiliza a Gaussiana, e imagens de bordas grossas obtidas a partir da aplicação do detector Sobel, também precedida do mesmo filtro, serão feitas neste trabalho para obtenção de diferenças de tempo. Daí a importância de se implementar diferentes filtros para se obter diferentes imagens de bordas.

7.5 - A detecção de formas circulares

7.5.1 - Utilizando a Transformada de Hough

Utilizando a Transformada de Hough para círculos a fim de detectar formas circulares com raio na faixa de 30(trinta) a 35(trinta e cinco) *pixels* na imagem de bordas contida na figura 7.9 obteve-se como resultado um espaço acumulador cuja sobreposição de seus picos em cada ponto da imagem está ilustrado no gráfico da

figura 7.19. Este gráfico foi obtido no Matlab através da importação da matriz formada pela sobreposição do acumulador exportada pelo **Scanc**.

Pode-se notar neste gráfico que há evidência de duas formas circulares dentro da faixa de raios pesquisada. Então o **Scanc** precisa utilizar seu filtro comparativo que decide onde estão e quais são as prováveis formas circulares contidas na imagem através de um critério de comparação entre os maiores valores acumulados. Um de cada raio.

Fig. 7.19 - Gráfico da sobreposição do acumulador obtido pela aplicação da Transformada de Hough para detectar as prováveis formas circulares numa imagem.

7.5.1.1 - Aplicando apenas o Filtro Comparativo

Aplicou-se no acumulador evidenciado pelas sobreposições ilustradas na figura 7.19 o filtro comparativo na razão de 98% do maior pico na imagem. O

resultado pode ser visualizado na imagem de borda da figura 7.20 ou mesmo na imagem original da figura 7.21.

O filtro encontrou uma forma circular apenas na posição (201, 65) com raio igual a 33(trinta e três) *pixels*, e 120(cento e vinte) pontos de borda contados no acumulador contribuíram para encontrar esta forma circular. Este resultado nos diz que um dos picos é pelo menos 2% mais alto que o outro. Dessa maneira a outra forma circular presente na imagem não pôde ser detectada com o filtro nessa razão.

Fig. 7.20 - Resultado da detecção pelo filtro comparativo visualizado na imagem de bordas.

Fig. 7.21 - Resultado da detecção pelo filtro comparativo visualizado na imagem original.

A outra forma circular é detectada quando o Filtro Comparativo é programado para uma razão de 85%. Foram então detectadas quatro formas

circulares como ilustrado na figura 7.22 : a primeira na posição (234,174), raio igual a 31(trinta e um) e 103(cento e três) pontos de borda no acumulador; a segunda na posição (230,174), raio 32(trinta e dois) e 107(cento e sete) pontos no acumulador; a terceira forma circular encontrada apresentou parâmetros, (201,65), raio 33, acumulador 120 e a quarta, (202,66), raio 34 e acumulador 106.

Fig. 7.22 - Resultado da detecção pelo filtro na razão de 85%.

Através dos valores acumulados percebe-se que a pequena diferença que existe entre a altura de três dos quatro pontos detectado faz com que o filtro encontre nessa razão quatro formas circulares na imagem, nessa faixa de raios, e não duas.

A outra forma circular presente na imagem da figura 7.5 pode ser detectada se for usada uma faixa para raios de maiores valores na detecção pela Transformada de Hough. Aplicando uma faixa que vai de 65(sessenta e cinco) a 70(setenta) *pixels* de raio obteve-se a sobreposição do acumulador apresentado na figura 7.23.

É aplicado então o filtro comparativo na razão de 98% detectando as formas circulares indicadas na figura 7.24 cujos parâmetros são: (198,64), raio 65(sessenta e cinco), acumulador 141; (196,65), raio 66, acumulador 142; (199,64), raio 67, acumulador 141.

Através de um programa que calcula o número de pontos traçados para um círculo de determinado raio [apêndice 7], pode-se verificar que para os raios 31(trinta e um), 32(trinta e dois), 33(trinta e três) e 34(trinta e quatro) o algoritmo utiliza 184(cento e oitenta e quatro), 192(cento e noventa e dois), 200(duzentos), e novamente 200(duzentos) pontos respectivamente para desenhar os círculos.

Estes dados dizem que para o raio igual a 31(trinta e um) o **Scanc** encontrou 103(cento e três) pontos de borda na imagem que contribuíram para detecção da forma circular, enquanto que um círculo de Bresenham com esse raio possui 184(cento e oitenta e quatro) pontos. Ou seja, na verdade o programa detectou para

Fig. 7.23 - Sobreposição do acumulador para raios na faixa 65 - 70.

Fig. 7.24 - Formas circulares detectadas pelo Filtro Comparativo.

esse raio apenas 56% de uma forma circular perfeita (partindo do princípio que o algoritmo Bresenham desenha um círculo perfeito). O mesmo acontece quando analisamos os outros raios: para raio igual a 32(trinta e dois) apenas 55,7% da forma circular foi detectada; para o raio igual a 33(trinta e três) 60%, e para o raio igual a 34(trinta e quatro) 53%.

Através desses números e da imagem contida na figura 7.19 pode-se perceber que a contribuição de pontos da forma circular cujos parâmetros são (201,65), raio 33 foi maior que a contribuição de pontos das outras formas circulares encontradas na mesma faixa quando a seletividade do Filtro Comparativo foi rebaixada para 85%, e que este filtro detectou apenas essa forma quando a seletividade foi escolhida a 98% ou mesmo até 90%.

Na figura 7.22 percebe-se que as formas circulares encontradas para o corpo (supostamente circular) inferior estão deslocadas, uma para direita e outra para esquerda. Conclui-se então que o corpo é levemente elíptico, devido a iluminação inadequada no momento da aquisição da imagem através da câmera CCD, e não perfeitamente circular como podia parecer na imagem da figura 7.5, fornecendo um número pequeno de pontos de contribuição. Como o filtro usado compara os picos do acumulador, se houver na imagem uma verdadeira forma circular, esta inibirá a detecção de outras formas, mesmo que sejam apenas levemente elípticas como é o caso desta imagem.

Para a outra faixa de raios pesquisada a forma circular foi detectada três vezes com percentagens de: 37,5% para raio igual a 65(sessenta e cinco), 37% para raio igual a 66(sessenta e seis) e 36% para raio igual a 67(sessenta e sete); de pontos de borda contribuintes pois os círculos de Bresenham foram formados com 376, 384, 392 pontos respectivamente. Aqui, a diferença entre as contagens do acumulador foi muito pequena e o Filtro Comparativo não conseguiu separá-las mesmo na razão de 98%. Os valores encontrados na detecção das formas circulares depois de aplicado o Filtro Comparativo estão discriminados nas tabelas 7.1 e 7.2.

7.5.1.2 - Utilizando o Filtro Fuzzy antes do Filtro Comparativo

É aplicado o Filtro Fuzzy no acumulador cuja sobreposição está ilustrada na figura 7.19. O resultado é a alteração do espaço acumulador. A figura 7.25 ilustra o gráfico da sobreposição do acumulador resultante da aplicação desse filtro.

Formas Circulares Menores - Faixa de Raios (<i>Pixels</i>): 30 - 35				
Centro (x,y)	Raio	Acumulado	Bresenham	Porcentagem
(234,174)	31	103	184	56%
(230,174)	32	107	192	55,7%
(201,65)	33	120	200	60%
(202,66)	34	106	200	53%

Tabela 7.1 - Dados correspondentes às formas circulares com raios entre 30-35 pixels, encontradas através da Transformada de Hough quando aplicado o Filtro Comparativo na razão de 85%.

Formas Circulares Maiores - Faixa de Raios (<i>Pixels</i>): 65 - 70				
Centro (x,y)	Raio	Acumulado	Bresenham	Porcentagem
(198,64)	65	141	376	37,5%
(196,65)	66	142	384	37%
(199,64)	67	141	392	36%

Tabela 7.2 - Dados correspondentes às formas circulares com raios entre 65-70 pixels encontradas através da Transformada de Hough quando aplicado o Filtro Comparativo na razão de 98%.

É aplicado nesse novo acumulador o Filtro Comparativo na razão de 98%, resultando na detecção de uma forma circular com parâmetros (201,65), raio 33, cujo valor acumulado original é 120(cento e vinte) pontos de contribuição, que pode ser visualizada pela imagem da figura 7.26.

Novamente o Filtro Comparativo foi aplicado na razão de 90% e três formas circulares visualizadas na figura 7.27 foram detectadas com parâmetros: (231,174),raio 32, acumulador 107, (201,65), raio 33, acumulador 120, (201,66), raio

Fig. 7.25 - Gráfico obtido a partir da sobreposição do acumulador alterado pelo Filtro Fuzzy.

Fig. 7.26 - Forma circular encontrada pelo Filtro Comparativo a 98% depois de aplicado o Filtro Fuzzy no acumulador.

34, acumulador 106. Nota-se que a perda de seletividade do Filtro Comparativo para haver a detecção da outra forma não precisou ser tão grande quanto quando ele foi

aplicado no acumulador original, além do mais o Filtro Fuzzy selecionou apenas uma forma circular para o corpo inferior direito da imagem, pois este filtro escolheu uma rota alternativa entre os pontos de borda das duas prováveis formas circulares encontrada para o corpo inferior direito encontrados antes da aplicação deste filtro aumentando o pico do acumulador na posição incrementada a partir dessa rota.

Porém o mesmo não aconteceu com as formas circulares detectadas nessa faixa para o corpo posicionado na parte superior da imagem, pois a duplicidade não aconteceu pelo fato do filtro ter encontrado rotas alternativas para as formas circulares mas pelo fato das bordas serem grossas o suficiente para gerar a duplicidade.

Fig. 7.27 - Formas circulares encontradas pelo Filtro Comparativo a 90% depois de aplicado o Filtro Fuzzy no acumulador.

Também foi aplicado o Filtro Fuzzy no acumulador original criado quando na aplicação da Transformada de Hough para a faixa de raios que vai de 65 a 70 *pixels* resultando no espaço acumulador representado pelas sobreposições da figura 7.28. E depois foi aplicado também o Filtro Comparativo na razão de 98% detectando as duas formas circulares mostradas na imagem da figura 7.29 cujos parâmetros são: (198,64), raio 66 e valor acumulado original de 142 pontos de borda e a outra forma circular com centro em (198,64), raio 67 e acumulador 141. Na figura 7.30 pode-se visualizar as duas formas circulares encontradas na imagem de bordas.

Fig. 7.28 - Sobreposição do acumulador modificado pelo Filtro Fuzzy.

Fig. 7.29 - Formas circulares encontradas pelo Filtro Comparativo na razão de 98% a partir do acumulador alterado pelo Filtro Fuzzy desenhadas na imagem original.

É também ilustrado, na figura 7.31, o posicionamento das formas circulares, na imagem de bordas, encontradas quando na aplicação do Filtro Comparativo no acumulador original. Através dessa imagem pode-se notar que há uma forma circular

detectada a partir de uma rota alternativa entre as bordas da imagem. O Filtro Fuzzy portanto, escolhe uma nova rota entre as bordas da imagem incrementando o centro encontrado para essa rota. Porém, novamente houve uma duplicidade na detecção causada pela largura da borda.

Fig. 7.30 - Formas circulares encontradas pelo Filtro Comparativo na razão de 98% a partir do acumulador alterado pelo Filtro Fuzzy desenhadas na imagem de borda.

Fig. 7.31 - Formas circulares encontradas pelo Filtro Comparativo na razão de 98% a partir do acumulador original desenhadas na imagem de borda.

7.5.1.3 - Aplicando o Filtro Equalizador antes do Comparativo

A imagem mamográfica da figura 7.32 foi submetida à análise através do **Scanc**. As bordas foram obtidas através de Sobel e da binarização da imagem. O resultando está ilustrado na figura 7.33.

Fig. 7.32 - Imagem original de uma mamografia.

Foi aplicada então a detecção através da Transformada de Hough para faixa de raios que vai de 25 a 50 *pixels* resultando no acumulador representado pela sobreposição de seus picos ilustrada na figura 7.34. A aplicação de um Filtro Comparativo na razão de 98% resultou na detecção de uma forma circular com parâmetros (464,92), raio 48, acumulador 169 que pode ser visualizada através da figura 7.35.

O Filtro Comparativo também foi aplicado a uma razão de 95%, detectando agora duas formas circulares cujos parâmetros são: (464,91), raio 47, acumulador 165 e (464,92), raio 48, acumulador 169. A figura 7.36 ilustra esta situação.

Fig. 7.33 - Imagem de bordas da mamografia.

Fig. 7.34 - Sobreposição do acumulador obtida pela aplicação da Transformada de Hough na imagem de bordas da mamografia.

Fig. 7.35 - Visualização na imagem original da forma circular encontrada quando usado o Filtro Comparativo na razão de 98%.

Fig. 7.36 - Visualização na imagem original das formas circulares encontradas quando aplicado o Filtro Comparativo na razão de 95%.

Foi aplicado o Filtro Comparativo na razão de 75% que é a menor permitida pelo programa, as formas circulares escolhidas pelo filtro estão ilustradas na figura 7.37.

Percebe-se que o pico referente ao corpo menor está abaixo de 75% do pico referente ao corpo maior. Esta grande diferença é causada principalmente pelo fato do corpo maior possuir mais pontos que o corpo menor sem necessariamente ser mais circular que o outro.

Fig. 7.37 - Visualização na imagem original das formas circulares encontradas quando aplicado o Filtro Comparativo na razão de 75%.

Nesta situação o Filtro Comparativo fornece um resultado indesejado já que ele provoca a perda da detecção de uma forma circular menor quando um corpo circular maior estiver presente na mesma faixa de raios.

Foi aplicado o Filtro Equalizador no acumulador original obtendo um novo espaço acumulador representado na figura 7.38 pelas sobreposições de seus picos.

Filtro Comparativo é novamente aplicado na razão de 98% selecionando assim uma forma circular apenas cujos parâmetros são: (49,171), raio 25, com 97(noventa e

sete) pontos de borda contribuindo no acumulador para sua detecção. A visualização dessa detecção pode ser feita na figura 7.39.

A seletividade do Filtro Comparativo é reduzida para razão de 90% a fim de permitir a detecção da outra forma circular presente na imagem (isso não aconteceu na razão de 95%). Três formas circulares, que podem ser visualizadas na

Fig. 7.38 - Sobreposição do acumulador modificado pelo Filtro Equalizador.

figura 7.40, foram selecionadas pelo filtro nessa razão, seus parâmetros são: (49,171), raio 25, acumulador 97; (464,91), raio 47, acumulador 165, (464,92), raio 48, acumulador 169.

Estes resultados nos dizem que a quantidade de pontos de borda que contribuíram para a detecção da forma circular foi proporcionalmente (em relação ao valor do raio) maior no corpo menor.

Fig. 7.39 - Formas circulares detectadas pelo Filtro Comparativo, na razão de 98%, aplicado no acumulador modificado pelo Filtro Equalizador.

Fig. 7.40 - Formas circulares detectadas pelo Filtro Comparativo, na razão de 90%, aplicado no acumulador modificado pelo Filtro Equalizador.

7.5.2 - Utilizando o método de *Template Matching*

Utilizando o método do *Template Matching* a fim de detectar formas circulares com raio na faixa de 30(trinta) a 35(trinta e cinco) *pixels* na imagem de bordas contida na figura 7.9 obteve-se como resultado um espaço acumulador cuja sobreposição de seus picos em cada ponto da imagem está ilustrado no gráfico da figura 7.41 que é idêntica ao gráfico na da figura 7.19 obtida com a aplicação da Transformada de Hough.

Fig. 7.41 - Gráfico da sobreposição do acumulador obtido pela aplicação da técnica de Template Matching para detectar as prováveis formas circulares na imagem.

Naturalmente a detecção pelo Filtro Comparativo se dará da mesma forma que no acumulador original obtido a partir da Transformada de Hough, e também pode ser alterado pelos Filtros Fuzzy e Equalizador obtendo-se os mesmos resultados no acumulador.

7.5.3 - Obtendo os tempos de execução

Para medir os tempos de execução dos acumuladores, formados pela Transformada de Hough pela técnica do *Template Matching*, bem como dos filtros Comparativo, Fuzzy e Equalizador, foram usadas as imagens das figuras 7.15 e 7.16 (com dimensões 320x240 *pixels*) obtidas a partir da aplicação de uma Gaussiana com desvio 0,2 na imagem original com *palette* trocada para tons de cinza e bordas obtidas a partir de Sobel e OU-Exclusivo respectivamente. Estas imagens de bordas têm respectivamente 3.598(três mil, quinhentos e noventa e oito) e 1.598(mil, quinhentos e noventa e oito) pontos de borda. Estes números de pontos são obtidos por um programa contador cujo código fonte está contido no apêndice 8. A figura 7.42 (com dimensões 378x283 *pixels*) que possui os mesmos pontos de borda que a imagem da figura 7.16 também foi usada, pois sendo maior que as outras permite uma comparação de resultados em imagens de diferentes tamanhos. Os resultados referentes às figuras 7.15, 7.16 e 7.42 constam nas tabelas 7.1, 7.2 e 7.3 respectivamente.

Gráficos tridimensionais onde um eixo representa o início das faixas de raios pesquisadas, o outro representa a amplitude da faixa de raios e o terceiro eixo representa o tempo de execução das rotinas do programa estão ilustrados nas figuras cuja numeração vai de 7.43 a 7.58.

Fig. 7.42 - Imagem - com dimensões 378x283 pixels - com bordas obtidas através do OU-exclusivo a partir de uma imagem suavizada pela Gaussiana.

Faixa de raios	Número de raios	TH (seg)	Fuzzy (seg)	Equaliz. (seg)	Comp. (seg)	TM (seg)	Fuzzy (seg)	Equaliz. (seg)	Comp. (seg)
20-25	6	19	57	1	0	183	57	1	0
20-35	16	62	153	5	1	586	158	5	1
20-45	26	126	252	8	2	1116	259	8	2
45-50	6	38	60	1	0	362	59	1	0
45-60	16	111	161	5	1	1058	159	5	1
45-70	26	205	263	8	2	1852	260	8	2
70-75	6	55	60	1	0	525	59	1	0
70-85	16	157	162	5	1	1481	161	5	1
70-95	26	297	264	8	2	2556	260	8	2

Tabela 7.3 - Tempos medidos na detecção de formas circulares numa imagem (320x240x256) de bordas grossas através da TH (Transformada de Hough) e TM (Template Matching) usando os filtros Fuzzy, Equalizador e Comparativo. A medição dos tempos dos filtros é independente.

Faixa de raios	Número de raios	TH (seg)	Fuzzy (seg)	Equaliz. (seg)	Comp. (seg)	TM (seg)	Fuzzy (seg)	Equaliz. (seg)	Comp. (seg)
20-25	6	9	57	1	0	174	57	1	0
20-35	16	30	153	5	1	556	153	5	1
20-45	26	70	252	8	2	1051	252	8	2
45-50	6	17	60	1	0	343	60	1	0
45-60	16	53	161	5	1	1007	161	5	1
45-70	26	103	263	8	2	1766	263	8	2
70-75	6	25	60	1	0	501	60	1	0
70-85	16	73	162	5	1	1421	162	5	1
70-95	26	130	168	8	2	2433	268	8	2

Tabela 7.4 - Tempos medidos na detecção de formas circulares numa imagem (320x240x256) de bordas finas através da TH (Transformada de Hough) e TM (Template Matching) usando os filtros Fuzzy, Equalizador e Comparativo. A medição dos tempos dos filtros é independente.

Faixa de raios	Número de raios	TH (seg)	Fuzzy (seg)	Equaliz. (seg)	Comp. (seg)	TM (seg)	Fuzzy (seg)	Equaliz. (seg)	Comp. (seg)
20-25	6	10	81	2	1	280	79	2	1
20-35	16	34	212	6	2	767	212	6	2
20-45	26	70	347	11	4	1475	350	11	4
45-50	6	20	81	2	0	474	80	2	0
45-60	16	60	217	6	2	1388	221	6	2
45-70	26	125	354	11	4	2470	352	11	4
70-75	6	29	82	2	1	700	80	2	1
70-85	16	96	221	6	2	1985	218	6	2
70-95	26	144	360	11	4	3406	359	11	4

Tabela 7.5 - Tempos medidos na detecção de formas circulares numa imagem (378x283x256) de bordas finas através da TH (Transformada de Hough) e TM (Template Matching) usando os filtros Fuzzy, Equalizador e Comparativo. A medição dos tempos dos filtros é independente.

Fig. 7.43 - Gráfico referente aos tempos medidos na execução do Arranjo Acumulador usando a Transformada de Hough na imagem (320x240x256) de bordas grossas. Os eixos do gráfico são dadas pelo início da faixa de raios que deverá ser usada para detectar a forma circular, a quantidade de raios a serem pesquisados menos um e o tempo em segundos que o detector consumiu para realizar a detecção.

Fig. 7.44 - Gráfico referente aos tempos medidos na execução do Arranjo Acumulador usando o método do Template Matching na imagem (320x240x256) de bordas grossas.

Fig. 7.45 - Gráfico referente aos tempos medidos na execução do Arranjo Acumulador usando a Transformada de Hough na imagem (320x240x256) de bordas finas.

Fig. 7.46 - Gráfico referente aos tempos medidos na execução do Arranjo Acumulador usando o método do Template Matching na imagem (320x240x256) de bordas finas.

Fig. 7.47 - Gráfico referente aos tempos medidos na execução do Arranjo Acumulador usando a Transformada de Hough na imagem (378x283x256) de bordas finas.

Fig. 7.48 - Gráfico referente aos tempos medidos na execução do Arranjo Acumulador usando o método do Template Matching na imagem (378x282x256) de bordas finas.

Fig. 7.49 - Gráfico referente aos tempos medidos na execução do Filtro Fuzzy após a construção do Arranjo Acumulador pela Transformada de Hough na imagem (320x240x256) de bordas grossas.

Fig. 7.50 - Gráfico referente aos tempos medidos na execução do Filtro Fuzzy após a construção do Arranjo Acumulador pela técnica do Template Matching na imagem (320x240x256) de bordas grossas.

Fig. 7.51 - Gráfico referente aos tempos medidos na execução do Filtro Fuzzy após a construção do Arranjo Acumulador pela Transformada de Hough na imagem (320x240x256) de bordas finas.

Fig. 7.52 - Gráfico referente aos tempos medidos na execução do Filtro Fuzzy após a construção do Arranjo Acumulador pela técnica do Template Matching na imagem (320x240x256) de bordas finas.

Fig. 7.53 - Gráfico referente aos tempos medidos na execução do Filtro Fuzzy após a construção do Arranjo Acumulador pela Transformada de Hough na imagem (378x282x256) de bordas finas.

Fig. 7.54 - Gráfico referente aos tempos medidos na execução do Filtro Fuzzy após a construção do Arranjo Acumulador pela técnica do Template Matching na imagem (378x282x256) de bordas finas.

Fig. 7.55 - Gráfico referente aos tempos medidos na execução do Filtro Equalizador após a construção do Arranjo Acumulador pela Transformada de Hough e Template Matching nas imagens (320x240x256) de bordas grossas e finas.

Fig. 7.56 - Gráfico referente aos tempos medidos na execução do Filtro Equalizador após a construção do Arranjo Acumulador pela Transformada de Hough e Template Matching na imagem (378x282x256) de bordas finas.

Fig. 7.57 - Gráfico referente aos tempos medidos na execução do Filtro Comparativo após a construção do Arranjo Acumulador pela Transformada de Hough e Template Matching nas imagens (320x240x256) de bordas grossas e finas.

Fig. 7.58 - Gráfico referente aos tempos medidos na execução do Filtro Comparativo após a construção do Arranjo Acumulador pela Transformada de Hough e Template Matching na imagem (378x282x256) de bordas finas.

Pode-se perceber através das tabelas 7.1, 7.2 e 7.3 e também pelos gráficos ilustrados nas figuras cujas numerações vão de 7.43 a 7.58, referentes ao tempo de construção do acumulador tanto pela aplicação da Transformada de Hough quanto pelo método do *Template Matching*, que o tempo de processamento cresce tanto com o aumento do número de raios pesquisados quanto com aumento do valor inicial da faixa, ou seja, quanto maior o número de raios que se quer pesquisar ou quanto maiores forem os valores absolutos dos raios pesquisados maior será o tempo gasto na execução do acumulador.

Outro fato que se pôde observar nestas mesmas tabelas e gráficos é que o tempo de execução do acumulador através do método do *Template Matching* foi quase sempre mais de 10(dez) vezes o tempo de execução através da Transformada de Hough. Principalmente quando as bordas na imagem eram finas.

O tempo de execução do acumulador através da Transformada de Hough para a imagem de bordas grossas da figura 7.15 foi consideravelmente maior que o tempo gasto para a imagem de bordas finas da figura 7.16, que por sua vez foi muito próximo do tempo gasto para a imagem de bordas finas, porém de maiores medidas, da figura 7.42, havendo nesse caso um pequeno incremento de tempo quando processada a imagem de maiores dimensões.

Resultado oposto se deu quando usado o método do *Template Matching*, que pouco variou o tempo de processamento entre as imagens de bordas grossas e finas, porém sofreu um considerável aumento no tempo quando foi usada uma imagem de maiores dimensões.

Através dos gráficos cujas figuras têm numerações que vão de 7.49 a 7.54 e novamente através das tabelas, pode-se perceber que o tempo de execução do Filtro Fuzzy foi praticamente o mesmo tanto na imagem de bordas finas quanto na imagem de bordas grossas, quando essas imagens possuíam as mesmas dimensões. Invariante ao fato do acumulador ser construído através da Transformada de Hough ou do método do *Template Matching*. A mudança ocorre quando usada uma imagem de maiores dimensões.

O Filtro Equalizador analisado nos gráficos das figuras 7.55 e 7.56 é muito mais rápido que o Filtro Fuzzy e mantém as mesmas características que este filtro no que diz respeito a relação do tempo de processamento com o tamanho da imagem e com o tipo de método utilizado na formação do acumulador.

Os gráficos representados nas figuras 7.57 e 7.58 ilustram o comportamento do tempo de execução do filtro Comparativo. Percebe-se através destes gráficos que o tempo de execução deste filtro é sempre muito baixo e é independente do fato do acumulador ter sido gerado através da Transformada de Hough ou do método do *Template Matching*. Independe também do número de pontos de borda em imagens de mesmo tamanho e do valor absoluto do raio pesquisado. Porém o tempo de processamento varia em imagens de diferentes dimensões ou para diferentes comprimentos de faixa de raios pesquisados.

Conclusões

Foi desenvolvido um programa capaz de detectar formas circulares em imagens digitais. No programa foram implementados dois métodos distintos de detecção de formas circulares: Transformada de Hough e *Template Matching*. Através de resultados experimentais constatou-se que a performance de ambos os métodos é dependente da borda detectada. Este resultado já era esperado, já que ambos os métodos utilizam os pontos de borda como ponto de partida para análise, embora apresentem características próprias. A Transformada de Hough analisa a imagem ponto a ponto e o *Template Matching* faz a comparação entre formas verificando se as posições dos pontos coincidem. Mas apesar da dependência da borda, ambos os métodos apresentaram uma boa tolerância, tanto a ruídos quanto à deformidades na forma.

Constatou-se também que o tempo de processamento requerido para o método de Hough, foi, de forma geral, menor que o requerido para o método de *Template Matching* utilizado. Isso aconteceu porque o método de Hough analisa apenas os pontos de borda enquanto que o método de *Template Matching* analisa todos os pontos da imagem. Portanto este método é mais adequado para aplicações em tempo real.

Verificou-se que o Filtro Comparativo implementado no programa foi capaz de discriminar as formas circulares da imagem a partir do conteúdo do acumulador gerado na aplicação dos métodos de detecção (Transformada de Hough e *Template Matching*) nas imagens de borda. Porém um melhor resultado poderia ser obtido se fosse aplicado um algoritmo de *Thining* nas bordas obtidas, pois bordas grossas geram detecções extras para a mesma forma circular.

Ao contrário do que alegaram ¹⁴Han et alii, 1994, concluímos que seu algoritmo baseado na lógica Fuzzy não é capaz de abolir as detecções extras de formas circulares que aconteceram por causa das bordas grossas. Por se tratar de uma convolução de um *template* circular com o espaço acumulador, o método de Han não afina os picos que ocorrem neste espaço, mas cria um pico apenas, num ponto médio ponderado nas situações em que havia dois picos muito próximos um do outro de

amplitudes aproximadamente iguais no acumulador, gerados pelos caminhos alternativos das formas circulares na imagem. Portanto o filtro de Han é eficaz quando o detetor encontrou formas circulares extras devido à distorções na forma. Porém a aplicação deste filtro eleva consideravelmente o tempo necessário para realizar a detecção.

O Filtro Comparativo apresentou limitações identificando apenas a maior forma circular quando os raios apresentam valores distantes. Esse problema é resolvido aplicando o Filtro Equalizador.

O Filtro Equalizador apresenta um bom desempenho igualando as amplitudes dos picos no acumulador, referentes às formas circulares.

Portanto a aplicação da Transformada de Hough juntamente com o Filtro Comparativo e o Filtro Equalizador constitui um método eficiente para detectar formas circulares.

Referências Bibliográficas

Periódicos

- (01)-ALBANESI, M.G.; FERRETTI, M., Shape Detection With Limited Memory, Pattern Recognition, v.24, n.12, p.1153-1166, 1991.
- (02)-BALLARD, D.H. Generalizing the Hough Transform to Detect Arbitrary Shapes, Pattern Recognition, v.13, n.2, p.111-122, 1981.
- (03)-BHANDARKAR, Suchendra M. A Fuzzy Probabilistic Model for the Generalized Hough Transform, IEEE Transactions on Systems, Man, and Cybernetics, v.24, n.5, 1994.
- (04)-CANNY, John A Computational Approach to Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, v. PAMI-8, n.6, p.679-698, 1986.
- (05)-CASASENT, David; KRISHNAPURAM, Raghuram. Curved Object Location by Hough Transformation and Inversions, Pattern Recognition, v.20, n.2, p.181-188, 1987.
- (06)-CHAN, R.; SIU, W.C. New Parallel Hough Transform for Circles, IEE Proceedings-E, v.138, n.5, 1991.
- (07)-COSTABILE, M.F.; PIERONI, G.G. Detecting Shape Correspondences by Using the Generalized Hough Transform, IEEE, 1986.
- (08)-DAVIES, E.R. Modelling Peak Shapes Obtained by Hough Transform, v.139, n.1, 1992.

- (09)-DUDA, Richard O.; HART Peter E. Use of the Hough Transformation To Detect Lines and Curves in Pictures, Graphics and Image Processing, v.15, n.1, p.11-15, 1972.
- (10)-FOSTER, Joseph W.; GRIFFIN, Paul M.; MESSIMER, Sherri L.; VILLALOBOS, J. René Automated Visual Inspection: A Tutorial, Computers ind. Engng, v.18, n.4, p.493-504, 1990.
- (11)-GATH, Isak; HOORY Dan Fuzzy Clustering of Elliptic Ring-Shaped Cluster, Pattern Recognition Letters, v.16, p.727-741, 1995.
- (12)-GERIG, Guido; KLEIN, Fernand. Fast Contourn Identification Through Efficient Hough Transform and Simplified Interpretation Strategy, IEEE, 1986.
- (13)-GONZAGA, Adilson; FRANÇA, Celso A. Um sistema de processamento de imagens dirigido a microcomputador compatível com PC, SIBGRAPI V, novembro, 1992.
- (14)-HAN, Joon H.; KÓCZY, László T.; POSTON, Timothy Fuzzy Hough Transform, Pattern Recognition Letters, v.15, p.649-658, 1994.
- (15)-HANAHARA, Keishi; HIYANE, Masao. A Circle-Detection Algorithm Simulating Wave Propagation, Machine Vision and Applications, v.3, p.97-111, 1990.
- (16)-HO, Chun-Ta; CHEN, Ling-Hwei A Fast Ellipse/Circle Detector Using Geometric Symmetry, Pattern Recognition, v.28, n.1, p.117-124, 1995.

- (17)-HORAUD, Radu; SOSSA, Humberto Polyhedral Object Recognition by Indexing, Pattern Recognition, v.28, n.12, p.1855-1870, 1995.
- (18)-HOUGH, P.V.C. Method and Means for Recognizing Complex Patters, U.S. Patent 3.069.654. 18 dec., 1962.
- (19)-HSU, Chia-Chun; HUANG, Jun S. Partitioned Hough Transform for Ellipsoid Detection, Pattern Recognition, v.23, n.2/4, p.275-282, 1990.
- (20)-ILLINGWORTH, J.; KITTLER, J. A Survey of the Hough Transform, Computer Vision, Graphics and Image Processing, v.44, p.87-116, 1988.
- (21)-ILLINGWORTH, J; KITTLER, J. The adaptative Hough Transform, IEEE Trans. PAMI, v.9, n.5, p.690-697,1987.
- (22)-KÄLVIÄINEN, Heikki; HIRVONEN, Petri; XU, Lei; OJA, Erkki Probabilistic and non -probabilistic hough transform: overview and comparisons, Image and Vision Computing, v.13, n.4, 1995.
- (23)-KIMME, Carolyn; BALLARD, Dana; SKLANSKY, Jack. Finding Circles by an Array of Accumulators, Graphics and Image Processing, v.18, n.2, p.120-122, 1975.
- (24)-KUMAR, Senthil; RANGANATHAN, N.; GOLDGOF, Dmitry Parallel Algorithms for Circle Detection in Images, Pattern Recognition, v.27, n.8, p.1019-1028, 1994.
- (25)-LEAVERS, V.F. The Dynamic Generalized Hough Transform: Its Relationships to the Probabilistic Hough Transform and an Application to the Concurrent Detection of Circles and Ellipses, CVGIP: Image Understanding, v.56, n.3, p.381-398, 1992.

- (26)-LEUNG, Mun K.; HUANG, Thomas S. Detecting the Wheel Pattern of a Vehicle Using Stereo Images. *Pattern Recognition*, v.24, n.12, p.1139-1151, 1991.
- (27)-LI, H.; LAVIN, M.A.; LeMASTER, R.J. Fast Hough Transform: a hierarchical approach, *Comput Vision, Graphics & Image Process*, v.36, p.139-161, 1986.
- (28)-MERLIN, Philip M.; FARBER, David J. A Parallel Mechanism for Detecting Curves in Pictures, *IEEE Transactions on Computers*, 1975.
- (29)-MUAMMAR, H.K.; NIXON, M. Tristage Hough Transform For Multiple Ellipse Extraction, *IEE Proceedings-E*, v.138, n.1, 1991.
- (30)-NIXON, M.S.; HAMES, T.K. New Technique for 3D Artery Modelling by Noninvasive Ultrasound, *IEE Proceedings-I*, v.140, n.1, p.86-94, 1993.
- (31)-PECKINPAUGH, Sarah H.; HOLYER, Ronald J. Circle Detection for Extrating Eddy Syze and Position from Satellite Imagery of the Ocean, *IEEE Transactions on Geocience and Remote Sensing*, v.32, n.2, p.267-273, 1994.
- (32)-PEI, Soo-Chang; HORNG, Ji-Hwei Circular Arc Detection Based on Hough Transform, *Pattern Recognition Letters*, v.16, p.615-625, 1995.
- (33)-PHILIP, K.P.; DOVE, E.L.; McPHERSON, D. D.; GOTTEINER, N.L.; STANFORD, W.; CHANDRAN, K.B. The Fuzzy Hough Transform-Feature Extraction in Medical Images, *IEEE Transactions on Medical Imaging*, v.13, n.2, p.235-240, 1994.
- (34)-RHODES, F. Matthew; DITURI, Joseph J.; CHAPMAN, Glenn H.; EMERSON, Bruce E.; SOARES, Antonio M.; RAFEL Jack I. A Monolithic

- Hough Transform Processor Based on Restructurable VLSI, IEEE Transactions on Pattern Analysis and Machine Intelligence, v.10, n.1, 1988.
- (35)-RITTER, G.X.; WILSON, J.N.; DAVIDSON, J.L. Image Algebra: An Overview, Computer Vision, Graphics, and Image Processing, v.49, p.297-331, 1990.
- (36)-SUN, Yung-Nien; TSAI, Ching-Tsorng A New Model-Based Approach for Industrial Visual Inspection, Pattern Recognition, v.25, n.11, p.1327-1336, 1992.
- (37)-TSUJI, S.; MATSUMOTO, M. Detection of Ellipses by Modified Hough Transformation, IEEE Trans., p.777-781, 1978.
- (38)-WATANABE, Takashi; SHIBATA, Toshihiro Detection of Broken Ellipses by the Hough Transforms and Multiresolutional Images, Systems and Computers in Japan, v.22, n.8, 1991.
- (39)-WU, Wen-Yen; WANG, Mao-Jiun J. Elliptical Object Detection by Using Geometric Properties, Pattern Recognition, v.26, n.10, p.1499-1509, 1993.
- (40)-YIP, Raymond K.K.; LAM, Wilson C. Y.; TAM, Peter K.S.; LEUNG, Dennis N.K. A Hough Transform Technique for the Detection of Rotational Symetry, Pattern Recognition Letters, v.15, p.919-928, 1994.
- (41)-YIP, Raymond K.K.; TAM, Peter K.S.; LEUNG Denis N.K. Modification of Hough Transform for Circles and Ellipses Detection Using a 2-Dimensional Array, Pattern Recognition , v.25, n.9, p.1007-1022, 1992.
- (42)-YUEN, H.K.; ILLINGWORTH, J.; KITTLER, J. Detecting Partially Occluded Ellipses Using the Hough Transform, v.7, n.1, p.31-39, 1989.

- (43)-YUEN, H.K.; PRINCEN, J.; ILLINGWORTH, J.; KITTLER, J. Comparative Study of Hough Transform Methods for Circle Finding, v.8, n.1, p.71-77, 1990.

Livros

- (44)- BATCHELOR, B. G.; HILL, D. A.; HODGSON, D. C. Automated Visual Inspection. North-Holland, IFS(Publications) Ltd, 1985.
- (45)-BOYLE, R. D.; THOMAS, R. C. Computer Vision - A First Course. Blackwell Scientific Publications, 1988.
- (46)-GONZALES, Rafael C.; WINTZ, Paul Digital Image Processing 2.ed., Addison-Wesley, 1987.
- (47)-JAMSA, Kris A. C/C++, 1001 dicas. Trad. de João Eduardo N. Tortello, Rio de Janeiro: LTC- Livros Técnicos e Científicos, 1994.
- (48)-KAY, David C.; LEVINE, John R. Graphics File Formats. Windcrest/McGraw-Hill, 1992.
- (49)-KITTLER, Josef; DUFF, Michael J. B. Image Processing System Architectures, Research Studies Press, 1986.
- (50)-MARR, David VISION, New York, W. H. Freeman and Company, 1980.
- (51)-MUELLER, Scott Upgrading and Repairing PCs. 2.ed., QUE, 1992.

- (52)-PETZOLD, Charles Programando para Windows 3.1. Trad. de Jeremias R. D. Pereira dos Santos, São Paulo, MAKRON Books, 1993.
- (53)-RIMMER, Steve Bit-Mapped Graphics. Windcrest, 1990.
- (54)-SCHALKOFF, Robert J. Digital Image Processing and Computer Vision. John Wiley & Sons, Inc, 1989.
- (55)-YOUNG, Tzay Y.; FU, King-Sun Handbook of Pattern Recognition and Image Processing, California, Academic Press, 1986.

Apêndice 1 - Programa fonte para detecção de formas circulares

```
/*-----  
SCANC - Detecção de Circunferências  
Paulo Cesar Lucena Bentes  
-----*/
```

Seção 1.1 - Declarações Globais

```
#include <windows.h>  
#include <windowsx.h>  
#include <commdlg.h>  
#include <stdlib.h>  
#include <math.h>  
#include <stdio.h>  
#include <string.h>  
#include <time.h>  
#include "scanc.h"  
#define MIN(a,b) ((a) < (b) ? (a) : (b))  
#define min(a,b) (((a) < (b)) ? (a) : (b))  
#define max(a,b) (((a) > (b)) ? (a) : (b))  
#define MAXIMOTAM 32  
HANDLE hCop;  
HCURSOR hcurSave;  
HPALETTE hpal=0, hpalPrev;  
HANDLE hStrings [6];  
DWORD dwTamanhoMbidPont;  
BITMAPFILEHEADER bmfh;  
short nTotal = 0;  
unsigned int raio_inf=1;  
unsigned int raio_sup=1;  
float razao=0;  
char szNomeAplic [] = "Scanc";  
unsigned int alt[27];  
unsigned int rai[27];  
unsigned int x_maior[27];  
unsigned int y_maior[27];  
float tp[3];  
char szString [MAXIMOTAM];
```

```
WORD  
int  
long FAR PASCAL _export  
int  
  
int  
  
void  
void  
  
int  
  
int  
  
void  
  
void  
void  
void  
DWORD  
unsigned int  
DWORD  
DWORD  
short  
unsigned int huge  
unsigned int  
float  
DWORD  
unsigned int  
  
GravaDib(char *,BYTE huge * );  
AcumPtr(BYTE huge *,DWORD);  
ProcJan (HWND, UINT, UINT, LONG);  
HoughCirc(unsigned int,unsigned int,BYTE huge *,  
          unsigned int huge *);  
Acumulador(unsigned int, unsigned int, unsigned int,  
            unsigned int,unsigned int, BYTE huge *,unsigned int huge *);  
GardaValor(BYTE huge *, unsigned int huge *);  
func(unsigned int, unsigned int, unsigned int, BYTE huge *,  
      unsigned int huge *);  
Bresenham(unsigned int,unsigned int,BYTE huge *,  
          unsigned int huge *);  
Template(unsigned int, unsigned int, unsigned int, unsigned int,  
          unsigned int, BYTE huge *,unsigned int huge *);  
reg(unsigned int, unsigned int, unsigned int, BYTE huge *,  
    unsigned int huge *);  
  
circulo(int, int, int, BYTE huge *);  
d_circulo(int, int, int, int,BYTE huge *);  
DrawlpMbid(int, int,BYTE huge *);  
val=0;  
x_v=0,y_v=0,z_v=0,bota=0;  
Tudo=0;  
Tam_x_y=0;  
altura=0, largura=0;  
end;  
Raio,X_cen,Y_cen;  
Topo;  
cont;  
Ampl, marcaAlt = 0, marca = 1;
```

Seção 1.2 - Criação da Janela

```
/*-----  
* JANELA PRINCIPAL  
* Declara as características da Janela  
* Cria a Janela  
-----*/
```



```

int PASCAL WinMain (HANDLE hCopia, HANDLE hPrevInstance,
                    LPSTR lpszLinhaCmd, int nCmdMostrar)
{
    HWND      hjan ;
    MSG       msg ;
    WNDCLASS  classejan ;

    if (!hPrevInstance)
    {
        classejan.style      = CS_HREDRAW | CS_VREDRAW ;
        classejan.lpfnWndProc = ProcJan ;
        classejan.cbClsExtra  = 0 ;
        classejan.cbWndExtra  = 0 ;
        classejan.hInstance  = hCopia ;
        classejan.hIcon      = LoadIcon (hCopia, "meuicone");
        classejan.hCursor    = LoadCursor (NULL, IDC_ARROW) ;
        classejan.hbrBackground = GetStockObject (WHITE_BRUSH) ;
        classejan.lpszMenuName = szNomeAplic ;
        classejan.lpszClassName = szNomeAplic ;

        RegisterClass (&classejan) ;
    }

    hCop = hCopia ;

    hjan = CreateWindow (szNomeAplic, "Scanc",
                        WS_OVERLAPPEDWINDOW | WS_VSCROLL |
                        WS_HSCROLL, CW_USEDEFAULT, CW_USEDEFAULT,
                        CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL,
                        hCopia, NULL) ;

    ShowWindow (hjan, nCmdMostrar) ;
    UpdateWindow (hjan) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
}

```

```

    }

    return msg.wParam ;
}

```

Seção 1.3 - Funções para extrair informações do ponteiro criado a partir da leitura do arquivo de imagem

```

/*****
 * Função que extrai o tamanho da estrutura BITMAPINFOHEADER em bytes
 *****/
DWORD TamCabecalhoInfoMbid (BYTE huge *lpMbids)
{
    return(((BITMAPINFOHEADER huge *) lpMbids)->biSize);
}

/*****
 * Função que extrai o valor de intensidade do vermelho da estrutura
 * RGBQUAD do pixel correspondente
 *****/
WORD ValorRrgbquad (BYTE huge *lpMbid)
{
    return((WORD)((RGBQUAD huge *) lpMbid)->rgbRed);
}

/*****
 * Função que extrai o valor de intensidade do verde da estrutura
 * RGBQUAD do pixel correspondente
 *****/
WORD ValorGrgbquad (BYTE huge *lpMbid)
{
    return((WORD)((RGBQUAD huge *) lpMbid)->rgbGreen);
}

/*****
 * Função que extrai o valor de intensidade do azul da estrutura
 * RGBQUAD do pixel correspondente
 *****/

```

```

WORD ValorBrgbquad(BYTE huge *lpMbid)
{
    return((WORD)((RGBQUAD huge *) lpMbid)->rgbBlue);
}

/*****
* Função que extrai da estrutura BITMAPINFOHEADER o valor do
* número de cores (ou número de níveis de cinza) do BITMAP
*****/

WORD NumCoresInfoMbid (BYTE huge * lpMbids)
{
    if (((BITMAPINFOHEADER huge *) lpMbids)->biClrUsed != 0)
        return((WORD) ((BITMAPINFOHEADER huge *) lpMbids)->biClrUsed);
    else
        return (WORD) pow(2,(((BITMAPINFOHEADER huge *) lpMbids)->biBitCount));
}

/*****
* Função que extrai da estrutura BITMAPINFOHEADER o valor da
* largura do mapa de bits em pontos
*****/

WORD LarguraMbid (BYTE huge *lpMbid)
{
    if (TamCabecalhoInfoMbid (lpMbid) == sizeof(BITMAPCOREHEADER))
        return (WORD) (((BITMAPCOREHEADER huge *) lpMbid)->bcWidth);
    else
        return (WORD) (((BITMAPINFOHEADER huge *) lpMbid)->biWidth);
}

/*****
* Função que extrai da estrutura BITMAPINFOHEADER o valor da
* altura do mapa de bits em pontos
*****/

WORD AlturaMbid (BYTE huge *lpMbid)
{
    if (TamCabecalhoInfoMbid (lpMbid) == sizeof(BITMAPCOREHEADER))

```

```

return (WORD) (((BITMAPCOREHEADER huge *) lpMbid)->bcHeight);
else
return (WORD) (((BITMAPINFOHEADER huge *) lpMbid)->biHeight);
}

```

```

/*****
* Função que calcula a posição do primeiro BYTE de imagem no
* ponteiro do BITMAP
*****/

BYTE huge *PegaEnderecoMbid (BYTE huge * lpMbid)
{
    DWORD dwNumCores, dwTamTabelaCores;
    WORD wContaBit;

    if (TamCabecalhoInfoMbid (lpMbid) == sizeof(BITMAPCOREHEADER))
    {
        wContaBit = ((BITMAPCOREHEADER huge *) lpMbid)->bcBitCount;

        if (wContaBit != 24)
            dwNumCores = 1L << wContaBit;
        else
            dwNumCores = 0;

        dwTamTabelaCores = dwNumCores * sizeof (RGBTRIPLE);
    }
    else
    {
        wContaBit = ((BITMAPINFOHEADER huge *) lpMbid)->biBitCount;

        if (TamCabecalhoInfoMbid (lpMbid) >= 36)
            dwNumCores = ((BITMAPINFOHEADER huge *) lpMbid)->biClrUsed;
        else
            dwNumCores = 0;

        if (dwNumCores == 0)
        {
            if (wContaBit != 24)
                dwNumCores = 1L << wContaBit;
            else
                dwNumCores = 0;
        }
    }
}

```

```

        dwTamTabelaCores = dwNumCores * sizeof( RGBQUAD );
    }

    return lpMbid + TamCabecalhoInfoMbid (lpMbid) +dwTamTabelaCores ;
}

```

Seção 1.4 - Funções de controle de caixas de diálogo

```

/*****
* Função que controla a Caixa de Diálogo usada no Sobre
*****/
BOOL FAR PASCAL _export ProcDlgSobre (HWND hDlg, UINT mensagem, UINT wParam,
                                     LONG lParam)
{
    switch (mensagem)
    {
        case WM_INITDIALOG:
            return TRUE ;

        case WM_COMMAND:
            switch (wParam)
            {
                case IDOK:
                case IDCANCEL:
                    EndDialog (hDlg, 0) ;
                    return TRUE ;

                break ;
            }

        return FALSE ;
    }

/*****
* Função que controla a Caixa de Diálogo usada na entrada do valor de
* limiar (Threshold) através do teclado
*****/

BOOL FAR PASCAL _export ProcDlg(HWND hDlg, UINT mensagem, UINT lParam,
                                LONG lParam)
{
    switch (mensagem)

```

```

    {
        case WM_INITDIALOG:
            SendDlgItemMessage (hDlg, IDD_STRING,
                               EM_LIMITTEXT,MAXIMOTAM - 1, 0L);

            return TRUE ;

        case WM_COMMAND:
            switch (wParam)
            {
                case IDOK:
                    GetDlgItemText (hDlg, IDD_STRING, szString, MAXIMOTAM);
                    EndDialog (hDlg, TRUE) ;
                    return TRUE ;

                }

            return FALSE ;
    }

/*****
* Função que controla a entrada do valor de limiar (Threshold) através
* do teclado na Caixa de Diálogo usada para receber este valor
*****/

BOOL FAR PASCAL _export AdicionaString (LPSTR lpStringEnt)
{
    int      nivIntCin ;

    nivIntCin = atoi(lpStringEnt);

    if ((0 == (strlen (lpStringEnt))) || nivIntCin > 255 || nivIntCin < 0)
        return NULL ;
    return (int) nivIntCin ;
}

/*****
* Função que tem o formato da função printf do DOS, para cinco argumento
* de saída, e gera uma caixa de diálogo para impressão desses argumentos na tela
*****/

void CxMsgOk(char *szTitulo, char *szFormato, long tempo1, long tempo2,
             long tempo3, long tempo4, long tempo5)

```



```

{
char    szBuffer[256];
char    *pArgumentos;

pArgumentos = (char *) &szFormato + sizeof szFormato;
vsprintf(szBuffer, szFormato, pArgumentos);

MessageBox(NULL, szBuffer, szTitulo, MB_OK);

}

```

Seção 1.5 - Funções para fazer ajustes na imagem

```

/*****
* Função determina quantos zeros estão acrescentados na largura da
* imagem BITMAP para que essa largura tenha valor multiplo de quatro
*****/
short Resto(BYTE huge * lpMbid)
{
short    cxMbid, desloc;

cxMbid = LarguraMbid (lpMbid);
desloc = cxMbid % 4;

if(desloc != 0) desloc = 4 - desloc;

return (short) desloc;
}

```

```

/*****
* Função que desloca os bits de uma imagem BITMAP de dezesseis cores
* para que o ponteiro de imagem possua oito bits para cada pixel
*****/
BYTE huge * BiDimPont16Cores(BYTE huge * lpMbid)
{
BYTE huge *    matUso;
BYTE huge *    bidbits;
short          cxMbid, cyMbid, deslocamento;
register int    i,j;
DWORD          Tamanho;

```

```

deslocamento = Resto(lpMbid);
bidbits = PegaEnderecoMbid (lpMbid);
cxMbid = LarguraMbid (lpMbid) + deslocamento;
cyMbid = AlturaMbid (lpMbid);

```

```

Tamanho = (long) (cxMbid) * cyMbid * sizeof(BYTE);
matUso = (BYTE huge *)GlobalAllocPtr(GMEM_MOVEABLE,Tamanho);

```

```

if(matUso==NULL) {
SetCursor(hcurSave);
MessageBox (NULL,"Nao consegui alocar memória para o ponteiro auxiliar",
szNomeAplic, MB_ICONEXCLAMATION | MB_OK);
return NULL;
}

```

```

for(i=0;i<cyMbid;i++) {
for(j=0;j<cxMbid/2;j++) {
matUso[(long) i * (cxMbid) + 2 * j] =
(bidbits[(long) i * (cxMbid)/2 + j] >> 4) & 0x0f;
matUso[(long) i * (cxMbid) + 2 * j + 1] =
bidbits[(long) i * (cxMbid)/2 + j] & 0x0f;
}
}
return matUso;
}

```

Seção 1.6 - Funções de leitura e gravação

```

/*****
* Função que lê um arquivo BITMAP no disco, separa a estrutura
* BITMAPFILEHEADER, e devolve um ponteiro contendo a estrutura
* BITMAPINFOHEADER, as estruturas RGBQUAD e os BYTES da imagem
*****/
BYTE huge * LerDib (char * szNomeArq)
{
BYTE huge *    lpMbid;
DWORD          dwTamanhoMbid, dwDesloc, dwTamCabeçalho;

```

```

int          hArquivo ;
WORD         wLerMbid ;

if (-1 == (hArquivo = _lopen (szNomeArq, OF_READ | OF_SHARE_DENY_WRITE)))
    return NULL ;

if (_lread (hArquivo, (LPSTR) &bmfh, sizeof (BITMAPFILEHEADER)) !=
    sizeof (BITMAPFILEHEADER))
{
    _lclose (hArquivo) ;
    return NULL ;
}

if (bmfh.bfType != * (WORD *) "BM")
{
    _lclose (hArquivo) ;
    return NULL ;
}

dwTamanhoMbid = bmfh.bfSize - sizeof (BITMAPFILEHEADER) ;
lpMbid = (BYTE huge *) GlobalAllocPtr (GMEM_MOVEABLE, dwTamanhoMbid) ;

if (lpMbid == NULL)
{
    _lclose (hArquivo) ;
    return NULL ;
}

dwDesloc = 0 ;

while (dwTamanhoMbid > 0)
{
    wLerMbid = (WORD) MIN (32768ul, dwTamanhoMbid) ;
    if (wLerMbid != _lread (hArquivo, (LPSTR) (lpMbid + dwDesloc), wLerMbid))
    {
        _lclose (hArquivo) ;
        GlobalFreePtr (lpMbid) ;
        return NULL ;
    }

    dwTamanhoMbid -= wLerMbid ;
    dwDesloc += wLerMbid ;
}

```

```

}

_lclose (hArquivo) ;

dwTamCabeçalho = TamCabeçalhoInfoMbid (lpMbid) ;

if (dwTamCabeçalho < 12 || (dwTamCabeçalho > 12 && dwTamCabeçalho < 16))
{
    GlobalFreePtr (lpMbid) ;
    return NULL ;
}

return lpMbid ;

}

/*****
* Função que grava um arquivo BITMAP no disco a partir do ponteiro de
* imagem devolvido na leitura
*****/
WORD GravaDib (char * szNomeArq, BYTE huge * lpMbid)
{
    DWORD dwTamanhoMbid, dwDesloc, dwTamCabeçalho ;
    int    hArquivo ;
    WORD   wLerMbid ;

    if (lpMbid == NULL) {
        MessageBox (NULL, "Abra um arquivo de imagem BMP",
            szNomeArq, MB_ICONEXCLAMATION | MB_OK) ;
        return 0 ;
    }

    else {
        if (((BITMAPINFOHEADER huge *) lpMbid)->biCompression != 0)
            ((BITMAPINFOHEADER huge *) lpMbid)->biCompression = 0 ;
        if (-1 == (hArquivo = _lcreat (szNomeArq, 0)))
            return NULL ;
        if (_lwrite (hArquivo, (LPSTR) &bmfh, sizeof (BITMAPFILEHEADER))
            sizeof (BITMAPFILEHEADER))
            return NULL ;
    }
}

```

```

dwTamanhoMbid = bmfh.bfSize - sizeof (BITMAPFILEHEADER);

if (lpMbid == NULL) {
    _lclose (hArquivo);
    return NULL;
}

dwDesloc = 0;

while (dwTamanhoMbid > 0)
{
    wLerMbid = (WORD) MIN (32768ul, dwTamanhoMbid);
    if (wLerMbid != _lwrite (hArquivo, (LPSTR) (lpMbid + dwDesloc),
                            wLerMbid))
    {
        _lclose (hArquivo);
        alFreePtr (lpMbid);
        return NULL;
    }

    wTamanhoMbid -= wLerMbid;
    wDesloc += wLerMbid;
}

_lclose (hArquivo);
dwTamCabeçalho = TamCabeçalhoInfoMbid (lpMbid);

if (dwTamCabeçalho < 12 || (dwTamCabeçalho > 12 && dwTamCabeçalho < 16))
{
    GlobalFreePtr (lpMbid);
    return NULL;
}

return 1;
}

}

/*****
* Função que grava um arquivo de valores em código ASCII no disco a
* partir do ponteiro de imagem devolvido na leitura
*****/
WORD GravaValor (char * szNomeArq, BYTE huge * lpMbid)

```

```

{
    short          cxMbid,cyMbid,desloc;
    register int    i,j;
    BYTE huge *     bidbits;
    BYTE huge *     posicao;
    FILE *          fp;

    if(lpMbid == NULL) {
        MessageBox (NULL,"Abra um arquivo de imagem BMP",
                    szNomeArq,MB_ICONEXCLAMATION | MB_OK);

        return 0;
    }

    else {
        desloc = Resto(lpMbid);

        bidbits = PegaEnderecoMbid (lpMbid);
        cxMbid = LarguraMbid (lpMbid) + desloc;
        cyMbid = AlturaMbid (lpMbid);

        if((fp=fopen(szNomeArq,"w"))==NULL) {
            MessageBox (NULL, "Não consegui abrir arquivo para os valores ASCII",
                        szNomeArq, MB_ICONEXCLAMATION | MB_OK);
            return NULL;
        }

        if(NumCoresInfoMbid(lpMbid) > 234) {
            posicao = bidbits;

            for(i=0;i<cyMbid-4;i++) {
                for(j=0;j<cxMbid-4;j++) {
                    fprintf(fp," %d", (int) posicao[(long)i * cxMbid + (long)j]);

                }
                fprintf(fp, ";\\n");
            }

            fclose(fp);
        }

        if(NumCoresInfoMbid(lpMbid) == 16) {
            posicao = BiDimPont16Cores(lpMbid);

            for(i=0;i<cyMbid-4;i++) {
                for(j=0;j<cxMbid-4;j++) {

```



```

        fprintf(fp,"%d ",(int) posicao[(long)i * cxMbid + (long)j]);
    }
    fprintf(fp,"\n");
}
fclose(fp);
}
return 1;
}
}

```

Seção 1.7 - Funções de mudanças e adaptações de palette

```

/*****
 * Função que adapta a paleta da janela tomando como referência a paleta
 * contida nas estruturas RGBQUAD do arquivo BITMAP que foi lido
 *****/
Palette256(HWND hjan, BYTE huge * lpMbid)
{
    PALETTEENTRY ape[256];
    LOGPALETTE * plgpl;
    register int i;
    HDC hdc;
    RGBQUAD huge * EndRgbQuad;

    EndRgbQuad = (RGBQUAD huge *) (lpMbid + TamCabecalhoInfoMbid(lpMbid));
    hdc = GetDC(hjan);
    plgpl = (LOGPALETTE *) GlobalAllocPtr(LPTR, sizeof(LOGPALETTE) +
        236L * sizeof(PALETTEENTRY));

    if(plgpl == NULL)
        MessageBox(hjan, "Não consegui alocar memória para nova paleta de janela",
            szNomeAplic, MB_ICONEXCLAMATION | MB_OK);

    plgpl->palNumEntries = 236;
    plgpl->palVersion = 0x300;

    for (i = 0; i < 236; i++) {
        ape[i].peRed =
            plgpl->palPalEntry[i].peRed = EndRgbQuad[i].rgbRed;

        ape[i].peGreen =
            plgpl->palPalEntry[i].peGreen = EndRgbQuad[i].rgbGreen;
    }
}

```

```

        ape[i].peBlue =
            plgpl->palPalEntry[i].peBlue = EndRgbQuad[i].rgbBlue;

        ape[i].peFlags =
            plgpl->palPalEntry[i].peFlags = PC_RESERVED;
    }
}

```

```

hpal = CreatePalette(plgpl);
GlobalFreePtr((HGLOBAL) plgpl);

```

```

ReleaseDC(hjan, hdc);
return 1;

```

```

/*****
 * Função que transforma a paleta colorida das estruturas RGBQUAD do
 * ponteiro de imagem numa paleta em níveis de cinza
 *****/
MudaPaleta(HWND hjan, BYTE huge * lpMbid)
{
    RGBQUAD huge * EndRgbQuad;
    int numEstrQuad, i;

    if(lpMbid == NULL) {
        MessageBox(hjan, "Abra um arquivo de imagem BMP",
            szNomeAplic, MB_ICONEXCLAMATION | MB_OK);

        return 0;
    }

    else {
        hcurSave = SetCursor(LoadCursor(NULL, IDC_WAIT));
        EndRgbQuad = (RGBQUAD huge *) (lpMbid + TamCabecalhoInfoMbid(lpMbid));
        numEstrQuad = NumCoresInfoMbid(lpMbid);

        for(i=0; i < numEstrQuad; i++) {
            EndRgbQuad[i].rgbRed = 235*i/numEstrQuad;
            EndRgbQuad[i].rgbGreen = 235*i/numEstrQuad;
            EndRgbQuad[i].rgbBlue = 235*i/numEstrQuad;
            EndRgbQuad[i].rgbReserved = 0;
        }
        InvalidateRect(hjan, NULL, TRUE);
    }
}

```

```
SetCursor(hcurSave);
return 0;
}
}
```

Seção 1.8 - Funções usadas para suavizar a imagem

```
/* ****
* Funções que fornecem os valores dos templates das gaussianas
* **** */
void Desvio_0_20
{
tp[0] = 3.978877;
tp[1] = 0.000015;
tp[2] = 0.000000;
}

void Desvio_0_40
{
tp[0] = 0.994719;
tp[1] = 0.043719;
tp[2] = 0.000004;
}

void Desvio_1_00
{
tp[0] = 0.159155;
tp[1] = 0.096537;
tp[2] = 0.021544;
}

/* ****
* Função que convolve o template da gaussiana com a imagem
* **** */
void Conv_5x5(HWND hjan, BYTE huge * lpMbid)
{
BYTE huge * bidbits;
BYTE huge * pos;
short cxMbid, cyMbid, desloc, l;
register int i, j;
```

```
float h1, h2, h3, h4, h5, H;
float v1, v2, v3, v4, v5, V;
int Gaussiana;
BYTE ValorPixel;

if(lpMbid == NULL) {
MessageBox (hjan, "Abra um arquivo de imagem BMP",
szNomeAplic, MB_ICONEXCLAMATION | MB_OK);
}

else {
hcurSave = SetCursor(LoadCursor(NULL, IDC_WAIT));

desloc = Resto(lpMbid);

bidbits = PegaEnderecoMbid (lpMbid);
cxMbid = LarguraMbid (lpMbid) + desloc;
cyMbid = AlturaMbid (lpMbid);

l = cxMbid;

if(NumCoresInfoMbid(lpMbid) > 234) {
pos = bidbits;

for(i=2; i<(cyMbid-2L); i++){
for(j=2; j<(cxMbid+desloc-2L); j++) {

h1=pos[(long)((long)(i-2L)*1+(long)(j-2L))]/255 * tp[2]+
pos[(long)((long)(i-2L)*1+(long)(j-1L))]/255 * tp[2]+
pos[(long)((long)(i-2L)*1+(long)(j ))]/255 * tp[2]+
pos[(long)((long)(i-2L)*1+(long)(j+1L))]/255 * tp[2]+
pos[(long)((long)(i-2L)*1+(long)(j+2L))]/255 * tp[2];
h2=pos[(long)((long)(i-1L)*1+(long)(j-2L))]/255 * tp[2]+
pos[(long)((long)(i-1L)*1+(long)(j-1L))]/255 * tp[1]+
pos[(long)((long)(i-1L)*1+(long)(j ))]/255 * tp[1]+
pos[(long)((long)(i-1L)*1+(long)(j+1L))]/255 * tp[1]+
pos[(long)((long)(i-1L)*1+(long)(j+2L))]/255 * tp[2];
h3=pos[(long)((long)(i )*1+(long)(j-2L))]/255 * tp[2]+
pos[(long)((long)(i )*1+(long)(j-1L))]/255 * tp[1]+
pos[(long)((long)(i )*1+(long)(j ))]/255 * tp[0]+
pos[(long)((long)(i )*1+(long)(j+1L))]/255 * tp[1]+
```

```

        pos[(long)((long)(i )*1+(long)(j+2L))]/255 * tp[2];
h4=pos[(long)((long)(i+1L)*1+(long)(j-2L))]/255 * tp[2]+
pos[(long)((long)(i+1L)*1+(long)(j-1L))]/255 * tp[1]+
pos[(long)((long)(i+1L)*1+(long)(j ))]/255 * tp[1]+
pos[(long)((long)(i+1L)*1+(long)(j+1L))]/255 * tp[1]+
pos[(long)((long)(i+1L)*1+(long)(j+2L))]/255 * tp[2];
h5=pos[(long)((long)(i+2L)*1+(long)(j-2L))]/255 * tp[2]+
pos[(long)((long)(i+2L)*1+(long)(j-1L))]/255 * tp[2]+
pos[(long)((long)(i+2L)*1+(long)(j ))]/255 * tp[2]+
pos[(long)((long)(i+2L)*1+(long)(j+1L))]/255 * tp[2]+
pos[(long)((long)(i+2L)*1+(long)(j+2L))]/255 * tp[2];

v1=pos[(long)((long)(i-2L)*1+(long)(j-2L))]/255 * tp[2]+
pos[(long)((long)(i-1L)*1+(long)(j-2L))]/255 * tp[2]+
pos[(long)((long)(i )*1+(long)(j-2L))]/255 * tp[2]+
pos[(long)((long)(i+1L)*1+(long)(j-2L))]/255 * tp[2]+
pos[(long)((long)(i+2L)*1+(long)(j-2L))]/255 * tp[2];
v2=pos[(long)((long)(i-2L)*1+(long)(j-1L))]/255 * tp[2]+
pos[(long)((long)(i-1L)*1+(long)(j-1L))]/255 * tp[1]+
pos[(long)((long)(i )*1+(long)(j-1L))]/255 * tp[1]+
pos[(long)((long)(i+1L)*1+(long)(j-1L))]/255 * tp[1]+
pos[(long)((long)(i+2L)*1+(long)(j-1L))]/255 * tp[2];
v3=pos[(long)((long)(i-2L)*1+(long)(j ))]/255 * tp[2]+
pos[(long)((long)(i-1L)*1+(long)(j ))]/255 * tp[1]+
pos[(long)((long)(i )*1+(long)(j ))]/255 * tp[0]+
pos[(long)((long)(i+1L)*1+(long)(j ))]/255 * tp[1]+
pos[(long)((long)(i+2L)*1+(long)(j ))]/255 * tp[2];
v4=pos[(long)((long)(i-2L)*1+(long)(j+1L))]/255 * tp[2]+
pos[(long)((long)(i-1L)*1+(long)(j+1L))]/255 * tp[1]+
pos[(long)((long)(i )*1+(long)(j+1L))]/255 * tp[1]+
pos[(long)((long)(i+1L)*1+(long)(j+1L))]/255 * tp[1]+
pos[(long)((long)(i+2L)*1+(long)(j+1L))]/255 * tp[2];
v5=pos[(long)((long)(i-2L)*1+(long)(j+2L))]/255 * tp[2]+
pos[(long)((long)(i-1L)*1+(long)(j+2L))]/255 * tp[2]+
pos[(long)((long)(i )*1+(long)(j+2L))]/255 * tp[2]+
pos[(long)((long)(i+1L)*1+(long)(j+2L))]/255 * tp[2]+
pos[(long)((long)(i+2L)*1+(long)(j+2L))]/255 * tp[2];

H = (float) (h1 + h2 + h3 + h4 + h5);
V = (float) (v1 + v2 + v3 + v4 + v5);

Gaussiana = (int)(H + V) * 255;

```

```

ValorPixel = Gaussiana;
bidbits[((i-2L) * (cxMbid)) + (j-2L)] = ValorPixel;
}

for(i=(cyMbid-3);i>=2;i--) {
    for(j=(cxMbid-3-desloc);j>=2;j--) {

        bidbits[((long)(i) * cxMbid) + (j)] =
            bidbits[((long)(i-2L) * cxMbid) + (j-2L)];

        bidbits[((i-2L) * (cxMbid)) + ((cxMbid-desloc-1L))] = 255;
        bidbits[((i-2L) * (cxMbid)) + ((cxMbid-desloc-2L))] = 255;
        bidbits[((i-2L) * (cxMbid)) + ((cxMbid-desloc-3L))] = 255;
        bidbits[((i-1L) * (cxMbid)) + ((cxMbid-desloc-1L))] = 255;
        bidbits[((i-1L) * (cxMbid)) + ((cxMbid-desloc-2L))] = 255;
        bidbits[((i-1L) * (cxMbid)) + ((cxMbid-desloc-3L))] = 255;
        bidbits[((cyMbid-1L) * (cxMbid)) + ((j-2L))] = 255;
        bidbits[((cyMbid-2L) * (cxMbid)) + ((j-2L))] = 255;
        bidbits[((cyMbid-3L) * (cxMbid)) + ((j-2L))] = 255;
        bidbits[((cyMbid-1L) * (cxMbid)) + ((j-1L))] = 255;
        bidbits[((cyMbid-2L) * (cxMbid)) + ((j-1L))] = 255;
        bidbits[((cyMbid-3L) * (cxMbid)) + ((j-1L))] = 255;

    }
}

for(i=(cyMbid-1);i>=(cyMbid-1-3L);i--) {
    for(j=(cxMbid-1-desloc);j>=(cxMbid-1-desloc-3L);j--) {
        bidbits[((long)(i) * cxMbid) + (j)] = 255;
    }
}

if(NumCoresInfoMbid(lpMbid)==16) {
    pos = BiDimPont16Cores(lpMbid);
    for(i=2;i<(cyMbid-2L);i++){
        for(j=2;j<(cxMbid-2L);j++) {
            h1=pos[(long)((long)(i-2L)*1+(long)(j-2L))]/255 * tp[2]+
            pos[(long)((long)(i-2L)*1+(long)(j-1L))]/255 * tp[2]+
            pos[(long)((long)(i-2L)*1+(long)(j ))]/255 * tp[2]+
            pos[(long)((long)(i-2L)*1+(long)(j+1L))]/255 * tp[2]+
            pos[(long)((long)(i-2L)*1+(long)(j+2L))]/255 * tp[2];

```



```

h2=pos[(long)((long)(i-1L)*1+(long)(j-2L))]/255 * tp[2]+
    pos[(long)((long)(i-1L)*1+(long)(j-1L))]/255 * tp[1]+
    pos[(long)((long)(i-1L)*1+(long)(j  ))]/255 * tp[1]+
    pos[(long)((long)(i-1L)*1+(long)(j+1L))]/255 * tp[1]+
    pos[(long)((long)(i-1L)*1+(long)(j+2L))]/255 * tp[2];
h3=pos[(long)((long)(i  )*1+(long)(j-2L))]/255 * tp[2]+
    pos[(long)((long)(i  )*1+(long)(j-1L))]/255 * tp[1]+
    pos[(long)((long)(i  )*1+(long)(j  ))]/255 * tp[0]+
    pos[(long)((long)(i  )*1+(long)(j+1L))]/255 * tp[1]+
    pos[(long)((long)(i  )*1+(long)(j+2L))]/255 * tp[2];
h4=pos[(long)((long)(i+1L)*1+(long)(j-2L))]/255 * tp[2]+
    pos[(long)((long)(i+1L)*1+(long)(j-1L))]/255 * tp[1]+
    pos[(long)((long)(i+1L)*1+(long)(j  ))]/255 * tp[1]+
    pos[(long)((long)(i+1L)*1+(long)(j+1L))]/255 * tp[1]+
    pos[(long)((long)(i+1L)*1+(long)(j+2L))]/255 * tp[2];
h5=pos[(long)((long)(i+2L)*1+(long)(j-2L))]/255 * tp[2]+
    pos[(long)((long)(i+2L)*1+(long)(j-1L))]/255 * tp[2]+
    pos[(long)((long)(i+2L)*1+(long)(j  ))]/255 * tp[2]+
    pos[(long)((long)(i+2L)*1+(long)(j+1L))]/255 * tp[2]+
    pos[(long)((long)(i+2L)*1+(long)(j+2L))]/255 * tp[2];

v1=pos[(long)((long)(i-2L)*1+(long)(j-2L))]/255 * tp[2]+
    pos[(long)((long)(i-1L)*1+(long)(j-2L))]/255 * tp[2]+
    pos[(long)((long)(i  )*1+(long)(j-2L))]/255 * tp[2]+
    pos[(long)((long)(i+1L)*1+(long)(j-2L))]/255 * tp[2]+
    pos[(long)((long)(i+2L)*1+(long)(j-2L))]/255 * tp[2];
v2=pos[(long)((long)(i-2L)*1+(long)(j-1L))]/255 * tp[2]+
    pos[(long)((long)(i-1L)*1+(long)(j-1L))]/255 * tp[1]+
    pos[(long)((long)(i  )*1+(long)(j-1L))]/255 * tp[1]+
    pos[(long)((long)(i+1L)*1+(long)(j-1L))]/255 * tp[1]+
    pos[(long)((long)(i+2L)*1+(long)(j-1L))]/255 * tp[2];
v3=pos[(long)((long)(i-2L)*1+(long)(j  ))]/255 * tp[2]+
    pos[(long)((long)(i-1L)*1+(long)(j  ))]/255 * tp[1]+
    pos[(long)((long)(i  )*1+(long)(j  ))]/255 * tp[0]+
    pos[(long)((long)(i+1L)*1+(long)(j  ))]/255 * tp[1]+
    pos[(long)((long)(i+2L)*1+(long)(j  ))]/255 * tp[2];
v4=pos[(long)((long)(i-2L)*1+(long)(j+1L))]/255 * tp[2]+
    pos[(long)((long)(i-1L)*1+(long)(j+1L))]/255 * tp[1]+
    pos[(long)((long)(i  )*1+(long)(j+1L))]/255 * tp[1]+
    pos[(long)((long)(i+1L)*1+(long)(j+1L))]/255 * tp[1]+
    pos[(long)((long)(i+2L)*1+(long)(j+1L))]/255 * tp[2];

```

```

v5=pos[(long)((long)(i-2L)*1+(long)(j+2L))]/255 * tp[2]+
    pos[(long)((long)(i-1L)*1+(long)(j+2L))]/255 * tp[2]+
    pos[(long)((long)(i  )*1+(long)(j+2L))]/255 * tp[2]+
    pos[(long)((long)(i+1L)*1+(long)(j+2L))]/255 * tp[2]+
    pos[(long)((long)(i+2L)*1+(long)(j+2L))]/255 * tp[2];

```

```

H = (float) (h1 + h2 + h3 + h4 + h5);
V = (float) (v1 + v2 + v3 + v4 + v5);

```

```

Gaussiana = (int)(H + V) * 255;
ValorPixel = Gaussiana;
bidbits[((i-2L) * (cxMbid)/2L) + (j-2L)/2L] = ValorPixel;

```

```

}
}

```

```

for(i=(cyMbid-3);i>=2;i--) {
    for(j=(cxMbid-3-desloc);j>=2;j--) {
        bidbits[((long)(i) * cxMbid/2) + (j)/2] =
            bidbits[((long)(i-2L) * cxMbid/2) + (j-2L)/2];

        bidbits[((i-2L) * (cxMbid)/2) + ((cxMbid-desloc-1L)/2)] = 255;
        bidbits[((i-2L) * (cxMbid)/2) + ((cxMbid-desloc-2L)/2)] = 255;
        bidbits[((i-2L) * (cxMbid)/2) + ((cxMbid-desloc-3L)/2)] = 255;
        bidbits[((i-1L) * (cxMbid)/2) + ((cxMbid-desloc-1L)/2)] = 255;
        bidbits[((i-1L) * (cxMbid)/2) + ((cxMbid-desloc-2L)/2)] = 255;
        bidbits[((i-1L) * (cxMbid)/2) + ((cxMbid-desloc-3L)/2)] = 255;
        bidbits[((cyMbid-1L) * (cxMbid)/2) + ((j-2L)/2)] = 255;
        bidbits[((cyMbid-2L) * (cxMbid)/2) + ((j-2L)/2)] = 255;
        bidbits[((cyMbid-3L) * (cxMbid)/2) + ((j-2L)/2)] = 255;
        bidbits[((cyMbid-1L) * (cxMbid)/2) + ((j-1L)/2)] = 255;
        bidbits[((cyMbid-2L) * (cxMbid)/2) + ((j-1L)/2)] = 255;
        bidbits[((cyMbid-3L) * (cxMbid)/2) + ((j-1L)/2)] = 255;
    }
}

```

```

for(i=(cyMbid-1);i>=(cyMbid-1-3);i--) {
    for(j=(cxMbid-1-desloc);j>=(cxMbid-1-desloc-3);j--) {
        bidbits[((long)(i) * cxMbid/2) + (j)/2] = 255;
    }
}

```

```

    }
}

InvalidateRect (hjan, NULL, TRUE);
SetCursor(hcurSave);

}

/*****
* Função que faz a suavização da imagem através da média utilizando
* um template 3x3
*****/
Media(HWND hjan, BYTE huge * lpMbid)
{
    short      cxMbid, cyMbid, l, desloc;
    BYTE huge * bidbits;
    BYTE huge * matMedia;
    register int i, j;
    BYTE      decs, decv, decf;

    if(lpMbid == NULL) {
        MessageBox (hjan, "Abra um arquivo de imagem BMP",
                    szNomeAplic, MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }

    else {
        hcurSave = SetCursor(LoadCursor(NULL, IDC_WAIT));
        desloc = Resto(lpMbid);

        bidbits = PegaEnderecoMbid (lpMbid);
        cxMbid = LarguraMbid (lpMbid) + desloc;
        cyMbid = AlturaMbid (lpMbid);

        if(NumCoresInfoMbid (lpMbid) >= 236) {
            matMedia = bidbits;

            l = cxMbid;

```

```

        for(i=0; i<cyMbid-2; i++) {
            for(j=0; j<cxMbid-2-desloc; j++) {
                decs = matMedia[(long)i * l + j] + matMedia[(long)i * l + (j+1L)] +
                    matMedia[(long)i * l + (j+2L)];
                decv = matMedia[(long)(i+1L) * l + j] + matMedia[(long)(i+1L) * l +
                    (j+1L)] + matMedia[(long)(i+1L) * l + (j+2L)];
                decf = matMedia[(long)(i+2L) * l + j] + matMedia[(long)(i+2L) * l +
                    (j+1L)] + matMedia[(long)(i+2L) * l + (j+2L)];

                decf = (decs+decv+decf)/9;
                bidbits[((long)i * cxMbid) + ((j))] = decf;
            }
        }

        for(i=0; i<cyMbid-2; i++) {
            for(j=0; j<cxMbid-1-desloc; j++) {

                bidbits[((i-1L) * cxMbid) + ((cxMbid-1L-desloc))] = 255;
                bidbits[((i-1L) * cxMbid) + ((cxMbid-2L-desloc))] = 255;
                bidbits[((i-1L) * cxMbid) + ((cxMbid-3L-desloc))] = 255;
                bidbits[((i-1L) * cxMbid) + ((cxMbid-4L-desloc))] = 255;
                bidbits[((i-1L) * cxMbid) + ((cxMbid-5L-desloc))] = 255;
            }
        }

        if(NumCoresInfoMbid (lpMbid) == 16) {
            matMedia = BiDimPont16Cores(lpMbid);
            l = cxMbid;

            for(i=0; i<cyMbid-2; i++) {
                for(j=0; j<cxMbid-2-desloc; j++) {
                    decs = matMedia[(long)i * l + j] + matMedia[(long)i * l + (j+1L)] +
                        matMedia[(long)i * l + (j+2L)];
                    decv = matMedia[(long)(i+1L) * l + j] + matMedia[(long)(i+1L) * l + (j+1L)] +
                        matMedia[(long)(i+1L) * l + (j+2L)];
                    decf = matMedia[(long)(i+2L) * l + j] + matMedia[(long)(i+2L) * l +
                        (j+1L)] + matMedia[(long)(i+2L) * l + (j+2L)];

```

```

    decf = (decs+decv+decf)/9;
    bidbits[((long)(i) * cxMbid/2L) + ((j)/2L)] = decf;
}

for(i=0;i<cyMbid-2;i++) {
    for(j=0;j<cxMbid-1-desloc;j++) {
        bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-1L)/2L)] = 255;
        bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-2L)/2L)] = 255;
        bidbits[((cyMbid-1L) * cxMbid/2L) + ((j-1L)/2L)] = 255;
        bidbits[((cyMbid-2L) * cxMbid/2L) + ((j-1L)/2L)] = 255;
        bidbits[((cyMbid-3L) * cxMbid/2L) + ((j-1L)/2L)] = 255;
        bidbits[((cyMbid-4L) * cxMbid/2L) + ((j-1L)/2L)] = 255;
    }
}

InvalidateRect (hjan, NULL, TRUE);
SetCursor(hcurSave);
return 0;
}
}

```

Seção 1.9 - Funções para detecção de bordas na imagem

```

/*****
* Função que convolui o template operador Sobel com a imagem
*****/
Sobel (HWND hjan, BYTE huge * lpMbid)
{
    BYTE huge *    bidbits;
    BYTE huge *    matSobel;
    short          cxMbid, cyMbid, l;
    register int i,j;
    int            t1, t2, t3, t4;
    int            sobel;
    BYTE           ValorPixel;
    short          desloc;

    if(lpMbid == NULL) {

```

```

        MessageBox (hjan, "Abra um arquivo de imagem BMP",
                    szNomeAplic, MB_ICONEXCLAMATION | MB_OK);
    return 0;
}

```

```

else {
    hcurSave = SetCursor(LoadCursor(NULL, IDC_WAIT));
    desloc = Resto(lpMbid);
    bidbits = PegaEnderecoMbid (lpMbid);
    cxMbid = LarguraMbid (lpMbid) + desloc;
    cyMbid = AlturaMbid (lpMbid);

    if(NumCoresInfoMbid (lpMbid) == 256) {
        matSobel = bidbits;
        l = cxMbid;

        for(i=1;i<(cyMbid-1);i++) {
            for(j=1;j<(cxMbid-1);j++) {
                t1 = (int) (-1L * (matSobel[((long)((long)(i-1L) * l + (j-1L)))] -
                    2L * (matSobel[((long)((long)(i-1L) * l + j))]-
                    1L * (matSobel[((long)((long)(i-1L) * l + (j+1L)))]));
                t3 = (int)((long)matSobel[((long)((long)(i+1L) * l + (j-1L)))]+
                    2L * matSobel[((long)((long)(i+1L) * l + j))]+
                    (long)matSobel[((long)((long)(i+1L) * l + (j+1L)))]);
                t2 = (int)(-1L * matSobel[((long)((long)(i-1L) * l + (j-1L)))]-
                    2L * matSobel[((long)((long)i * l + (j-1L)))]-
                    1L * matSobel[((long)((long)(i+1L) * l + (j-1L)))]);
                t4 = (int)((long)matSobel[((long)((long)(i-1L) * l + (j+1L)))]+
                    2L * matSobel[((long)((long)i * l + (j+1L)))]+
                    (long) matSobel[((long)((long)(i+1L) * l + (j+1L)))]);

                sobel = (int)(abs(t1+t3) + abs(t2+t4));
                ValorPixel = sobel;

                bidbits[((long)(i-1L) * cxMbid) + (j-1L)] = ValorPixel;
            }
        }

        for(i=(cyMbid-2);i>=1;i--) {
            for(j=(cxMbid-2-desloc);j>=1;j--) {
                bidbits[((long)(i) * cxMbid) + (j)] =
                    bidbits[((long)(i-1L) * cxMbid) + (j-1L)];

```



```

        bidbits[((i-1L) * cxMbid) + ((cxMbid-1L-desloc))] = 0;
        bidbits[((i-1L) * cxMbid) + ((cxMbid-2L-desloc))] = 0;
        bidbits[((i-1L) * cxMbid) + ((cxMbid-3L-desloc))] = 0;
        bidbits[((i-1L) * cxMbid) + ((cxMbid-4L-desloc))] = 0;
        bidbits[((i-1L) * cxMbid) + ((cxMbid-5L-desloc))] = 0;
        bidbits[((i-1L) * cxMbid) + ((cxMbid-6L-desloc))] = 0;
        bidbits[((cyMbid-1L) * cxMbid) + ((j-1L))] = 0;
        bidbits[((cyMbid-2L) * cxMbid) + ((j-1L))] = 0;
        bidbits[((cyMbid-3L) * cxMbid) + ((j-1L))] = 0;
    }
}

for(i=(cyMbid-1);i>=(cyMbid-1-3);i--) {
    for(j=(cxMbid-1-desloc);j>=(cxMbid-1-desloc-3);j--) {
        bidbits[((long)(i) * cxMbid) + (j)] = 0;
    }
}

if(NumCoresInfoMbid(lpMbid) == 16) {
    matSobel = BiDimPont16Cores(lpMbid);
    l = cxMbid;

    for(i=1;i<(cyMbid-1);i++) {
        for(j=1;j<(cxMbid-1-desloc);j++) {
            t1 = (int) (-1L * (matSobel[(long)((long)(i-1L) * 1 + (j-1L))]) -
                2L * (matSobel[(long)((long)(i-1L) * 1 + j)]) -
                1L * (matSobel[(long)((long)(i-1L) * 1 + (j+1L)))]);
            t3 = (int)((long)matSobel[(long)((long)(i+1L) * 1 + (j-1L))]+
                2L * matSobel[(long)((long)(i+1L) * 1 + j)]+
                (long)matSobel[(long)((long)(i+1L) * 1 + (j+1L))]);
            t2 = (int)(-1L * matSobel[(long)((long)(i-1L) * 1 + (j-1L))]-
                2L * matSobel[(long)((long)i * 1 + (j-1L))]-
                1L * matSobel[(long)((long)(i+1L) * 1 + (j-1L))]);
            t4 = (int)((long)matSobel[(long)((long)(i-1L) * 1 + (j+1L))]+
                2L * matSobel[(long)((long)i * 1 + (j+1L))]+
                (long) matSobel[(long)((long)(i+1L) * 1 + (j+1L))]);

            sobel = (int)(abs(t1+t3) + abs(t2+t4));
            ValorPixel = sobel;
        }
    }
}

```

```

        bidbits[((long)(i-1) * cxMbid/2L) + ((j-1)/2L)] = ValorPixel;
    }
}

for(i=(cyMbid-2);i>=1;i--) {
    for(j=(cxMbid-2-desloc);j>=1;j--) {
        bidbits[((long)(i) * cxMbid/2L) + (j)/2L] =
            bidbits[((long)(i-1L) * cxMbid/2L) + (j-1L)/2L];

        bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-1L-desloc)/2L)] = 0;
        bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-2L-desloc)/2L)] = 0;
        bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-3L-desloc)/2L)] = 0;
        bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-4L-desloc)/2L)] = 0;
        bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-5L-desloc)/2L)] = 0;
        bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-6L-desloc)/2L)] = 0;
        bidbits[((cyMbid-1L) * cxMbid/2L) + ((j-1L)/2L)] = 0;
        bidbits[((cyMbid-2L) * cxMbid/2L) + ((j-1L)/2L)] = 0;
        bidbits[((cyMbid-3L) * cxMbid/2L) + ((j-1L)/2L)] = 0;
    }
}

for(i=(cyMbid-1);i>=(cyMbid-1-3);i--) {
    for(j=(cxMbid-1-desloc);j>=(cxMbid-1-desloc-3);j--) {
        bidbits[((long)(i) * cxMbid/2L) + (j)/2L] = 0;
    }
}

InvalidateRect (hjan, NULL, TRUE);
SetCursor(hcurSave);
return 1;
}
}

/*****
* Função que detecta bordas na imagem a partir do operador
* ou-exclusivo
*****/
Gonzaga(HWND hjan, BYTE huge * lpMbid)
{

```

```

short      cxMbid, cyMbid, l, desloc;
BYTE huge * bidbits;
BYTE huge * matGonz;
register int i, j;
BYTE       decc, decv, decf;

if(lpMbid == NULL) {
    MessageBox (hjan, "Abra um arquivo de imagem BMP",
        szNomeAplic, MB_ICONEXCLAMATION | MB_OK);
    return 0;
}

else {
    hcurSave = SetCursor(LoadCursor(NULL, IDC_WAIT));

    desloc = Resto(lpMbid);

    bidbits = PegaEnderecoMbid (lpMbid);
    cxMbid = LarguraMbid (lpMbid) + desloc;
    cyMbid = AlturaMbid (lpMbid);

    if(NumCoresInfoMbid (lpMbid) >= 236) {
        matGonz = bidbits;

        l = cxMbid;

        for(i=0; i<cyMbid-2; i++) {
            for(j=0; j<cxMbid-1-desloc; j++) {
                decc = (BYTE) abs(matGonz[(long) i * l + j] - matGonz[(long)(i+1L) * l + j]);
                decv = (BYTE) abs(matGonz[(long) i * l + j] - matGonz[(long) i * l + (j+1L)]);
                decf = (BYTE) (decc + decv);

                decf = (decc * 16 > 8 * 16) ? 0 : 255;
                bidbits[((long)i * cxMbid) + ((j))] = decf;
            }
        }

        for(i=0; i<cyMbid-2; i++) {
            for(j=0; j<cxMbid-1-desloc; j++) {
                bidbits[((i-1L) * cxMbid) + ((cxMbid-1L-desloc))] = 255;
                bidbits[((i-1L) * cxMbid) + ((cxMbid-2L-desloc))] = 255;
            }
        }
    }
}

```

```

        bidbits[((i-1L) * cxMbid) + ((cxMbid-3L-desloc))] = 255;
        bidbits[((i-1L) * cxMbid) + ((cxMbid-4L-desloc))] = 255;
        bidbits[((i-1L) * cxMbid) + ((cxMbid-5L-desloc))] = 255;
        bidbits[((cyMbid-1L) * cxMbid) + ((j-1L))] = 255;
        bidbits[((cyMbid-2L) * cxMbid) + ((j-1L))] = 255;
        bidbits[((cyMbid-3L) * cxMbid) + ((j-1L))] = 255;
        bidbits[((cyMbid-4L) * cxMbid) + ((j-1L))] = 255;
        bidbits[((cyMbid-5L) * cxMbid) + ((j-1L))] = 255;
    }
}

for(i=(cyMbid-1); i>=(cyMbid-1-3); i--) {
    for(j=(cxMbid-1-desloc); j>=(cxMbid-1-desloc-3); j--) {
        bidbits[((long)(i) * cxMbid) + (j)] = 255;
    }
}

if(NumCoresInfoMbid (lpMbid) == 16) {
    matGonz = BiDimPont16Cores(lpMbid);
    l = cxMbid;

    for(i=0; i<cyMbid-2; i++) {
        for(j=0; j<cxMbid-1-desloc; j++) {
            decc = (BYTE) abs(matGonz[(long) i * l + j] -
                matGonz[(long)(i+1L) * l + j]);
            decv = (BYTE) abs(matGonz[(long) i * l + j] -
                matGonz[(long) i * l + (j+1L)]);
            decf = (BYTE) (decc + decv);
            decf = (decc * 16 > 8 * 16) ? 0 : 255;

            bidbits[((long)(i) * cxMbid/2L) + ((j)/2L)] = decf;
        }
    }

    for(i=0; i<cyMbid-2; i++) {
        for(j=0; j<cxMbid-1-desloc; j++) {
            bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-1L-desloc)/2L)] = 255;
            bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-2L-desloc)/2L)] = 255;
        }
    }
}

```

```

        bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-3L-desloc)/2L)] = 255;
        bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-4L-desloc)/2L)] = 255;
        bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-5L-desloc)/2L)] = 255;
        bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-6L-desloc)/2L)] = 255;
        bidbits[(cyMbid-1L) * cxMbid/2L + ((j-1L)/2L)] = 255;
        bidbits[(cyMbid-2L) * cxMbid/2L + ((j-1L)/2L)] = 255;
        bidbits[(cyMbid-3L) * cxMbid/2L + ((j-1L)/2L)] = 255;
        bidbits[(cyMbid-4L) * cxMbid/2L + ((j-1L)/2L)] = 255;
        bidbits[(cyMbid-5L) * cxMbid/2L + ((j-1L)/2L)] = 255;
        bidbits[(cyMbid-6L) * cxMbid/2L + ((j-1L)/2L)] = 255;
    }
}
for(i=(cyMbid-1);i>=(cyMbid-1-3);i--) {
    for(j=(cxMbid-1-desloc);j>=(cxMbid-1-desloc-3);j--) {
        bidbits[((long)(i) * cxMbid/2L) + (j)/2L] = 255;
    }
}

InvalidateRect (hjan, NULL, TRUE);
SetCursor(hcurSave);
return 0;
}
}

```

Seção 1.10 - Funções que binarizam a imagem

```

/*****
* Função que binariza um imagem de 256 níveis, a partir de um valor
* limiar (Threshold)
*****/
Threshold256(HWND hjan,int Nivel, BYTE huge * lpMbid)
{
    BYTE huge *    bidbits;
    short          cxMbid, cyMbid, desloc;
    register int    i,j;
    int            ValorPixel;

    hcurSave = SetCursor(LoadCursor(NULL, IDC_WAIT));
    desloc = Resto(lpMbid);

```

```

    bidbits = PegaEnderecoMbid (lpMbid) ;
    cxMbid  = LarguraMbid  (lpMbid)+ desloc ;
    cyMbid  = AlturaMbid   (lpMbid) ;

    for(i=1;i<(cyMbid-2);i++) {
        for(j=1;j<(cxMbid-2-desloc);j++) {

            ValorPixel = bidbits[((i-1L) * cxMbid) + (j-1L)];
            ValorPixel = (ValorPixel > Nivel) ? 0 : 255;

            bidbits[((i-1L) * cxMbid) + (j-1L)] = ValorPixel;

            bidbits[((i-1L) * cxMbid) + ((cxMbid-1L-desloc))] = 255;
            bidbits[((i-1L) * cxMbid) + ((cxMbid-2L-desloc))] = 255;
            bidbits[((i-1L) * cxMbid) + ((cxMbid-3L-desloc))] = 255;
            bidbits[((i-1L) * cxMbid) + ((cxMbid-4L-desloc))] = 255;
            bidbits[((i-1L) * cxMbid) + ((cxMbid-5L-desloc))] = 255;
            bidbits[((i-1L) * cxMbid) + ((cxMbid-6L-desloc))] = 255;
            bidbits[(cyMbid-1L) * cxMbid + ((j-1L))] = 255;
            bidbits[(cyMbid-2L) * cxMbid + ((j-1L))] = 255;
            bidbits[(cyMbid-3L) * cxMbid + ((j-1L))] = 255;
        }
    }
}

```

```

    for(i=(cyMbid-1);i>=(cyMbid-1-3);i--) {
        for(j=(cxMbid-1-desloc);j>=(cxMbid-1-desloc-3);j--) {
            bidbits[((long)(i) * cxMbid) + (j)] = 255;
        }
    }
    InvalidateRect (hjan, NULL, TRUE);
    SetCursor(hcurSave);
    return 0;
}

```

```

/*****
* Função que binariza um imagem de 16 níveis, a partir de um valor
* limiar (Threshold)
*****/
Threshold16(HWND hjan,int Nivel, BYTE huge * lpMbid)
{

```



```

BYTE huge *   bidbits;
BYTE huge *   matbits;
short        cxMbid, cyMbid, desloc;
register int   i,j;
int           ValorPixel;

```

```

hcurSave = SetCursor(LoadCursor(NULL, IDC_WAIT));
desloc = Resto(lpMbid);

```

```

bidbits = PegaEnderecoMbid (lpMbid);
cxMbid  = LarguraMbid   (lpMbid)+desloc;
cyMbid  = AlturaMbid    (lpMbid);

```

```

matbits = BiDimPont16Cores(lpMbid);

```

```

for(i=1;i<(cyMbid-2);i++) {
    for(j=1;j<(cxMbid-2-desloc);j++) {

```

```

        ValorPixel = matbits[(long)((i-1L) * (long) cxMbid) + (j-1L)];
        ValorPixel = (ValorPixel * 16 > Nivel) ? 0 : 255;

```

```

        bidbits[((i-1L) * cxMbid/2L) + (j-1L)/2L] = ValorPixel;

```

```

        bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-1L-desloc)/2L)] = 255;
        bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-2L-desloc)/2L)] = 255;
        bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-3L-desloc)/2L)] = 255;

```

```

void rai_5_10()

```

```

{
    raio_inf = 5 ;
    raio_sup = 10 ;
}

```

```

void rai_5_15()

```

```

{
    raio_inf = 5 ;
    raio_sup = 15 ;
}

```

```

void rai_5_25()

```

```

{
    raio_inf = 5 ;
    raio_sup = 25 ;
}

```

```

void rai_5_35()

```

```

{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

```

void rai_5_45()

```

```

{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

```

void rai_5_55()

```

```

{

```

```

    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

```

void rai_5_65()

```

```

{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

```

void rai_5_75()

```

```

{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

```

void rai_5_85()

```

```

{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

```

void rai_5_95()

```

```

{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

```

void rai_10_10()

```

```

{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

```

        bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-4L-desloc)/2L)] = 255;
        bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-5L-desloc)/2L)] = 255;
        bidbits[((i-1L) * cxMbid/2L) + ((cxMbid-6L-desloc)/2L)] = 255;
        bidbits[((cyMbid-1L) * cxMbid/2L) + ((j-1L)/2L)] = 255;
        bidbits[((cyMbid-2L) * cxMbid/2L) + ((j-1L)/2L)] = 255;
        bidbits[((cyMbid-3L) * cxMbid/2L) + ((j-1L)/2L)] = 255;

```

```

    }
}

```

```

for(i=(cyMbid-1);i>=(cyMbid-1-3);i--) {
    for(j=(cxMbid-1-desloc);j>=(cxMbid-1-desloc-3);j--) {
        bidbits[((long)i) * cxMbid/2L + (j)/2L] = 255;
    }
}

```

```

InvalidateRect (hjan, NULL, TRUE);
SetCursor(hcurSave);
return 0;
}

```

Seção 1.11 - Funções que fornecem a faixa de valores para os raios que se pretende detectar

```

void rai_5_85()

```

```

{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

```

void rai_5_95()

```

```

{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

```

void rai_10_10()

```

```

{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

```

}

```

```

void rai_10_15()

```

```

{
    raio_inf = 10 ;
    raio_sup = 15 ;
}

```

```

void rai_10_25()

```

```

{
    raio_inf = 10 ;
    raio_sup = 25 ;
}

```

```

void rai_10_35()

```

```

{

```

```

    raio_inf = 10 ;
    raio_sup = 35 ;
}

```

```

void rai_10_45()

```

```

{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

```

void rai_10_55()

```

```

{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

```

void rai_10_65()

```

```

{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

```

void rai_10_75()

```

```

{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

```

void rai_10_85()

```

```

{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

<pre> } void rai_10_95() { raio_inf = 1 ; raio_sup = 1 ; } void rai_20_10() { raio_inf = 10 ; raio_sup = 20 ; } void rai_20_15() { raio_inf = 15 ; raio_sup = 20 ; } void rai_20_25() { raio_inf = 20 ; raio_sup = 25 ; } void rai_20_35() { raio_inf = 20 ; raio_sup = 35 ; } void rai_20_45() { raio_inf = 20 ; raio_sup = 45 ; } void rai_20_55() { raio_inf = 1 ; raio_sup = 1 ; } </pre>	<pre> } void rai_20_65() { raio_inf = 1 ; raio_sup = 1 ; } void rai_20_75() { raio_inf = 1 ; raio_sup = 1 ; } void rai_20_85() { raio_inf = 1 ; raio_sup = 1 ; } void rai_20_95() { raio_inf = 1 ; raio_sup = 1 ; } void rai_30_10() { raio_inf = 10 ; raio_sup = 30 ; } void rai_30_15() { raio_inf = 15 ; raio_sup = 30 ; } void rai_30_25() { raio_inf = 25 ; raio_sup = 30 ; } </pre>	<pre> } void rai_30_35() { raio_inf = 30 ; raio_sup = 35 ; } void rai_30_45() { raio_inf = 30 ; raio_sup = 45 ; } void rai_30_55() { raio_inf = 30 ; raio_sup = 55 ; } void rai_30_65() { raio_inf = 1 ; raio_sup = 1 ; } void rai_30_75() { raio_inf = 1 ; raio_sup = 1 ; } void rai_30_85() { raio_inf = 1 ; raio_sup = 1 ; } void rai_30_95() { raio_inf = 1 ; raio_sup = 1 ; } </pre>	<pre> } void rai_40_10() { raio_inf = 1 ; raio_sup = 1 ; } void rai_40_15() { raio_inf = 15 ; raio_sup = 40 ; } void rai_40_25() { raio_inf = 25 ; raio_sup = 40 ; } void rai_40_35() { raio_inf = 35 ; raio_sup = 40 ; } void rai_40_45() { raio_inf = 40 ; raio_sup = 45 ; } void rai_40_55() { raio_inf = 40 ; raio_sup = 55 ; } void rai_40_65() { raio_inf = 40 ; raio_sup = 65 ; } </pre>	<pre> } void rai_40_75() { raio_inf = 1 ; raio_sup = 1 ; } void rai_40_85() { raio_inf = 1 ; raio_sup = 1 ; } void rai_40_95() { raio_inf = 1 ; raio_sup = 1 ; } void rai_50_10() { raio_inf = 1 ; raio_sup = 1 ; } void rai_50_15() { raio_inf = 1 ; raio_sup = 1 ; } void rai_50_25() { raio_inf = 25 ; raio_sup = 50 ; } void rai_50_35() { raio_inf = 35 ; raio_sup = 50 ; } </pre>	<pre> } void rai_50_45() { raio_inf = 45 ; raio_sup = 50 ; } void rai_50_55() { raio_inf = 50 ; raio_sup = 55 ; } void rai_50_65() { raio_inf = 50 ; raio_sup = 65 ; } void rai_50_75() { raio_inf = 50 ; raio_sup = 75 ; } void rai_50_85() { raio_inf = 1 ; raio_sup = 1 ; } void rai_50_95() { raio_inf = 1 ; raio_sup = 1 ; } void rai_60_10() { raio_inf = 1 ; raio_sup = 1 ; } </pre>	<pre> } void rai_60_15() { raio_inf = 1 ; raio_sup = 1 ; } void rai_60_25() { raio_inf = 1 ; raio_sup = 1 ; } void rai_60_35() { raio_inf = 35 ; raio_sup = 60 ; } void rai_60_45() { raio_inf = 45 ; raio_sup = 60 ; } void rai_60_55() { raio_inf = 55 ; raio_sup = 60 ; } void rai_60_65() { raio_inf = 60 ; raio_sup = 65 ; } void rai_60_75() { raio_inf = 60 ; raio_sup = 75 ; } </pre>
--	--	--	--	--	--	--

```

}

void rai_60_85()
{
    raio_inf = 60 ;
    raio_sup = 85 ;
}

void rai_60_95()
{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

void rai_70_10()
{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

void rai_70_15()
{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

void rai_90_85()
{
    raio_inf = 85 ;
    raio_sup = 90 ;
}

void rai_90_95()
{
    raio_inf = 90 ;
    raio_sup = 95 ;
}

```

```

void rai_70_25()
{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

void rai_70_35()
{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

void rai_70_45()
{
    raio_inf = 45 ;
    raio_sup = 70 ;
}

void rai_70_55()
{
    raio_inf = 55 ;
    raio_sup = 70 ;
}

void rai_70_65()
{

```

```

    raio_inf = 65 ;
    raio_sup = 70 ;
}

void rai_70_75()
{
    raio_inf = 70 ;
    raio_sup = 75 ;
}

void rai_70_85()
{
    raio_inf = 70 ;
    raio_sup = 85 ;
}

void rai_70_95()
{
    raio_inf = 70 ;
    raio_sup = 95 ;
}

void rai_80_10()
{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

```

}

void rai_80_15()
{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

void rai_80_25()
{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

void rai_80_35()
{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

void rai_80_45()
{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

```

void rai_80_55()
{
    raio_inf = 55 ;
    raio_sup = 80 ;
}

void rai_80_65()
{
    raio_inf = 65 ;
    raio_sup = 80 ;
}

void rai_80_75()
{
    raio_inf = 75 ;
    raio_sup = 80 ;
}

void rai_80_85()
{
    raio_inf = 80 ;
    raio_sup = 85 ;
}

void rai_80_95()
{

```

```

    raio_inf = 80 ;
    raio_sup = 95 ;
}

void rai_90_10()
{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

void rai_90_15()
{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

void rai_90_25()
{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

void rai_90_35()
{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

```

```

}

void rai_90_45()
{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

void rai_90_55()
{
    raio_inf = 1 ;
    raio_sup = 1 ;
}

void rai_90_65()
{
    raio_inf = 65 ;
    raio_sup = 90 ;
}

void rai_90_75()
{
    raio_inf = 75 ;
    raio_sup = 90 ;
}

```

```

/*****
* Funções que realizam a transformada de Hough Standard devolvendo um
* ponteiro contendo o acumulador
*****/

unsigned int huge * HoughStand(BYTE huge * lpMbid)
{
    BYTE huge *      bidbits;
    BYTE huge *      pos;
    short            cxMbid, novocxMbid, cyMbid, desloc;
    register unsigned int i,j;
    short            faixa_raio;
    long              Tamanho;
    long              Comprimento;
    long              zero;
    unsigned int huge * Acumul;

```

Seção 1.12 - Funções que realizam o acumulador da Transformada de Hough


```

clock_t      start, end;

if(lpMbid == NULL) {
    MessageBox (NULL, "Abra um arquivo de imagem BMP",
                szNomeAplic, MB_ICONEXCLAMATION | MB_OK);
    return 0;
}

else {

    hcurSave = SetCursor(LoadCursor(NULL, IDC_WAIT));
    desloc = Resto(lpMbid);

    bidbits = PegaEnderecoMbid (lpMbid);
    cxMbid = LarguraMbid (lpMbid)+ desloc;
    novocxMbid = LarguraMbid (lpMbid);
    cyMbid = AlturaMbid (lpMbid);

    altura = cyMbid;
    largura = cxMbid;

    faixa_raio = raio_sup - raio_inf + 1;

    if (Acumul != NULL)
        GlobalFreePtr(Acumul);

    Tamanho = (long) ((long)cxMbid * cyMbid * faixa_raio * sizeof(int));
    Comprimento = (long) ((long)cxMbid * cyMbid * faixa_raio);

    Acumul = (unsigned int huge *)GlobalAllocPtr(GMEM_MOVEABLE, Tamanho);

    if(Acumul==NULL) {

        SetCursor(hcurSave);
        MessageBox (NULL, "Nao consegui alocar memoria para Acumulador",
                    szNomeAplic, MB_ICONEXCLAMATION | MB_OK);

    return NULL;
    }

    start = clock();
    for (zero = 0; zero < Comprimento; zero++) Acumul[zero] = 0;

```

```

if(NumCoresInfoMbid(lpMbid) > 234) {
    pos = bidbits;

    for(i=0; i<(cyMbid); i++){
        for(j=0; j<(novocxMbid); j++) {
            if (pos[(long)((long)(i) * cxMbid) + (j)] < 100)
                HoughCirc(j, i, lpMbid, Acumul);
        }
    }
}

if(NumCoresInfoMbid(lpMbid)==16) {

    pos = BiDimPont16Cores(lpMbid);

    for(i=0; i<(cyMbid); i++){
        for(j=0; j<(novocxMbid); j++) {
            if (pos[(long)((long)(i) * cxMbid) + (j)] < 5)

                HoughCirc(j, i, lpMbid, Acumul);
        }
    }
}

end = clock();
SetCursor(hcurSave);
CxMsgOk(szNomeAplic, "Operação concluída em %ld segundos",
        (long)(end-start)/CLK_TCK, NULL, NULL, NULL, NULL);

return (unsigned int huge *)Acumul;
}

HoughCirc(unsigned int c_ctr, unsigned int l_ctr, BYTE huge * lpMbid,
          unsigned int huge * Acumul)

/*****
* Rotina que implementa a Transformada de Hough para circulos
*****/

{

```

```
int delta;
unsigned int y;
register unsigned int x, raio;
```

```
for (raio=raio_inf ; raio<=raio_sup; raio++) {
    delta= 3 - (raio<1);
    y=raio;
    for(x=0;x<y;x++) {
        Acumulador(x,y,c_ctr, l_ctr,raio,lpMbid,Acumul);
        delta += (delta<0) ? ((x<2)+6) : (((x-y--)<2) + 10);
    }
    if(y)
        Acumulador(y,y,c_ctr,l_ctr,raio,lpMbid,Acumul);
    return 0;
}
```

```
/* *****
* Rotina que acumula no ponteiro o numero de vezes que os circulos
* gerados pelo algoritmo de BRESENHAM se interceptam
* ***** */
Acumulador(unsigned int x, unsigned int y, unsigned int c_ctr,
            unsigned int l_ctr,unsigned int raio, BYTE huge * lpMbid,
            unsigned int huge * Acumul)
{
    register unsigned int lin, col;
    int r = 1; // razao entre a linha horizontal e vertical

    for( col=x*r ; col< (x+1)*r ; col++) {
        func( y+l_ctr, col+c_ctr, raio, lpMbid, Acumul);
        func(-y+l_ctr, col+c_ctr, raio, lpMbid, Acumul);
        func(-y+l_ctr,-col+c_ctr, raio, lpMbid, Acumul);
        func( y+l_ctr,-col+c_ctr, raio, lpMbid, Acumul);
    }

    for( lin=y*r ; lin< (y+1) * r; lin++) {
        func( x+l_ctr, lin+c_ctr, raio, lpMbid, Acumul);
        func(-x+l_ctr, lin+c_ctr, raio, lpMbid, Acumul);
    }
}
```

```
func(-x+l_ctr,-lin+c_ctr, raio, lpMbid, Acumul);
func( x+l_ctr,-lin+c_ctr, raio, lpMbid, Acumul);
}
```

```
return 0;
}
```

```
void func(unsigned int y, unsigned int x, unsigned int z, BYTE huge * lpMbid,
          unsigned int huge * Acumul)
```

```
{
    short    cxMbid, novocxMbid, cyMbid, desloc;
    long     Plano;
    long     indice;
```

```
desloc =      Resto(lpMbid);
cyMbid  =      AlturaMbid  (lpMbid);
cxMbid  =      LarguraMbid  (lpMbid) + desloc;
novocxMbid =      LarguraMbid  (lpMbid);
```

```
Plano = (long)((long)cxMbid * cyMbid);
```

```
if(x >= 0 && x < novocxMbid && y >= 0 && y < cyMbid) {
    indice = (long)((long)((long)(z-raio_inf) * Plano + (long)y * cxMbid + (long) x));

    Acumul[indice]++;

    if( *Acumul >= val) {val= *Acumul; z_v=z;y_v=y;x_v=x;}
    cont++;
}
}
```

Seção 1.13 - Funções que realizam o acumulador para o Template Matching

```
/* *****
* Funções que realizam o Template Match devolvendo um ponteiro
* contendo o acumulador
* ***** */
unsigned int huge * TemplateMatch(BYTE huge * lpMbid)
{
    short    cxMbid, novocxMbid, cyMbid, desloc;
```

```

register unsigned int i,j;
short                faixa_raio;
long                 Tamanho;
long                 Comprimento;
long                 zero;
unsigned int huge *   Acumul;
clock_t              start, end;

if(lpMbid == NULL) {
    MessageBox (NULL,"Abra um arquivo de imagem BMP",
                szNomeAplic,MB_ICONEXCLAMATION | MB_OK);
    return 0;
}

else {
    hcurSave = SetCursor(LoadCursor(NULL, IDC_WAIT));
    desloc = Resto(lpMbid);

    cxMbid   =   LarguraMbid   (lpMbid)+ desloc ;
    novocxMbid =   LarguraMbid   (lpMbid);
    cyMbid   =   AlturaMbid    (lpMbid);

    altura = cyMbid;
    largura = cxMbid;

    faixa_raio = raio_sup - raio_inf + 1;

    if (Acumul != NULL)
        GlobalFreePtr(Acumul);

    Tamanho = (long) ((long)cxMbid * cyMbid * faixa_raio * sizeof(int));
    Comprimento = (long) ((long)cxMbid * cyMbid * faixa_raio);
    Acumul = (unsigned int huge *)GlobalAllocPtr(GMEM_MOVEABLE,Tamanho);

    if(Acumul==NULL) {
        SetCursor(hcurSave);
        MessageBox (NULL, "Nao consegui alocar memoria para Acumulador",
                    szNomeAplic, MB_ICONEXCLAMATION | MB_OK);
        return NULL;
    }

    start = clock();
    for (zero = 0; zero < Comprimento; zero++) Acumul[zero] = 0;

```

```

if(NumCoresInfoMbid(lpMbid) > 234) {

for(i=0;i<(cyMbid);i++){
    for(j=0;j<(novocxMbid);j++) {
        Bresenham(j,i,lpMbid,Acumul);
    }
}

if(NumCoresInfoMbid(lpMbid)==16) {
for(i=0;i<(cyMbid);i++){
    for(j=0;j<(novocxMbid);j++) {
        Bresenham(j,i,lpMbid, Acumul);
    }
}

end = clock();
SetCursor(hcurSave);

CxMsgOk(szNomeAplic, "Operação concluída em %ld segundos",
        (long)(end-start)/CLK_TCK,NULL,NULL,NULL,NULL);

return (unsigned int huge *)Acumul;
}

}

/*****
 * Rotina que implementa o Template Match para circulos
 *****/
Bresenham(unsigned int c_ctr, unsigned int l_ctr, BYTE huge * lpMbid,
           unsigned int huge * Acumul)

{
    int delta;
    unsigned int y;
    register unsigned int x, raio;
    for (raio=raio_inf ; raio<=raio_sup; raio++) {
        delta= 3 - (raio<<1);
        y=raio;

        for(x=0;x<y;x++)
            {

```



```

Template(x,y,c_ctr,l_ctr,raio,lpMbid,Acumul);
delta += (delta<0) ? ((x<<2)+6) : (((x-y--)<<2) + 10);
}
if(y)
Template(y,y,c_ctr,l_ctr,raio,lpMbid,Acumul);
}
return 0;
}

Template(unsigned int x, unsigned int y, unsigned int c_ctr,
          unsigned int l_ctr, unsigned int raio, BYTE huge * lpMbid,
          unsigned int huge * Acumul)

/*****
* Rotina que acumula no ponteiro o numero de vezes que os circulos
* gerados pelo algoritimo de BRESENHAM se interceptam
*****/

{
register unsigned int lin, col;
int r = 1; // razao entre a linha horizontal e vertical
short cxMbid, cyMbid, desloc;
BYTE huge * bidbits;
BYTE huge * posaux;

desloc = Resto(lpMbid);
bidbits = PegaEnderecoMbid(lpMbid);
cyMbid = AlturaMbid (lpMbid);
cxMbid = LarguraMbid (lpMbid)+ desloc;

if(NumCoresInfoMbid(lpMbid) > 234) posaux = bidbits;
if(NumCoresInfoMbid(lpMbid)==16) posaux = BiDimPont16Cores(lpMbid);

for( col=x*r; col<(x+1)*r; col++) {
if((col+c_ctr)>= 0 && (col+c_ctr) < cxMbid &&
(y+l_ctr)>= 0 && (y+l_ctr) < cyMbid) {
if(posaux[(long)((long) (y+l_ctr) * cxMbid) + ((long)(col+c_ctr))]==0)
reg(l_ctr, c_ctr, raio, lpMbid, Acumul);
}
if((col+c_ctr)>= 0 && (col+c_ctr) < cxMbid &&
y+l_ctr >= 0 && (-y+l_ctr) < cyMbid) {
if(posaux[(long)((long) (-y+l_ctr) * cxMbid) + ((long)(col+c_ctr))]==0)

```

```

reg(l_ctr, c_ctr, raio, lpMbid, Acumul);
}

if((-col+c_ctr)>= 0 && (-col+c_ctr) < cxMbid &&
y+l_ctr >= 0 && (-y+l_ctr) < cyMbid) {
if(posaux[(long)((long) (-y+l_ctr) * cxMbid) + ((long)(-col+c_ctr))]==0)
reg(l_ctr, c_ctr, raio, lpMbid, Acumul);
}

if((-col+c_ctr)>= 0 && (-col+c_ctr) < cxMbid &&
(y+l_ctr)>= 0 && (y+l_ctr) < cyMbid) {
if(posaux[(long)((long) (y+l_ctr) * cxMbid) + ((long)(-col+c_ctr))]==0)
reg(l_ctr, c_ctr, raio, lpMbid, Acumul);
}

}
for( lin=y*r; lin<(y+1) * r; lin++) {
if((lin+c_ctr)>= 0 && (lin+c_ctr) < cxMbid &&
(x+l_ctr)>= 0 && (x+l_ctr) < cyMbid) {
if(posaux[(long)((long) (x+l_ctr) * cxMbid) +
((long)(lin+c_ctr))]==0)
reg(l_ctr, c_ctr, raio, lpMbid, Acumul);
}
if((lin+c_ctr)>= 0 && (lin+c_ctr) < cxMbid &&
(-x+l_ctr)>= 0 && (-x+l_ctr) < cyMbid) {
if(posaux[(long)((long) (-x+l_ctr) * cxMbid) +
((long)(lin+c_ctr))]==0)
reg(l_ctr, c_ctr, raio, lpMbid, Acumul);
}

if((-lin+c_ctr)>= 0 && (-lin+c_ctr) < cxMbid &&
(-x+l_ctr)>= 0 && (-x+l_ctr) < cyMbid) {
if(posaux[(long)((long) (-x+l_ctr) * cxMbid) +
((long)(-lin+c_ctr))]==0)
reg(l_ctr, c_ctr, raio, lpMbid, Acumul);
}
}

```

```

        if((-lin+c_ctr) >= 0 && (-lin+c_ctr) < cxMbid &&
            (x+l_ctr) >= 0 && (x+l_ctr) < cyMbid) {
            if(posaux[(long)((long) (x+l_ctr) * cxMbid) +
                ((long)(-lin+c_ctr))]==0)
                reg(1_ctr, c_ctr, raio, lpMbid, Acumul);
        }
    }

return 0;
}

void reg(unsigned int y, unsigned int x, unsigned int z, BYTE huge * lpMbid,
        unsigned int huge * Acumul)
{
    short    cxMbid,novocxMbid, cyMbid, desloc;
    long     Plano;
    long     indice;

    desloc = Resto(lpMbid);
    cyMbid = AlturaMbid (lpMbid);
    cxMbid = LarguraMbid (lpMbid)+ desloc;
    novocxMbid = LarguraMbid (lpMbid);

    Plano = (long)((long)cxMbid * cyMbid);

    if(x >= 0 && x < novocxMbid && y >= 0 && y < cyMbid) {
        indice = (long)((long)((long)(z-raio_inf) * Plano + (long)y * cxMbid + (long)x));
        Acumul[indice]++;

        if( *Acumul >= val) {val= *Acumul; z_v=z; y_v=y; x_v=x;}
        cont++;
    }
}

```

Seção 1.14 - Funções para filtrar o acumulador

```

/* ****
* Rotina que equaliza a sobreposição do acumulador dividindo
* o valor acumulado pelo perímetro da circunferência
**** */
void Equaliza(BYTE huge * lpMbid, unsigned int huge * Acumul)

```

```

{
    unsigned int    k;
    unsigned register int j,i;
    short          cxMbid, cyMbid;
    long           Plano;
    float          PtrAux;
    unsigned int    fuz;
    clock_t        start, end;

    heurSave = SetCursor(LoadCursor(NULL, IDC_WAIT));

    cyMbid = AlturaMbid (lpMbid);
    cxMbid = LarguraMbid (lpMbid) + Resto(lpMbid);

    Plano = (long) ((long)cxMbid * cyMbid);

    start = clock();

    for( k = raio_inf; k <= raio_sup; k++) {
        for (i=0; i< cyMbid; i++) {
            for(j=0; j < cxMbid; j++) {
                PtrAux=(float) (Acumul[(long)((long)(k-raio_inf) * Plano +
                    (long)i * cxMbid + (long)j)]/((float)2 * 3.14159 * k));
                fuz = (unsigned int) 255 * PtrAux;

                Acumul[(long)((long)(k-raio_inf) * Plano + (long)i * cxMbid +
                    (long)j)] = fuz;
            }
        }
    }

    marcaAlt = 1;
    end = clock();
    SetCursor(heurSave);
    CxMsgOk(szNomeAplic, "Operação concluída em %ld segundos",
        (long)(end-start)/CLK_TCK,NULL,NULL,NULL,NULL);
}

```

```

/* ****
* Rotina que convolui o template de fuzzy com o acumulador
**** */

```

```

void Fuzzy(unsigned int huge * Acumul, BYTE huge * lpMbid)
{
    unsigned int huge *    pos;
    short                 cxMbid, cyMbid, desloc, l;
    register int           i, j;
    float                 h1, h2, h3, h4, h5, H;
    float                 v1, v2, v3, v4, v5, V;
    int                   k, Fuz;
    BYTE                  ValorPixel;
    unsigned int           cume=0, PtrAux;
    long                  Plano, indice;
    float                 fz[3];
    float                 raz;
    clock_t               start, end;

    heurSave = SetCursor(LoadCursor(NULL, IDC_WAIT));

    raz=0.75;
    fz[0] = 1.000000;
    fz[1] = 0.778821;
    fz[2] = 0.367918;

    desloc = Resto(lpMbid);

    pos      = Acumul ;
    cxMbid   = LarguraMbid  (lpMbid)+ desloc ;
    cyMbid   = AlturaMbid   (lpMbid) ;

    Plano = (long) ((long)cxMbid * cyMbid );

    start = clock();

    for( k = raio_inf; k <= raio_sup; k++) {
        for (i=0; i < cyMbid; i++) {
            for(j=0; j < cxMbid; j++) {
                indice = (long)((long)(k-raio_inf) * Plano + (long)i * cxMbid +
                                (long)j);

                PtrAux =(unsigned int) (Acumul[indice]);
                if( PtrAux > cume) {
                    cume = PtrAux;

```

```

                }
            }
        }
        l = cxMbid;
        Plano = (long) cxMbid * cyMbid;

        for(k=raio_inf; k<=raio_sup; k++) {
            for(i=2; i<(cyMbid-2); i++){
                for(j=2; j<(cxMbid+desloc-2); j++) {
                    h1=pos[(long)((long) Plano * (k-raio_inf) + (long)(i-2L)*l+
                                (long)(j-2L))]/(cume * raz) * fz[2]+
                        pos[(long)((long)Plano * (k-raio_inf) +
                                (long)(i-2L)*l+(long)(j-1L))]/(cume * raz) * fz[2]+
                        pos[(long)((long) Plano * (k-raio_inf) +
                                (long)(i-2L)*l+ (long)(j  ))]/(cume * raz) * fz[2]+
                        pos[(long)((long) Plano * (k-raio_inf) + (long)(i-
                                2L)*l+ (long)(j+1L))]/(cume * raz) * fz[2]+
                        pos[(long)((long) Plano * (k-raio_inf) +
                                (long)(i-2L)*l+(long)(j+2L))]/(cume * raz) * fz[2];

                    h2=pos[(long)((long) Plano * (k-raio_inf) + (long)(i-1L)*l+
                                (long)(j-2L))]/(cume * raz) * fz[2]+
                        pos[(long)((long) Plano * (k-raio_inf) +
                                (long)(i-1L)*l+ (long)(j-1L))]/(cume * raz) * fz[1]+
                        pos[(long)((long) Plano * (k-raio_inf) +
                                (long)(i-1L)*l+(long)(j  ))]/(cume * raz) * fz[1]+
                        pos[(long)((long) Plano * (k-raio_inf) +
                                (long)(i-1L)*l+ (long)(j+1L))]/(cume * raz) * fz[1]+
                        pos[(long)((long) Plano * (k-raio_inf) +
                                (long)(i-1L)*l+(long)(j+2L))]/(cume * raz) * fz[2];

                    h3=pos[(long)((long) Plano * (k-raio_inf) + (long)(i  )*l+
                                (long)(j-2L))]/(cume * raz) * fz[2]+
                        pos[(long)((long) Plano * (k-raio_inf) +
                                (long)(i  )*l+ (long)(j-1L))]/(cume * raz) * fz[1]+
                        pos[(long)((long) Plano * (k-raio_inf) +
                                (long)(i  )*l+ (long)(j  ))]/(cume * raz) * fz[0]+
                        pos[(long)((long) Plano * (k-raio_inf) +
                                (long)(i  )*l+ (long)(j+1L))]/(cume * raz) * fz[1]+
                        pos[(long)((long) Plano * (k-raio_inf) +
                                (long)(i  )*l+(long)(j+2L))]/(cume * raz) * fz[2];

```



```

h4=pos[(long)((long) Plano * (k-raio_inf) + (long)(i+1L)*1+
(long)(j-2L))]/(cume * raz) * fz[2]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i+1L)*1+(long)(j-1L))]/(cume * raz) * fz[1]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i+1L)*1+(long)(j ))]/(cume * raz) * fz[1]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i+1L)*1+(long)(j+1L))]/(cume * raz) * fz[1]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i+1L)*1+(long)(j+2L))]/(cume * raz) * fz[2];
h5=pos[(long)((long) Plano * (k-raio_inf) + (long)(i+2L)*1+
(long)(j-2L))]/(cume * raz) * fz[2]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i+2L)*1+(long)(j-1L))]/(cume * raz) * fz[2]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i+2L)*1+(long)(j ))]/(cume * raz) * fz[2]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i+2L)*1+(long)(j+1L))]/(cume * raz) * fz[2]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i+2L)*1+(long)(j+2L))]/(cume * raz) * fz[2];

v1=pos[(long)((long) Plano * (k-raio_inf) + (long)(i-2L)*1+
(long)(j-2L))]/(cume * raz) * fz[2]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i-1L)*1+(long)(j-2L))]/(cume * raz) * fz[2]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i )*1+(long)(j-2L))]/(cume * raz) * fz[2]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i+1L)*1+(long)(j-2L))]/(cume * raz) * fz[2]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i+2L)*1+(long)(j-2L))]/(cume * raz) * fz[2];

v2=pos[(long)((long) Plano * (k-raio_inf) + (long)(i-2L)*1+
(long)(j-1L))]/(cume * raz) * fz[2]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i-1L)*1+(long)(j-1L))]/(cume * raz) * fz[1]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i )*1+(long)(j-1L))]/(cume * raz) * fz[1]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i+1L)*1+(long)(j-1L))]/(cume * raz) * fz[1]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i+2L)*1+(long)(j-1L))]/(cume * raz) * fz[2];

```

```

v3=pos[(long)((long) Plano * (k-raio_inf) + (long)(i-2L)*1+
(long)(j ))]/(cume * raz) * fz[2]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i-1L)*1+(long)(j ))]/(cume * raz) * fz[1]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i )*1+(long)(j ))]/(cume * raz) * fz[0]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i+1L)*1+(long)(j ))]/(cume * raz) * fz[1]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i+2L)*1+(long)(j ))]/(cume * raz) * fz[2];
v4=pos[(long)((long) Plano * (k-raio_inf) + (long)(i-2L)*1+
(long)(j+1L))]/(cume * raz) * fz[2]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i-1L)*1+(long)(j+1L))]/(cume * raz) * fz[1]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i )*1+(long)(j+1L))]/(cume * raz) * fz[1]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i+1L)*1+(long)(j+1L))]/(cume * raz) * fz[1]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i+2L)*1+(long)(j+1L))]/(cume * raz) * fz[2];

v5=pos[(long)((long) Plano * (k-raio_inf) + (long)(i-2L)*1+
(long)(j+2L))]/(cume * raz) * fz[2]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i-1L)*1+(long)(j+2L))]/(cume * raz) * fz[2]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i )*1+(long)(j+2L))]/(cume * raz) * fz[2]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i+1L)*1+(long)(j+2L))]/(cume * raz) * fz[2]+
pos[(long)((long) Plano * (k-raio_inf) +
(long)(i+2L)*1+(long)(j+2L))]/(cume * raz) * fz[2];

```

```

H = (float) (h1 + h2 + h3 + h4 + h5);
V = (float) (v1 + v2 + v3 + v4 + v5);

```

```
Fuz = (float)(H + V) * 4;
```

```
ValorPixel = (BYTE) Fuz;
```

```
Acumul[(long) Plano * (k-raio_inf) + (long)(i-2L) * (cxMbid) + (j-2L)] = ValorPixel;
```

```
}
```

```

}
}

for(k=raio_inf; k<=raio_sup; k++) {
    for(i=(cyMbid-3); i>=2; i--) {
        for(j=(cxMbid-3); j>=2; j--) {
            Acumul[(long) Plano * (k-raio_inf) + (long)(i) * (cxMbid) + (j)] =
            Acumul[(long) Plano * (k-raio_inf) + (long)(i-2L) * (cxMbid) + (j-2L)];

            Acumul[(long) Plano * (k-raio_inf) + ((long)(i-2L) * (cxMbid)) +
            ((cxMbid-1L))] = 0;
            Acumul[(long) Plano * (k-raio_inf) + ((long)(i-2L) * (cxMbid)) +
            ((cxMbid-2L))] = 0;
            Acumul[(long) Plano * (k-raio_inf) + ((long)(i-2L) * (cxMbid)) +
            ((cxMbid-3L))] = 0;
            Acumul[(long) Plano * (k-raio_inf) + ((long)(i-2L) * (cxMbid)) +
            ((cxMbid-4L))] = 0;
            Acumul[(long) Plano * (k-raio_inf) + ((long)(i-2L) * (cxMbid)) +
            ((cxMbid-5L))] = 0;

            Acumul[(long) Plano * (k-raio_inf) + ((long)(i-1L) * (cxMbid)) +
            ((cxMbid-1L))] = 0;
            Acumul[(long) Plano * (k-raio_inf) + ((long)(i-1L) * (cxMbid)) +
            ((cxMbid-2L))] = 0;
            Acumul[(long) Plano * (k-raio_inf) + ((long)(i-1L) * (cxMbid)) +
            ((cxMbid-3L))] = 0;
            Acumul[(long) Plano * (k-raio_inf) + ((long)(i-1L) * (cxMbid)) +
            ((cxMbid-4L))] = 0;
            Acumul[(long) Plano * (k-raio_inf) + ((long)(i-1L) * (cxMbid)) +
            ((cxMbid-5L))] = 0;

            Acumul[(long) Plano * (k-raio_inf) +
            ((long)(cyMbid-1L) * (cxMbid)) + ((j-2L))] = 0;
            Acumul[(long) Plano * (k-raio_inf) +
            ((long)(cyMbid-2L) * (cxMbid)) + ((j-2L))] = 0;
            Acumul[(long) Plano * (k-raio_inf) +
            ((long)(cyMbid-3L) * (cxMbid)) + ((j-2L))] = 0;
            Acumul[(long) Plano * (k-raio_inf) +
            ((long)(cyMbid-4L) * (cxMbid)) + ((j-2L))] = 0;
            Acumul[(long) Plano * (k-raio_inf) +
            ((long)(cyMbid-5L) * (cxMbid)) + ((j-2L))] = 0;
            Acumul[(long) Plano * (k-raio_inf) +

```

```

            ((long)(cyMbid-1L) * (cxMbid)) + ((j-1L))] = 0;
            Acumul[(long) Plano * (k-raio_inf) +
            ((long)(cyMbid-2L) * (cxMbid)) + ((j-1L))] = 0;
            Acumul[(long) Plano * (k-raio_inf) +
            ((long)(cyMbid-3L) * (cxMbid)) + ((j-1L))] = 0;
            Acumul[(long) Plano * (k-raio_inf) +
            ((long)(cyMbid-4L) * (cxMbid)) + ((j-1L))] = 0;
            Acumul[(long) Plano * (k-raio_inf) +
            ((long)(cyMbid-5L) * (cxMbid)) + ((j-1L))] = 0;

        }
    }
}

for(k=raio_inf; k<=raio_sup; k++) {
    for(i=(cyMbid-1); i>=(cyMbid-1-3); i--) {
        for(j=(cxMbid-1-desloc); j>=(cxMbid-1-desloc-3); j--) {
            Acumul[(long) Plano * (k-raio_inf) + (long)(i) * (cxMbid) + (j)] = 0;
        }
    }
}

marcaAlt = 2;
end = clock();
SetCursor(hcurSave);

CxMsgOk(szNomeAplic, "Operação concluída em %ld segundos",
        (long)(end-start)/CLK_TCK, NULL, NULL, NULL, NULL);
}

/*****
* Rotina que fornece os valores percentuais do filtro comparativo
*****/
void percent_75()
{
    razao = 0.75;
}

void percent_85()
{
    razao = 0.85;
}

void percent_90()

```

```

{
    razao = 0.90;
}

void percent_95()
{
    razao = 0.95;
}

void percent_98()
{
    razao = 0.98;
}

/*****
* FILTRO COMPARATIVO
* Rotina que determina os pontos do acumulador que são considerados
* como prováveis círculos a partir de uma comparação do valor
* corrente do acumulador com o seu valor máximo
*****/

Localiza(IHWND hjan, BYTE huge * lpMbid, unsigned int huge * Acumul)
{
    unsigned int    pico=0;
    unsigned int    k;
    unsigned register int j,i;
    short          cxMbid, cyMbid;
    long           Plano;
    unsigned int    PtrAux;
    clock_t        start,end;

    hcurSave = SetCursor(LoadCursor(NULL, IDC_WAIT));

    cyMbid = AlturaMbid (lpMbid);
    cxMbid = LarguraMbid (lpMbid) + Resto(lpMbid);

    Plano = (long) ((long)cxMbid * cyMbid);

    start = clock();

    for( k = raio_inf; k <= raio_sup; k++) {
        alt[(k-raio_inf)] = 0;
    }

```

```

        x_maior[(k-raio_inf)] = 0;
        y_maior[(k-raio_inf)] = 0;
        rai[(k-raio_inf)] = 0;

    for (i=0; i< cyMbid; i++) {
        for(j=0; j < cxMbid; j++) {
            PtrAux =(unsigned int) Acumul[(long)((long){k-raio_inf} * Plano +
                                                    (long)i * cxMbid + (long)j)];

            if( PtrAux > alt[(k-raio_inf)]) {
                if(marcaAlt ==2)
                    altAux[(k-raio_inf)] = altAuxfuzzy[(k-raio_inf)];
                if(marcaAlt == 1)
                    altAux[(k-raio_inf)] = altAuxfuzzy[(k-raio_inf)];
                if(marcaAlt == 0)
                    altAux[(k-raio_inf)] = PtrAux;
                Ampl = alt[(k-raio_inf)] = PtrAux;
                X_cen = x_maior[(k-raio_inf)] = j;
                Y_cen = y_maior[(k-raio_inf)] = i;
                Raio = rai[(k-raio_inf)] = k;
            }
            if(PtrAux > pico) {Topo = pico = PtrAux;}
        }
    }

    cont = 0;

    for( k = raio_inf; k <= raio_sup; k++) {
        if(alt[(k-raio_inf)] < (razao * pico) || alt[(k-raio_inf)] < 2*k ) {
            alt[(k-raio_inf)] = 0;
            altAux[(k-raio_inf)] = 0;
        }
    }

    end = clock();

    if(cont==0) {

```



```

        SetCursor(hcurSave);
        MessageBox (hjan,"O filtro não conseguiu detectar circulos",
                    szNomeAplic,MB_ICONEXCLAMATION | MB_OK);
    }

    SetCursor(hcurSave);
    CxMsgOk(szNomeAplic, "Operação concluída em %ld segundos",
            (long)(end-start)/CLK_TCK,NULL,NULL,NULL,NULL);

    return 1;
}

/*****
 * Rotina que transforma os vetores de posição em argumentos para que
 * as circunferências encontradas pelo filtro comparativo sejam
 * desenhadas
 *****/

void Centro(BYTE huge * lpMbid)
{
    unsigned int     ampl;
    unsigned int     raio,x_cen,y_cen;
    register unsigned int k;

    hcurSave = SetCursor(LoadCursor(NULL, IDC_WAIT));

    for( k = raio_inf; k <= raio_sup; k++) {
        ampl = alt[(k-raio_inf)];
        raio = rai[(k-raio_inf)];
        y_cen = y_maior[(k-raio_inf)];
        x_cen = x_maior[(k-raio_inf)];

        if(ampl != 0) {
            circulo(x_cen,y_cen,raio,lpMbid);
        }
    }

    SetCursor(hcurSave);
}

```

Seção 1.15 - Funções que permitem a visualização dos resultados na tela

```

/*****
 * Rotina que imprime na tela a posição, o raio e o valor acumulado
 * dos circulos encontrados pelo filtro comparativo
 *****/

void Lista(BYTE huge * lpMbid)
{
    unsigned int     ampl, indice = 1;
    unsigned int     raio,x_cen,y_cen,val_y;
    short            cyMbid;
    register unsigned int k;

    cyMbid = AlturaMbid(lpMbid);

    for( k = raio_inf; k <= raio_sup; k++) {
        ampl = alt[(k-raio_inf)];
        raio = rai[(k-raio_inf)];
        y_cen = y_maior[(k-raio_inf)];
        x_cen = x_maior[(k-raio_inf)];
        val_y = (int)cyMbid - y_cen - 1;

        if(ampl != 0) {
            CxMsgOk(szNomeAplic,"Circulo %ld , Raio : %ld , X : %ld , Y : %ld ,
                            Acum: %ld", indice, raio, x_cen, val_y, ampl);

            indice++;
        }
    }

/*****
 * Função que desenha um circulo a partir das coordenadas do centro e do valor do raio
 * Utiliza o algoritmo de BRESENHAM.
 * Recebe como argumento as coordenadas do centro e o raio
 *****/

void circulo(int c_ctr, int l_ctr, int raio, BYTE huge * lpMbid)
{

```

```

int delta= 3 - (raio<<1);
int x, y=raio;

hcurSave = SetCursor(LoadCursor(NULL, IDC_WAIT));

for(x=0;x<y;x++)
{
    d_circulo(x,y,c_ctr,l_ctr,lpMbid);
    delta += (delta<0) ? ((x<<2)+6) : (((x-y--)<<2) + 10);
}

if(y) d_circulo(y,y,c_ctr,l_ctr,lpMbid);
SetCursor(hcurSave);
}

void d_circulo(int y, int x, int c_ctr, int l_ctr, BYTE huge * lpMbid)
{
    int lin, col;
    int r = 1; /* razao entre a linha horizontal e vertical */

    /* Desenha os oito pontos */

    for( col=x*r; col< (x+1)*r; col++)
    {
        DrawlpMbid( col+c_ctr, y+l_ctr, lpMbid); /* caso 4*/
        DrawlpMbid( col+c_ctr, -y+l_ctr, lpMbid); /* caso 1*/
        DrawlpMbid(-col+c_ctr, -y+l_ctr, lpMbid); /* caso 8*/
        DrawlpMbid(-col+c_ctr, y+l_ctr, lpMbid); /* caso 5*/
    }

    for( lin=y*r; lin< (y+1) * r; lin++)
    {
        DrawlpMbid( lin+c_ctr, x+l_ctr, lpMbid); /* caso 3*/
        DrawlpMbid( lin+c_ctr, -x+l_ctr, lpMbid); /* caso 2*/
        DrawlpMbid(-lin+c_ctr, -x+l_ctr, lpMbid); /* caso 7*/
        DrawlpMbid(-lin+c_ctr, x+l_ctr, lpMbid); /* caso 6*/
    }
}

void DrawlpMbid(int x, int y, BYTE huge * lpMbid)

```

```

{
    BYTE huge *      bidbits;
    short            cxMbid,cyMbid,desloc;

    desloc = Resto(lpMbid);

    bidbits = PegaEnderecoMbid (lpMbid) ;
    cxMbid   = LarguraMbid   (lpMbid) + desloc;
    cyMbid   = AlturaMbid    (lpMbid) ;

    if(NumCoresInfoMbid(lpMbid) > 234) {
        if(x >= 0 && x < (cxMbid - desloc) && y >= 0 && y < cyMbid) {
            bidbits[(DWORD)((long) y * cxMbid + ((long)x))] = 2;
        }
    }

    if(NumCoresInfoMbid(lpMbid)==16) {
        if(x >= 0 && x < (cxMbid - desloc) && y >= 0 && y < cyMbid) {
            bidbits[(DWORD)((long) y * cxMbid/2 + ((long)x/2))] = 2;
        }
    }
}

/* *****
* Rotina que coloca no ponteiro de renovação de tela a sobreposição
* do Acumulador
* ***** */
MostraSobrAcum(HWND hjan, BYTE huge * lpMbid, unsigned int huge * Acumul)
{
    float            divisao;
    unsigned int     k;
    long             indice;
    unsigned int     pico;
    BYTE             normalizado;
    short            cxMbid, cyMbid, desloc ;
    register int     i,j;
    BYTE huge *      bidbits;
    long             Plano;

    hcurSave = SetCursor(LoadCursor(NULL, IDC_WAIT));

```

```

desloc = Resto(lpMbid);

bidbits = PegaEnderecoMbid (lpMbid);
cxMbid = LarguraMbid (lpMbid) + desloc;
cyMbid = AlturaMbid (lpMbid);

Plano = (long)((long)cxMbid * cyMbid);

for(j=0; j<cyMbid; j++) {
    for(i=0; i<cxMbid; i++) {
        pico = 0;
        for(k=raio_inf; k<=raio_sup; k++) {

            if(j >= 0 && j < cyMbid && i >= 0 && i < cxMbid) {
                indice = (long)((long)(k-raio_inf) * Plano +
                                (long)i * cxMbid + (long)j));

                if(Acumul[indice] > pico) {
                    pico = Acumul[indice];
                }
            }
        }

        divisao = (float)((float)pico/1.);
        if(NumCoresInfoMbid (lpMbid) >= 236) {
            normalizado = (BYTE) ((float)1 * divisao);
            bidbits[((long)i * cxMbid) + ((j))] = normalizado;
        }
        if(NumCoresInfoMbid (lpMbid) == 16) {
            normalizado = (BYTE) ( 16 * divisao );

            bidbits[((long)i * cxMbid/2) + ((j/2))] = normalizado;
        }
    }
}

InvalidateRect (hjan, NULL, TRUE);
SetCursor(hcurSave);
return 0;
}

```

Seção 1.16 - Procedimento de Janela

```

/*****
* Procedimento de Janela
*****/
long FAR PASCAL _export ProcJan (HWND hjan, UINT mensa,
                                UINT wParam, LONG lParam)

{
    static FARPROC lpfnProcDlgSobre;
    static char  szNomeArq [_MAX_PATH], szTitulo [_MAX_FNAME + _MAX_EXT];
    static char * szFiltro[] = { "Arquivos Bitmap (*.BMP)", "*.bmp",
                                "Arquivos MBIDs (*.DIB)", "*.dib",
                                "Arquivos ASCII (*.DAT)", "*.dat", "" };

    BYTE huge *    matUso;
    static BYTE huge * lpMbid;
    static BYTE huge * MbidPont;
    static BYTE huge * VetDirGrad;
    static unsigned int huge * Acumul;
    static OPENFILENAME ofn;
    static short cxCliente, cyCliente;
    static WORD wExibe = IDM_REAL;
    static HICON hIcon;
    static HDC hdc;
    static HMENU hMenu;
    static BYTE huge * lpMbidBits, gpMbid;
    static PAINTSTRUCT ps;
    static short cxMbid, cyMbid;
    static HANDLE hInst;
    static FARPROC lpfnProcDlg;
    static int Threshold;
    static short cxCarac, cyCarac, nRolaPosV, nRolaVMax, nRolaPosH,
                                nRolaHMax;

    static short nIncRolaV, nIncRolaH;

    switch (mensa)
    {
        case WM_CREATE:
            ofn.lStructSize = sizeof (OPENFILENAME);
            ofn.hwndOwner = hjan;
            ofn.lpstrFilter = szFiltro [0];
            ofn.lpstrFile = szNomeArq;
            ofn.nMaxFile = _MAX_PATH;
            ofn.lpstrFileName = szTitulo;
            ofn.nMaxFileName = _MAX_FNAME + _MAX_EXT;

```



```

ofn.lpstrDefExt      =      "bmp" ;

cyCarac = 48;
cxCarac = 64;

hIcon = LoadIcon (hCop, "meuicone");
hInst = ((LPCREATESTRUCT) lParam)->hInstance ;
lpfnProcDlg = MakeProcInstance ((FARPROC) ProcDlg, hInst);
lpfnProcDlgSobre = MakeProcInstance ((FARPROC) ProcDlgSobre,
                                     hInst);

return 0 ;

case WM_SIZE:
    cxCliente = LOWORD (lParam) ;
    cyCliente = HIWORD (lParam) ;
    return 0 ;

case WM_VSCROLL:
    switch (wParam)
    {
        case SB_TOP:
            nIncRolaV = -nRolaPosV ;
            break ;

        case SB_BOTTOM:
            nIncRolaV = nRolaVMax - nRolaPosV ;
            break ;

        case SB_LINEUP:
            nIncRolaV = -8 ;
            break ;

        case SB_LINEDOWN:
            nIncRolaV = 8 ;
            break ;

        case SB_PAGEUP:
            nIncRolaV = -16;
            break ;
    }

```

```

        case SB_PAGEDOWN:
            nIncRolaV = 16;
            break ;

        case SB_THUMBTRACK:
            nIncRolaV = LOWORD (lParam) - nRolaPosV ;
            break ;

        default:
            nIncRolaV = 0 ;
    }
    nIncRolaV = max (-nRolaPosV,
                    min (nIncRolaV, nRolaVMax - nRolaPosV));

    if (nIncRolaV != 0)
    {
        nRolaPosV += nIncRolaV ;
        ScrollWindow (hJan, 0, -cyCarac * nIncRolaV, NULL, NULL);
        SetScrollPos (hJan, SB_VERT, nRolaPosV, TRUE);
        InvalidateRect (hJan, NULL, TRUE);
        UpdateWindow (hJan);
    }

    return 0 ;

case WM_HSCROLL:
    switch (wParam)
    {
        case SB_LINEUP:
            nIncRolaH = -8 ;
            break ;

        case SB_LINEDOWN:
            nIncRolaH = 8 ;
            break ;

        case SB_PAGEUP:
            nIncRolaH = -16 ;
            break ;

        case SB_PAGEDOWN:
            nIncRolaH = 16 ;
            break ;
    }

```

```

        case SB_THUMBPOSITION:
            nIncRolaH = LOWORD (lParam) - nRolaPosH ;
            break ;

        default:
            nIncRolaH = 0 ;
    }

    nIncRolaH = max (-nRolaPosH, min (nIncRolaH, nRolaHMax - nRolaPosH)) ;

    if (nIncRolaH != 0)
    {
        nRolaPosH += nIncRolaH ;
        ScrollWindow (hJan, -cxCarac * nIncRolaH, 0, NULL, NULL);
        SetScrollPos (hJan, SB_HORZ, nRolaPosH, TRUE);
        InvalidateRect (hJan, NULL, TRUE);
        UpdateWindow (hJan);
    }
    return 0 ;

case WM_COMMAND:
    hMenu = GetMenu (hJan);

    switch (wParam)
    {
        case IDM_SOBRE:
            DialogBox(hInst, "CaixaSobre", hJan, lpfnProcDlgSobre);
            return 0;

        case IDM_ABRIR:
            if (GetOpenFileName (&ofn))
            {
                if (lpMbid != NULL)
                {
                    GlobalFreePtr (lpMbid);
                    lpMbid = NULL;
                }

                if (MbidPont != NULL)
                {
                    GlobalFreePtr (MbidPont);
                    MbidPont = NULL;
                }
            }
    }

```

```

    }

    if (VetDirGrad != NULL)
    {
        GlobalFreePtr (VetDirGrad);
        VetDirGrad = NULL;
    }

    lpMbid = LcrDib (szNomeArq);
    if (lpMbid == NULL)
        MessageBox (hJan,
            "Nao consegui abrir o arquivo do BITMAP",
            szNomeAplic, MB_ICONEXCLAMATION |
            MB_OK);

    if (NumCoresInfoMbid (lpMbid) == 256)
        if (Palette256(hJan,lpMbid) == NULL)
            MessageBox (hJan,
                "Nao foi possivel mudar palette para 256 niveis",
                szNomeAplic, MB_ICONEXCLAMATION |
                MB_OK);

    InvalidateRect (hJan, NULL, TRUE);
}
return 0 ;

case IDM_SALVARCO:
    ofn.Flags = OFN_OVERWRITEPROMPT;

    if (GetSaveFileName (&ofn))
    {
        gpMbid = GravaDib( szNomeArq, lpMbid);
        if (gpMbid == NULL)
            MessageBox (hJan,"Nao consegui gravar o arquivo do MBID",
                szNomeAplic, MB_ICONEXCLAMATION | MB_OK);
    }
    return 0 ;

case IDM_SALVALOR:
    ofn.Flags = OFN_OVERWRITEPROMPT;
    if (GetSaveFileName (&ofn))
    {
        if (GravaValor( szNomeArq, lpMbid) == NULL)
            MessageBox (hJan,

```

```

        "Nao consegui gravar o arquivo do Valor",
        szNomeAplic, MB_ICONEXCLAMATION |
        MB_OK);
    }
    return 0;

case IDM_CINZA:
    MudaPaleta(hjan, lpMbid);
    return 0;

case IDM_GAUSS02:
    Desvio_0_20;
    Conv_5x5(hjan,lpMbid);
    return 0;
case IDM_GAUSS04:
    Desvio_0_40;
    Conv_5x5(hjan,lpMbid);
    return 0;

case IDM_GAUSS1:
    Desvio_1_00;
    Conv_5x5(hjan,lpMbid);
    return 0;

case IDM_SOBEL:
    Sobel(hjan,lpMbid);
    return 0;

case IDM_BINARIZAR:
    if(lpMbid == NULL)
    {
        MessageBox (hjan,
        "Abra um arquivo de imagem BMP",
        szNomeAplic,MB_ICONEXCLAMATION|
        MB_OK);
        return 0;
    }

    else
        if(MessageBox (hInst, "IncluirDlg", hjan,
        lpfnProcDlg))
            Threshold = AdicionaString (szString);

```

```

        if(NULL == Threshold )
        {
            MessageBox (hjan,
            "Valor de Threshold invalido",
            szNomeAplic,
            MB_ICONEXCLAMATION | MB_OK);
            break ;
        }

    else {
        if (matUso != NULL)
        {
            GlobalFreePtr (matUso);
            matUso = NULL;
        }

        if(NumCoresInfoMbid(lpMbid) > 234)
        {
            Threshold256(hjan,Threshold,lpMbid);
        }

        if(NumCoresInfoMbid(lpMbid) == 16)
        {
            Threshold16(hjan,Threshold,lpMbid);
        }
    }

    return 0;

case IDM_GONZAGA:
    Gonzaga(hjan, lpMbid);
    ReleaseDC (hjan,hdc);
    return 0;

case IDM_MEDIA:
    Media(hjan, lpMbid);
    ReleaseDC (hjan,hdc);
    return 0;

case IDM_SRAI10:
    rai_5_100;
    if(raio_inf==1 && raio_sup==1)

```



```

        {
            MessageBox (hjan,"Escolha fora de faixa",
            szNomeAplic,MB_ICONEXCLAMATION | MB_OK);
            return 0;
        }
        Acumul=HoughStand(lpMbid);
        return 0;

case IDM_5RAI15:
    rai_5_150;
    if(rai_5_inf==1 && rai_5_sup==1)
    {
        MessageBox (hjan,"Escolha fora de faixa",
        szNomeAplic,MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }
    Acumul=HoughStand(lpMbid);

    return 0;

case IDM_5RAI25:
    rai_5_250;
    if(rai_5_inf==1 && rai_5_sup==1)
    {
        MessageBox (hjan,"Escolha fora de faixa",
        szNomeAplic,MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }
    Acumul=HoughStand(lpMbid);
    return 0;

case IDM_5RAI35:
    rai_5_350;
    if(rai_5_inf==1 && rai_5_sup==1)
    {
        MessageBox (hjan,"Escolha fora de faixa",
        szNomeAplic,MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }
    Acumul=HoughStand(lpMbid);
    return 0;

```

```

case IDM_5RAI45:
    rai_5_450;
    if(rai_5_inf==1 && rai_5_sup==1)
    {
        MessageBox (hjan,"Escolha fora de faixa",
        szNomeAplic,MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }
    Acumul=HoughStand(lpMbid);
    return 0;

```

Estes casos são repetidos até "case IDM_90RAI95"

```

case IDM_T5RAI10:
    rai_5_100;
    if(rai_5_inf==1 && rai_5_sup==1)
    {
        MessageBox (hjan,"Escolha fora de faixa",
        szNomeAplic,MB_ICONEXCLAMATION |
        MB_OK);
        return 0;
    }
    Acumul=TemplateMatch(lpMbid);
    return 0;

case IDM_T5RAI15:
    rai_5_150;
    if(rai_5_inf==1 && rai_5_sup==1)
    {
        MessageBox (hjan,"Escolha fora de faixa",
        szNomeAplic,MB_ICONEXCLAMATION |
        MB_OK);
        return 0;
    }
    Acumul=TemplateMatch(lpMbid);
    return 0;

case IDM_T5RAI25:
    rai_5_250;
    if(rai_5_inf==1 && rai_5_sup==1)
    {

```

```

        MessageBox (hjan,"Escolha fora de faixa",
        szNomeAplic,MB_ICONEXCLAMATION |
        MB_OK);
        return 0;
    }
    Acumul=TemplateMatch(lpMbid);
    return 0;

```

Estes casos se repetem até "case IDM_T90RAI95"

```

case IDM_EQUALIZA:
    if(Acumul == NULL)
    {
        MessageBox (hjan,
        "Execute um dos detectores antes dos filtros",
        szNomeAplic,MB_ICONEXCLAMATION |
        MB_OK);
        return 0;
    }
    else
        Equaliza(lpMbid,Acumul);
        return 0;

case IDM_FUZZY:
    if(Acumul == NULL)
    {
        MessageBox (hjan,
        "Execute um dos detectores antes dos filtros",
        szNomeAplic,MB_ICONEXCLAMATION |
        MB_OK);
        return 0;
    }
    else
        Fuzzy(Acumul,lpMbid);
        return 0;

case IDM_75P:
    if(Acumul == NULL)
    {
        MessageBox (hjan,

```

```

        "Execute um dos detectores antes dos filtros",
        szNomeAplic,MB_ICONEXCLAMATION |
        MB_OK);
        return 0;
    }

```

```

    else
        percent_75();
        Localiza(hjan,lpMbid, Acumul);
        return 0;

```

```

case IDM_85P:
    if(Acumul == NULL)
    {
        MessageBox (hjan,
        "Execute um dos detectores antes dos filtros",
        szNomeAplic,MB_ICONEXCLAMATION |
        MB_OK);
        return 0;
    }
    else
        percent_85();
        Localiza(hjan,lpMbid, Acumul);
        return 0;

```

```

case IDM_90P:
    if(Acumul == NULL)
    {
        MessageBox (hjan,
        "Execute um dos detectores antes dos filtros",
        szNomeAplic,MB_ICONEXCLAMATION |
        MB_OK);
        return 0;
    }
    else
        percent_90();
        Localiza(hjan,lpMbid, Acumul);
        return 0;

```

```

case IDM_95P:
    if(Acumul == NULL)
    {
        MessageBox (hjan,

```

```

        "Execute um dos detectores antes dos filtros",
        szNomeAplic, MB_ICONEXCLAMATION |
        MB_OK);
        return 0;
    }
else
    percent_95();
    Localiza(hjan, lpMbid, Acumul);
    return 0;

case IDM_98P:
    if(Acumul == NULL)
    {
        MessageBox(hjan,
        "Execute um dos detectores antes dos filtros",
        szNomeAplic, MB_ICONEXCLAMATION |
        MB_OK);
        return 0;
    }
else
    percent_98();
    Localiza(hjan, lpMbid, Acumul);
    return 0;

case IDM_CIRCULO:
    Centro(lpMbid);
    InvalidateRect(hjan, NULL, TRUE);
    return 0;

case IDM_ACMASOBRE:
    if(Acumul == NULL)
    {
        MessageBox(hjan,
        "Execute um dos detectores antes",
        szNomeAplic, MB_ICONEXCLAMATION |
        MB_OK);
        return 0;
    }
else
    MostraSobrAcum(hjan, lpMbid, Acumul);
    return 0;

```

```

case IDM_LISTA:
    if(lpMbid == NULL)
    {
        MessageBox(hjan,
        "Abra um arquivo de imagem BMP",
        szNomeAplic, MB_ICONEXCLAMATION |
        MB_OK);
        return 0;
    }
else
    if(alt == NULL)
    {
        MessageBox(hjan,
        "Execute o filtro comparativo antes",
        szNomeAplic, MB_ICONEXCLAMATION |
        MB_OK);
        return 0;
    }
else
    Lista(lpMbid);
    return 0;

case IDM_REAL:
case IDM_EXPANDIR:
    CheckMenuItem(hMenu, wExibe, MF_UNCHECKED);
    wExibe = wParam;
    CheckMenuItem(hMenu, wExibe, MF_CHECKED);
    InvalidateRect(hjan, NULL, TRUE);
    return 0;

case IDM_SAIR:
    PostQuitMessage(0);
    return 0;
}
break;

case WM_PAINT:
    hdc = BeginPaint(hjan, &ps);
    /* Use o handle da palette. */
    hpalPrev = SelectPalette(hdc, hpal, FALSE);
    RealizePalette(hdc);
    if(lpMbid != NULL)

```



```

        {
            lpMbidBits = PegaEnderecoMbid (lpMbid);
            cxMbid = LarguraMbid (lpMbid);
            cyMbid = AlturaMbid (lpMbid);
            SetStretchBltMode (hdc, COLORONCOLOR);
        }
    if (wExibe == IDM_REAL)
        SetDIBitsToDevice (hdc, 0, 0, cxMbid, cyMbid, nRolaPosH,
                           0, nRolaPosV, cyMbid, (LPSTR) lpMbidBits,
                           (LPBITMAPINFO) lpMbid, DIB_RGB_COLORS);

    if (wExibe == IDM_EXPANDIR)
        StretchDIBits (hdc, 0, 0, cxCliente, cyCliente, 0, 0, cxMbid, cyMbid,
                       (LPSTR) lpMbidBits, (LPBITMAPINFO) lpMbid,
                       DIB_RGB_COLORS, SRCCOPY);
}
SelectPalette(hdc, hpalPrev, FALSE);
RealizePalette(hdc);
EndPaint (hjan, &ps);
nRolaVMax = max (0, cyMbid - cyCliente);
nRolaPosV = min (nRolaPosV, nRolaVMax);
SetScrollRange (hjan, SB_VERT, 0, nRolaVMax, FALSE);
SetScrollPos (hjan, SB_VERT, nRolaPosV, TRUE);
nRolaHMax = max (0, cxMbid - cxCliente);
nRolaPosH = min (nRolaPosH, nRolaHMax);
SetScrollRange (hjan, SB_HORZ, 0, nRolaHMax, FALSE);
SetScrollPos (hjan, SB_HORZ, nRolaPosH, TRUE);
return 0;

case WM_DESTROY:
    if (lpMbid != NULL)
        GlobalFreePtr (lpMbid);

    if (matUso != NULL)
        GlobalFreePtr (matUso);

    if (VetDirGrad != NULL)
        GlobalFreePtr (VetDirGrad);

    if (Acumul != NULL)
        GlobalFreePtr (Acumul);

    if (hpal != 0)

```

```

        DeleteObject(hpal);

        PostQuitMessage (0);
        return 0;
    }
    return DefWindowProc (hjan, mensa, wParam, lParam);
}

```

Apêndice 2 - Lista de Recursos para o Scanc

```
/*-----
Scanc.RC Lista de Recursos
-----*/
```

```
#include "scanc.h"
meuicone ICON scanc.ico
scanc MENU
{
    POPUP "&Arquivo"
    {
        MENUITEM "&Abrir...",          IDM_ABRIR
        MENUITEM "S&alvar &Como...",    IDM_SALVARCO
        MENUITEM "S&alvar &Valor Como...", IDM_SALVALOR
        MENUITEM "S&ai&r",              IDM_SAIR
    }
    POPUP "&Exibir"
    {
        MENUITEM "&Tamanho Real",      IDM_REAL, CHECKED
        MENUITEM "&Expandir na Janela", IDM_EXPANDIR
    }
    POPUP "&Ferramentas"
    {
        MENUITEM "&Cinza",              IDM_CINZA
        POPUP "&Gaussiana"
        {
            MENUITEM "Desvio=0,&2",    IDM_GAUSS02
            MENUITEM "Desvio=0,&4",    IDM_GAUSS04
            MENUITEM "Desvio=&1",      IDM_GAUSS1
        }
        MENUITEM "&Media",              IDM_MEDIA
        MENUITEM "B&i&narizar",        IDM_BINARIZAR
        POPUP "&Bordas",
        {
            MENUITEM "&Sobel",          IDM_SOBEL
            MENUITEM "&Gonzaga",        IDM_GONZAGA
        }
    }
}
```

```
}
POPUP "&Detectores"
{
    POPUP "&Hough Standard"
    {
        POPUP "raio &inferior: 5"
        {
            MENUITEM "raio &superior: 10", IDM_5RAI10
            MENUITEM "raio &superior: 15", IDM_5RAI15
            MENUITEM "raio &superior: 25", IDM_5RAI25
            MENUITEM "raio &superior: 35", IDM_5RAI35
            MENUITEM "raio &superior: 45", IDM_5RAI45
            MENUITEM "raio &superior: 55", IDM_5RAI55
            MENUITEM "raio &superior: 65", IDM_5RAI65
            MENUITEM "raio &superior: 75", IDM_5RAI75
            MENUITEM "raio &superior: 85", IDM_5RAI85
            MENUITEM "raio &superior: 95", IDM_5RAI95
        }
        POPUP "raio &inferior: 10"
        {
            MENUITEM "raio &superior: 10", IDM_10RAI10
            MENUITEM "raio &superior: 15", IDM_10RAI15
            MENUITEM "raio &superior: 25", IDM_10RAI25
            MENUITEM "raio &superior: 35", IDM_10RAI35
            MENUITEM "raio &superior: 45", IDM_10RAI45
            MENUITEM "raio &superior: 55", IDM_10RAI55
            MENUITEM "raio &superior: 65", IDM_10RAI65
            MENUITEM "raio &superior: 75", IDM_10RAI75
            MENUITEM "raio &superior: 85", IDM_10RAI85
            MENUITEM "raio &superior: 95", IDM_10RAI95
        }
        POPUP "raio &inferior: 20"
        {
            MENUITEM "raio &superior: 10", IDM_20RAI10
            MENUITEM "raio &superior: 15", IDM_20RAI15
            MENUITEM "raio &superior: 25", IDM_20RAI25
            MENUITEM "raio &superior: 35", IDM_20RAI35
            MENUITEM "raio &superior: 45", IDM_20RAI45
            MENUITEM "raio &superior: 55", IDM_20RAI55
        }
    }
}
```



```

        {
            MENUITEM "raio &superior: 10", IDM_90RAI10
            MENUITEM "raio &superior: 15", IDM_90RAI15
            MENUITEM "raio &superior: 25", IDM_90RAI25
            MENUITEM "raio &superior: 35", IDM_90RAI35
            MENUITEM "raio &superior: 45", IDM_90RAI45
            MENUITEM "raio &superior: 55", IDM_90RAI55
            MENUITEM "raio &superior: 65", IDM_90RAI65
            MENUITEM "raio &superior: 75", IDM_90RAI75
            MENUITEM "raio &superior: 85", IDM_90RAI85
            MENUITEM "raio &superior: 95", IDM_90RAI95
        }
    }
POPUP "&Template matching"
{
    POPUP "raio &inferior: 3"
    {
        MENUITEM "raio &superior: 10", IDM_T5RAI10
        MENUITEM "raio &superior: 15", IDM_T5RAI15
        MENUITEM "raio &superior: 25", IDM_T5RAI25
        MENUITEM "raio &superior: 35", IDM_T5RAI35
        MENUITEM "raio &superior: 45", IDM_T5RAI45
        MENUITEM "raio &superior: 55", IDM_T5RAI55
        MENUITEM "raio &superior: 65", IDM_T5RAI65
        MENUITEM "raio &superior: 75", IDM_T5RAI75
        MENUITEM "raio &superior: 85", IDM_T5RAI85
        MENUITEM "raio &superior: 95", IDM_T5RAI95
    }

    POPUP "raio &inferior: 10"
    {
        MENUITEM "raio &superior: 10", IDM_T10RAI10
        MENUITEM "raio &superior: 15", IDM_T10RAI15
        MENUITEM "raio &superior: 25", IDM_T10RAI25
        MENUITEM "raio &superior: 35", IDM_T10RAI35
        MENUITEM "raio &superior: 45", IDM_T10RAI45
        MENUITEM "raio &superior: 55", IDM_T10RAI55
        MENUITEM "raio &superior: 65", IDM_T10RAI65
        MENUITEM "raio &superior: 75", IDM_T10RAI75
        MENUITEM "raio &superior: 85", IDM_T10RAI85
        MENUITEM "raio &superior: 95", IDM_T10RAI95
    }

    POPUP "raio &inferior: 20"
    {

```

```

        MENUITEM "raio &superior: 10", IDM_T20RAI10
        MENUITEM "raio &superior: 15", IDM_T20RAI15
        MENUITEM "raio &superior: 25", IDM_T20RAI25
        MENUITEM "raio &superior: 35", IDM_T20RAI35
        MENUITEM "raio &superior: 45", IDM_T20RAI45
        MENUITEM "raio &superior: 55", IDM_T20RAI55
        MENUITEM "raio &superior: 65", IDM_T20RAI65
        MENUITEM "raio &superior: 75", IDM_T20RAI75
        MENUITEM "raio &superior: 85", IDM_T20RAI85
        MENUITEM "raio &superior: 95", IDM_T20RAI95
    }
POPUP "raio &inferior: 30"
{
    MENUITEM "raio &superior: 10", IDM_T30RAI10
    MENUITEM "raio &superior: 15", IDM_T30RAI15
    MENUITEM "raio &superior: 25", IDM_T30RAI25
    MENUITEM "raio &superior: 35", IDM_T30RAI35
    MENUITEM "raio &superior: 45", IDM_T30RAI45
    MENUITEM "raio &superior: 55", IDM_T30RAI55
    MENUITEM "raio &superior: 65", IDM_T30RAI65
    MENUITEM "raio &superior: 75", IDM_T30RAI75
    MENUITEM "raio &superior: 85", IDM_T30RAI85
    MENUITEM "raio &superior: 95", IDM_T30RAI95
}

POPUP "raio &inferior: 40"
{
    MENUITEM "raio &superior: 10", IDM_T40RAI10
    MENUITEM "raio &superior: 15", IDM_T40RAI15
    MENUITEM "raio &superior: 25", IDM_T40RAI25
    MENUITEM "raio &superior: 35", IDM_T40RAI35
    MENUITEM "raio &superior: 45", IDM_T40RAI45
    MENUITEM "raio &superior: 55", IDM_T40RAI55
    MENUITEM "raio &superior: 65", IDM_T40RAI65
    MENUITEM "raio &superior: 75", IDM_T40RAI75
    MENUITEM "raio &superior: 85", IDM_T40RAI85
    MENUITEM "raio &superior: 95", IDM_T40RAI95
}

POPUP "raio &inferior: 50"
{
    MENUITEM "raio &superior: 10", IDM_T50RAI10
    MENUITEM "raio &superior: 15", IDM_T50RAI15
    MENUITEM "raio &superior: 25", IDM_T50RAI25
    MENUITEM "raio &superior: 35", IDM_T50RAI35
    MENUITEM "raio &superior: 45", IDM_T50RAI45

```

```

        MENUITEM "raio &superior: 55", IDM_T50RAI55
        MENUITEM "raio &superior: 65", IDM_T50RAI65
        MENUITEM "raio &superior: 75", IDM_T50RAI75
        MENUITEM "raio &superior: 85", IDM_T50RAI85
        MENUITEM "raio &superior: 95", IDM_T50RAI95
    }
POPUP "raio &inferior: 60"
{
    MENUITEM "raio &superior: 10", IDM_T60RAI10
    MENUITEM "raio &superior: 15", IDM_T60RAI15
    MENUITEM "raio &superior: 25", IDM_T60RAI25
    MENUITEM "raio &superior: 35", IDM_T60RAI35
    MENUITEM "raio &superior: 45", IDM_T60RAI45
    MENUITEM "raio &superior: 55", IDM_T60RAI55
    MENUITEM "raio &superior: 65", IDM_T60RAI65
    MENUITEM "raio &superior: 75", IDM_T60RAI75
    MENUITEM "raio &superior: 85", IDM_T60RAI85
    MENUITEM "raio &superior: 95", IDM_T60RAI95
}
POPUP "raio &inferior: 70"
{
    MENUITEM "raio &superior: 10", IDM_T70RAI10
    MENUITEM "raio &superior: 15", IDM_T70RAI15
    MENUITEM "raio &superior: 25", IDM_T70RAI25
    MENUITEM "raio &superior: 35", IDM_T70RAI35
    MENUITEM "raio &superior: 45", IDM_T70RAI45
    MENUITEM "raio &superior: 55", IDM_T70RAI55
    MENUITEM "raio &superior: 65", IDM_T70RAI65
    MENUITEM "raio &superior: 75", IDM_T70RAI75
    MENUITEM "raio &superior: 85", IDM_T70RAI85
    MENUITEM "raio &superior: 95", IDM_T70RAI95
}
POPUP "raio &inferior: 80"
{
    MENUITEM "raio &superior: 10", IDM_T80RAI10
    MENUITEM "raio &superior: 15", IDM_T80RAI15
    MENUITEM "raio &superior: 25", IDM_T80RAI25
    MENUITEM "raio &superior: 35", IDM_T80RAI35
    MENUITEM "raio &superior: 45", IDM_T80RAI45
    MENUITEM "raio &superior: 55", IDM_T80RAI55
    MENUITEM "raio &superior: 65", IDM_T80RAI65
    MENUITEM "raio &superior: 75", IDM_T80RAI75
    MENUITEM "raio &superior: 85", IDM_T80RAI85
    MENUITEM "raio &superior: 95", IDM_T80RAI95
}

```

```

    }
POPUP "raio &inferior: 90"
{
    MENUITEM "raio &superior: 10", IDM_T90RAI10
    MENUITEM "raio &superior: 15", IDM_T90RAI15
    MENUITEM "raio &superior: 25", IDM_T90RAI25
    MENUITEM "raio &superior: 35", IDM_T90RAI35
    MENUITEM "raio &superior: 45", IDM_T90RAI45
    MENUITEM "raio &superior: 55", IDM_T90RAI55
    MENUITEM "raio &superior: 65", IDM_T90RAI65
    MENUITEM "raio &superior: 75", IDM_T90RAI75
    MENUITEM "raio &superior: 85", IDM_T90RAI85
    MENUITEM "raio &superior: 95", IDM_T90RAI95
}
}

POPUP "Fi&ltros"
{
    POPUP "&Comparativo"
    {
        MENUITEM " razão de 75% ", IDM_75P
        MENUITEM " razão de 85% ", IDM_85P
        MENUITEM " razão de 90% ", IDM_90P
        MENUITEM " razão de 95% ", IDM_95P
        MENUITEM " razão de 98% ", IDM_98P
    }
    MENUITEM "&Equaliza", IDM_EQUALIZA
    MENUITEM "&Fuzzy", IDM_FUZZY
}

POPUP "&Visualizar"
{
    MENUITEM "&Circulos", IDM_CIRCULO
    MENUITEM "&Sobreposição do Acumulador", IDM_ACMASOBRE
    MENUITEM "&Dados do círculo", IDM_LISTA
}
MENUITEM " So&br " IDM_SOBRE
}

```

```
IncluirDlg DIALOG 24, 24, 190, 44 STYLE WS_POPUP | WS_DLGFRAME
{
    LTEXT                "&Thresh:", 0, 4, 8, 24, 8
    EDITTEXT             IDD_STRING, 32, 6, 154, 12
    DEFPUSHBUTTON        "OK",    IDOK,    70, 24, 32, 14
}
```

```
CaixaSobre DIALOG 20, 20, 160, 80 STYLE WS_POPUP | WS_DLGFRAME
{
    CTEXT "Scanc"                -1, 0, 12, 160, 8
    ICON  "meuicone"             -1, 8, 8, 0, 0
    CTEXT "USP - EESC - Dep. de Eng. Elétrica" -1, 0, 24, 160, 8
    CTEXT "Grupo de Visão Computacional"      -1, 0, 36, 160, 8
    CTEXT "Paulo Cesar Lucena Bentes, 1996"    -1, 0, 48, 160, 8
    DEFPUSHBUTTON "OK"            IDOK, 64, 60, 32, 14, WS_GROUP
}
```


Apêndice 3 - Definições para o Scanc

```
/*-----
COMANDOS.H Arquivo de Cabeçalho
-----*/
```

```
#define IDM_ABRIR          100
#define IDM_SALVARCO       110
#define IDM_SALVALOR       120
#define IDM_SAIR           130
#define IDM_REAL           200
#define IDM_EXPANDIR       210
#define IDM_CINZA          300
#define IDM_GAUSS02        400
#define IDM_GAUSS04        410
#define IDM_GAUSS1         420
#define IDM_MEDIA          310
#define IDM_BINARIZAR      320
#define IDM_SOBEL          500
#define IDM_GONZAGA        510
#define IDM_EXECUTE        330
#define IDM_5RAI10         600
#define IDM_5RAI15         601
#define IDM_5RAI25         602
#define IDM_5RAI35         603
#define IDM_5RAI45         604
#define IDM_5RAI55         605
#define IDM_5RAI65         606
#define IDM_5RAI75         607
#define IDM_5RAI85         608
#define IDM_5RAI95         609
#define IDM_10RAI10        610
#define IDM_10RAI15        611
#define IDM_10RAI25        612
#define IDM_10RAI35        613
#define IDM_10RAI45        614
#define IDM_10RAI55        615
#define IDM_10RAI65        616
#define IDM_10RAI75        617
#define IDM_10RAI85        618
#define IDM_10RAI95        619
```

```
#define IDM_20RAI10        620
#define IDM_20RAI15        621
#define IDM_20RAI25        622
#define IDM_20RAI35        623
#define IDM_20RAI45        624
#define IDM_20RAI55        625
#define IDM_20RAI65        626
#define IDM_20RAI75        627
#define IDM_20RAI85        628
#define IDM_20RAI95        629
#define IDM_30RAI10        630
#define IDM_30RAI15        631
#define IDM_30RAI25        632
#define IDM_30RAI35        633
#define IDM_30RAI45        634
#define IDM_30RAI55        635
#define IDM_30RAI65        636
#define IDM_30RAI75        637
#define IDM_30RAI85        638
#define IDM_30RAI95        639
#define IDM_40RAI10        640
#define IDM_40RAI15        641
#define IDM_40RAI25        642
#define IDM_40RAI35        643
#define IDM_40RAI45        644
#define IDM_40RAI55        645
#define IDM_40RAI65        646
#define IDM_40RAI75        647
#define IDM_40RAI85        648
#define IDM_40RAI95        649
#define IDM_50RAI10        650
#define IDM_50RAI15        651
#define IDM_50RAI25        652
#define IDM_50RAI35        653
#define IDM_50RAI45        654
#define IDM_50RAI55        655
#define IDM_50RAI65        656
#define IDM_50RAI75        657
#define IDM_50RAI85        658
```

```
#define IDM_50RAI95        659
#define IDM_60RAI10        660
#define IDM_60RAI15        661
#define IDM_60RAI25        662
#define IDM_60RAI35        663
#define IDM_60RAI45        664
#define IDM_60RAI55        665
#define IDM_60RAI65        666
#define IDM_60RAI75        667
#define IDM_60RAI85        668
#define IDM_60RAI95        669
#define IDM_70RAI10        670
#define IDM_70RAI15        671
#define IDM_70RAI25        672
#define IDM_70RAI35        673
#define IDM_70RAI45        674
#define IDM_70RAI55        675
#define IDM_70RAI65        676
#define IDM_70RAI75        677
#define IDM_70RAI85        678
#define IDM_70RAI95        679
#define IDM_80RAI10        680
#define IDM_80RAI15        681
#define IDM_80RAI25        682
#define IDM_80RAI35        683
#define IDM_80RAI45        684
#define IDM_80RAI55        685
#define IDM_80RAI65        686
#define IDM_80RAI75        687
#define IDM_80RAI85        688
#define IDM_80RAI95        689
#define IDM_90RAI10        690
#define IDM_90RAI15        691
#define IDM_90RAI25        692
#define IDM_90RAI35        693
#define IDM_90RAI45        694
#define IDM_90RAI55        695
#define IDM_90RAI65        696
#define IDM_90RAI75        697
```

```

#define IDM_90RAI85      698
#define IDM_90RAI95      699
#define IDM_T5RAI10      700
#define IDM_T5RAI15      701
#define IDM_T5RAI25      702
#define IDM_T5RAI35      703
#define IDM_T5RAI45      704
#define IDM_T5RAI55      705
#define IDM_T5RAI65      706
#define IDM_T5RAI75      707
#define IDM_T5RAI85      708
#define IDM_T5RAI95      709
#define IDM_T10RAI10     710
#define IDM_T10RAI15     711
#define IDM_T10RAI25     712
#define IDM_T10RAI35     713
#define IDM_T10RAI45     714
#define IDM_T10RAI55     715
#define IDM_T10RAI65     716
#define IDM_T10RAI75     717
#define IDM_T10RAI85     718
#define IDM_T10RAI95     719
#define IDM_T20RAI10     720
#define IDM_T20RAI15     721
#define IDM_T20RAI25     722
#define IDM_T20RAI35     723
#define IDM_T20RAI45     724
#define IDM_T20RAI55     725
#define IDM_T20RAI65     726
#define IDM_T20RAI75     727
#define IDM_T20RAI85     728
#define IDM_T20RAI95     729
#define IDM_T30RAI10     730
#define IDM_T30RAI15     731
#define IDM_T30RAI25     732
#define IDM_T30RAI35     733
#define IDM_T30RAI45     734
#define IDM_T30RAI55     735
#define IDM_T30RAI65     736
#define IDM_T30RAI75     737
#define IDM_T30RAI85     738
#define IDM_T30RAI95     739

```

```

#define IDM_T40RAI10     740
#define IDM_T40RAI15     741
#define IDM_T40RAI25     742
#define IDM_T40RAI35     743
#define IDM_T40RAI45     744
#define IDM_T40RAI55     745
#define IDM_T40RAI65     746
#define IDM_T40RAI75     747
#define IDM_T40RAI85     748
#define IDM_T40RAI95     749
#define IDM_T50RAI10     750
#define IDM_T50RAI15     751
#define IDM_T50RAI25     752
#define IDM_T50RAI35     753
#define IDM_T50RAI45     754
#define IDM_T50RAI55     755
#define IDM_T50RAI65     756
#define IDM_T50RAI75     757
#define IDM_T50RAI85     758
#define IDM_T50RAI95     759
#define IDM_T60RAI10     760
#define IDM_T60RAI15     761
#define IDM_T60RAI25     762
#define IDM_T60RAI35     763
#define IDM_T60RAI45     764
#define IDM_T60RAI55     765
#define IDM_T60RAI65     766
#define IDM_T60RAI75     767
#define IDM_T60RAI85     768
#define IDM_T60RAI95     769
#define IDM_T70RAI10     770
#define IDM_T70RAI15     771
#define IDM_T70RAI25     772
#define IDM_T70RAI35     773
#define IDM_T70RAI45     774
#define IDM_T70RAI55     775
#define IDM_T70RAI65     776
#define IDM_T70RAI75     777
#define IDM_T70RAI85     778
#define IDM_T70RAI95     779
#define IDM_T80RAI10     780
#define IDM_T80RAI15     781

```

```

#define IDM_T80RAI25     782
#define IDM_T80RAI35     783
#define IDM_T80RAI45     784
#define IDM_T80RAI55     785
#define IDM_T80RAI65     786
#define IDM_T80RAI75     787
#define IDM_T80RAI85     788
#define IDM_T80RAI95     789
#define IDM_T90RAI10     790
#define IDM_T90RAI15     791
#define IDM_T90RAI25     792
#define IDM_T90RAI35     793
#define IDM_T90RAI45     794
#define IDM_T90RAI55     795
#define IDM_T90RAI65     796
#define IDM_T90RAI75     797
#define IDM_T90RAI85     798
#define IDM_T90RAI95     799
#define IDM_75P           801
#define IDM_85P           802
#define IDM_90P           803
#define IDM_95P           804
#define IDM_98P           805
#define IDM_EQUALIZA      810
#define IDM_FUZZY         820
#define IDM_CIRCULO       910
#define IDM_ACMSOBRE      920
#define IDM_LISTA         930
#define IDM_SOBRE         1000
#define IDD_STRING        0x10

```

Apêndice 4 - Algoritmo de Bresenham para desenhar círculos implementado em C

```

circulo(int l_ctr, int c_ctr, int raio)
{
    int delta = 3 - 2 * raio;
    int x, y = raio;

    for(x = 0; x < y; x++)
    {
        d_circulo(x, y, l_ctr, c_ctr);
        delta += (delta < 0) ? (4 * x + 6) : (4 * (x-y--) + 10);
    }

    if(y) d_circulo(y, y, l_ctr, c_ctr);
}

d_circulo(int x, int y, int l_ctr, int c_ctr);
{
    int lin, col;
    int r = 1; /* razão entre a linha horizontal e vertical */

    /* Desenha os oito pontos */
    for(lin = x * r; lin < (x + 1) * r; lin++)
    {
        imprpt( lin+l_ctr, y+c_ctr);
        imprpt( lin+l_ctr, -y+c_ctr);
        imprpt( -lin+l_ctr, -y+c_ctr);
        imprpt( -lin+l_ctr, y+c_ctr);
    }

    for(col=y * r; col < (y+1) * r; col++)
    {
        imprpt( col+l_ctr, x+c_ctr);
        imprpt( col+l_ctr, -x+c_ctr);
        imprpt( -col+l_ctr, -x+c_ctr);
        imprpt( -col+l_ctr, x+c_ctr);
    }
}

```


Apêndice 5 - Programa fonte para obter os valores para formar o *template* da Gaussiana

```

/*****
* Obtenção dos valores para o template da gaussiana
*****/

#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>

void main()
{
    int DIM,NDIM,r,x,i,intl;
    float ncp,desvio;
    unsigned char resp;

    int temp[10][10];
    float gaussiana[10],teste,teste1;

    ncp=2.718;

    parte1:

    clrscr();

    printf("\nentre com o valor do desvio padrao:");

    scanf("%f",&desvio);

    /*DIM deve ter apenas valores impares*/

    printf("\nentre com o valor da dimensao do template:");

    scanf("%d",&DIM);

    if((DIM==3)||(DIM==5)||(DIM==7)||(DIM==9)||(DIM==11)||(DIM==13)||(DIM==15))
        x=(DIM-1)/2;
    else
    {
        printf("\n Valor da dimensao deve ser impar entre 3 e 15\n");
    }

```

```

        if((DIM>15)||(DIM<3))
        {
            printf("\n Dimensao esta fora de intervalo");
            if(DIM>15)
            {
                x=7;
                printf("\n Dimensao entao sera 15\n"); }
            if(DIM<3)
            {
                x=1;
                printf("\n Dimensao entao sera 3\n");
                printf("\n"); } }

        else
        {
            x=(DIM-2)/2;
            NDIM=DIM-1;
            printf("\n o tamanho da dimensao sera %d ",NDIM);
            printf("\n\n");
        } }

    for(i=0;i<=x;i++){
        r=i;

        teste=((1)/(2 * 3.14159 * pow(desvio,2)));
        teste1=1/pow(ncp,(pow(r,2)/(2*(pow(desvio,2)))));
        gaussiana[i]= teste * teste1*(255);
        printf("\n");
        intl=floor(gaussiana[i]);

        if(gaussiana[i]-intl>0.5)
            temp[0][i]=ceil(gaussiana[i]);
        else
            temp[0][i]=intl;

        printf("\n  %d  %f  %f",temp[0][i],gaussiana[i],gaussiana[i]/255);

    } getch();

        printf("\n");
        printf("\n");

    clrscr();

```

```
printf("\nVoce deseja calcular um novo template? (sim ou qualquer tecla):");  
  
resp=getche();  
  
if(resp=='s')  
{ clrscr();  
  goto parte1; }  
  
clrscr();  
  
}
```

Apêndice 6 - Programa fonte para obter os valores para formar o *template* do Filtro Fuzzy

```

/*****
* Obtenção dos valores para o template da gaussiana *
*****/
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
void main()
{
    int DIM,NDIM,r,x,i,intl,k;
    float nep;
    unsigned char resp;
    int temp[10][10];
    float fuzzy[10],teste,teste1;
    nep=2.718;
    k = 255;
    parte1:
        clrscr();

    /*DIM deve ter apenas valores impares*/
    printf("\nentre com o valor da dimensao do template:");
    scanf("%d",&DIM);

    if((DIM==3)||(DIM==5)||(DIM==7)||(DIM==9)||(DIM==11)||(DIM==13)||(DIM==15))
        x=(DIM-1)/2;
    else
    {
        printf("\n Valor da dimensao deve ser impar entre 3 e 15\n");

        if((DIM>15)||(DIM<3))
        {
            printf("\n Dimensao esta fora de intervalo");
            if(DIM>15)
            {
                x=7;
                printf("\n Dimensao entao sera 15\n");
            }
            if(DIM<3)
            {
                x=1;
                printf("\n Dimensao entao sera 3\n");
                printf("\n");
            }
        }
    }
}

```

```

    { x=(DIM-2)/2;
      NDIM=DIM-1;
      printf("\n o tamanho da dimensao sera %d ",NDIM);
      printf("\n\n");
    } }

    for(i=0;i<=x;i++){
        r=i;
        teste= k;
        teste1=1/pow(nep,(pow(r,2)/pow(x,2)));
        fuzzy[i]= teste* teste1;
        printf("\n");
        intl=floor(fuzzy[i]);

        if(fuzzy[i]-intl>0.5)
            temp[0][i]=ceil(fuzzy[i]);
        else
            temp[0][i]=intl;

        printf("\n  %d  %f  %f",temp[0][i],fuzzy[i],fuzzy[i]/255);

    } getch();

        printf("\n");
        printf("\n");

    clrscr();

    printf("\nVoce deseja calcular um novo template? (sim ou qualquer tecla):");

    resp=getche();

    if(resp=='s')
    {
        clrscr();

        goto parte1;
    }

    clrscr();
}

```


Apêndice 7 - Programa que calcula o número de pontos de um círculo construído a partir do Algoritmo de Bresenham para círculos

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>

void main()
{
    int raio, numero;
    int delta;
    int x, y;
    unsigned char resp;

    parte1:
    numero = 0;
    printf("\nentre com o valor do raio:");
    scanf("%d",&raio);
    delta = 3 - 2 * raio;
    y = raio;

    for(x = 0; x < y; x++)
    {
        numero= numero + 8;
        delta += (delta<0) ? (4 * x + 6) : (4 * (x-y--) + 10);
    }
    if(y) numero = numero + 8;

    printf("\nnúmero de pontos para este raio: %d ",numero);
    getch();

    printf("\n");
    printf("\n");

    clrscr();

    printf("\nVoce deseja calcular novamente? (sim ou qualquer tecla):");
    resp=getche();
    if(resp=='s')
    { clrscr();
      goto parte1; }

    clrscr();
}
```

Apêndice 8 - Programa usado para contar o número de pontos de borda existentes numa imagem

```
#include <windows.h>
#include <windowsx.h>
#include <stdlib.h>
#include <stdarg.h>
#include <commdlg.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include "scanc.h"
#include "contador.h"

#define MIN(a,b) ((a) < (b) ? (a) : (b))

long FAR PASCAL _export ProcJan (HWND, UINT, UINT, LONG);

char szNomeAplic [] = "MostraMbid";

int PASCAL WinMain (HANDLE hCopia, HANDLE hPrevInstance,
                    LPSTR lpszLinhaCmd, int
                    nCmdMostrar)
{
    HWND    hjan;
    MSG     msg;
    WNDCLASS classejan;

    if (!hPrevInstance)
    {
        classejan.style      = CS_HREDRAW | CS_VREDRAW;
        classejan.lpfnWndProc = ProcJan;
        classejan.cbClsExtra  = 0;
        classejan.cbWndExtra  = 0;
        classejan.hInstance  = hCopia;
        classejan.hIcon       = NULL;
        classejan.hCursor     = LoadCursor (NULL, IDC_ARROW);
        classejan.hbrBackground = GetStockObject (WHITE_BRUSH);
        classejan.lpszMenuName = szNomeAplic;
        classejan.lpszClassName = szNomeAplic;

        RegisterClass (&classejan);
    }
}
```

```
    }

    hjan = CreateWindow (szNomeAplic, "Contador",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT,
        NULL, NULL, hCopia, NULL);

    ShowWindow (hjan, nCmdMostrar);
    UpdateWindow (hjan);

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return msg.wParam;
}

// Funcoes para extrair informacoes dos blocos de memoria do MBID

DWORD TamCabecalhoInfoMbid (BYTE huge * lpMbids)
{
    return ((BITMAPINFOHEADER huge *) lpMbids)->biSize;
}

WORD LarguraMbid (BYTE huge * lpMbid)
{
    if (TamCabecalhoInfoMbid (lpMbid) == sizeof (BITMAPCOREHEADER))
        return (WORD) (((BITMAPCOREHEADER huge *) lpMbid)->bcWidth);
    else
        return (WORD) (((BITMAPINFOHEADER huge *) lpMbid)->biWidth);
}

WORD AlturaMbid (BYTE huge * lpMbid)
{
    if (TamCabecalhoInfoMbid (lpMbid) == sizeof (BITMAPCOREHEADER))
        return (WORD) (((BITMAPCOREHEADER huge *) lpMbid)->bcHeight);
    else
        return (WORD) (((BITMAPINFOHEADER huge *) lpMbid)->biHeight);
}
```

```

}

/*****
* Função que extrai da estrutura BITMAPINFOHEADER o valor do
* número de cores (ou número de níveis de cinza) do BITMAP
*****/
WORD NumCoresInfoMbid (BYTE huge * lpMbids)
{
    if (((BITMAPINFOHEADER huge *) lpMbids)->biClrUsed != 0)
        return((WORD) ((BITMAPINFOHEADER huge *) lpMbids)->biClrUsed);
    else
        return (WORD) pow(2,(((BITMAPINFOHEADER huge *)
        lpMbids)->biBitCount));
}

BYTE huge * PegaEnderecoMbid (BYTE huge * lpMbid)
{
    DWORD dwNumCores, dwTamTabelaCores;
    WORD wContaBit;

    if (TamCabecalhoInfoMbid (lpMbid) == sizeof (BITMAPCOREHEADER))
    {
        wContaBit = ((BITMAPCOREHEADER huge *) lpMbid)->bcBitCount;

        if (wContaBit != 24)
            dwNumCores = 1L << wContaBit;
        else
            dwNumCores = 0;

        dwTamTabelaCores = dwNumCores * sizeof (RGBTRIPLE);
    }
    else
    {
        wContaBit = ((BITMAPINFOHEADER huge *) lpMbid)->biBitCount;

        if (TamCabecalhoInfoMbid (lpMbid) >= 36)
            dwNumCores = ((BITMAPINFOHEADER huge *) lpMbid)->biClrUsed;
        else
            dwNumCores = 0;

        if (dwNumCores == 0)

```

```

{
    if (wContaBit != 24)
        dwNumCores = 1L << wContaBit;
    else
        dwNumCores = 0;
}

dwTamTabelaCores = dwNumCores * sizeof (RGBQUAD);
}

return lpMbid + TamCabecalhoInfoMbid (lpMbid) + dwTamTabelaCores;
}

/*****
* Função que tem o formato da função printf do DOS, para cinco argumentos*
* de saída, e gera uma caixa de diálogo para impressão desses argumentos *
* tela
*****/
void CxMsgOk(char *szTitulo, char *szFormato, long tempo1, long tempo2,
             long tempo3, long tempo4, long tempo5)
{
    char szBuffer[256];
    char *pArgumentos;

    pArgumentos = (char *) &szFormato + sizeof szFormato;
    vsprintf(szBuffer, szFormato, pArgumentos);

    MessageBox(NULL, szBuffer, szTitulo, MB_OK);
}

/*****
* Função determina quantos zeros estão acrescentados na largura da
* imagem BITMAP para que essa largura tenha valor múltiplo de quatro *
*****/
short Resto(BYTE huge * lpMbid)
{
    short cxMbid, desloc;

    cxMbid = LarguraMbid (lpMbid);

```



```

desloc = cxMbid % 4;

if(desloc != 0) desloc = 4 - desloc;

return (short) desloc;

}

/*****
* Função que desloca os bits de uma imagem BITMAP de dezesseis cores *
* para que o ponteiro de imagem possua oito bits para cada pixel *
*****/
BYTE huge * BiDimPont16Cores(BYTE huge * lpMbid)
{
    BYTE huge *    matUso;
    BYTE huge *    bidbits;
    short          cxMbid, cyMbid, deslocamento;
    register int    i,j;
    DWORD          Tamanho;

    deslocamento = Resto(lpMbid);
    bidbits = PegaEnderecoMbid (lpMbid);
    cxMbid = LarguraMbid (lpMbid) + deslocamento;
    cyMbid = AlturaMbid (lpMbid);

    Tamanho = (long) (cxMbid) * cyMbid * sizeof(BYTE);

    matUso = (BYTE huge *)GlobalAllocPtr(GMEM_MOVEABLE,Tamanho);

    if(matUso==NULL) {
        MessageBox (NULL,"Nao consegui alocar memória para o ponteiro auxiliar",
                    szNomeAplic, MB_ICONEXCLAMATION | MB_OK);
        return NULL;
    }

    for(i=0;i<cyMbid;i++) {
        for(j=0;j<cxMbid/2;j++) {
            matUso[(long) i * (cxMbid) + 2 * j] =
                (bidbits[(long) i * (cxMbid)/2 + j] >> 4) & 0x0f;
            matUso[(long) i * (cxMbid) + 2 * j + 1] =
                bidbits[(long) i * (cxMbid)/2 + j] & 0x0f;

```

```

        }
    }

    return matUso;

}

// Le um MBID de um arquivo para a memoria
BYTE huge * LerDib (char * szNomeArq)
{
    BITMAPFILEHEADER bmfh;
    BYTE huge *    lpMbid;
    DWORD          dwTamanhoMbid, dwDesloc, dwTamCabecalho;
    int             hArquivo;
    WORD            wLerMbid;

    if (-1 == (hArquivo = _lopen (szNomeArq, OF_READ | OF_SHARE_DENY_WRITE)))
        return NULL;

    if (_lread (hArquivo, (LPSTR) &bmfh, sizeof (BITMAPFILEHEADER)) != sizeof
        (BITMAPFILEHEADER))
    {
        _lclose (hArquivo);
        return NULL;
    }

    if (bmfh.bfType != * (WORD *) "BM")
    {
        _lclose (hArquivo);
        return NULL;
    }

    dwTamanhoMbid = bmfh.bfSize - sizeof (BITMAPFILEHEADER);

    lpMbid = (BYTE huge *) GlobalAllocPtr (GMEM_MOVEABLE, dwTamanhoMbid);

    if (lpMbid == NULL)
    {
        _lclose (hArquivo);

```

```

    return NULL;
}

dwDesloc = 0;

while (dwTamanhoMbid > 0)
{
    wLerMbid = (WORD) MIN (32768ul, dwTamanhoMbid);
    if (wLerMbid != _lread (hArquivo, (LPSTR) (lpMbid + dwDesloc), wLerMbid))
    {
        _lclose (hArquivo);
        GlobalFreePtr (lpMbid);
        return NULL;
    }

    dwTamanhoMbid -= wLerMbid;
    dwDesloc += wLerMbid;
}

_lclose (hArquivo);

dwTamCabecalho = TamCabecalhoInfoMbid (lpMbid);
if (dwTamCabecalho < 12 || (dwTamCabecalho > 12 && dwTamCabecalho < 16))
{
    GlobalFreePtr (lpMbid);
    return NULL;
}

return lpMbid;
}

/*
* Função que realiza a contagem de numero de pontos de borda
*/
unsigned int FuncCont(BYTE huge * lpMbid)
{
    BYTE huge * bidbits;
    BYTE huge * pos;
    short cxMbid, novocxMbid, cyMbid, desloc;
    register unsigned int i,j;

```

```

    short faixa_raio;
    long Tamanho;
    long Comprimento;
    long zero;
    int contador=0;

    if (lpMbid == NULL) {
        MessageBox (NULL, "Abra um arquivo de imagem BMP",
            szNomeAplic, MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }

    else {

        desloc = Resto(lpMbid);
        bidbits = PegaEnderecoMbid (lpMbid);
        cxMbid = LarguraMbid (lpMbid)+ desloc;
        novocxMbid = LarguraMbid (lpMbid);
        cyMbid = AlturaMbid (lpMbid);

        if (NumCoresInfoMbid(lpMbid) > 234) {
            pos = bidbits;

            for(i=0; i<(cyMbid); i++){
                for(j=0; j<(novocxMbid); j++) {

                    if (pos[(long)((long)(i) * cxMbid) + (j)] < 100)

                        contador++;

                }
            }

            if (NumCoresInfoMbid(lpMbid)==16) {

                pos = BiDimPont16Cores(lpMbid);
                for(i=0; i<(cyMbid); i++){
                    for(j=0; j<(novocxMbid); j++) {

                        if (pos[(long)((long)(i) * cxMbid) + (j)] < 5)

```

```

        contador++ ;

    }
}

CxMsgOk(szNomeAplic, "As bordas desta imagem tem %d
pontos", (int)contador, NULL, NULL, NULL, NULL);

}
}

long FAR PASCAL _export ProcJan (HWND hjan, UINT mensa, UINT wParam,
IParam)
{
    static char    szNomeArq [_MAX_PATH],
        szTitulo  [_MAX_FNAME + _MAX_EXT];
    static char *  szFiltro[] = { "Arquivos Bitmap (*.BMP)", "*.bmp",
        "Arquivos MBIDs (*.DIB)", "*.dib",
        "" };
    static BYTE huge * lpMbid ;
    static OPENFILENAME ofn ;
    static short    cxCliente, cyCliente ;
    static WORD     wExibe = IDM_REAL ;
    HDC             hdc ;
    HMENU           hMenu ;
    BYTE huge *     lpMbidBits ;
    PAINTSTRUCT     ps ;
    short           cxMbid, cyMbid ;

    switch (mensa)
    {
        case WM_CREATE:
            ofn.lStructSize = sizeof (OPENFILENAME) ;
            ofn.hwndOwner = hjan ;
            ofn.lpstrFilter = szFiltro [0] ;
            ofn.lpstrFile = szNomeArq ;
            ofn.nMaxFile = _MAX_PATH ;
            ofn.lpstrFileTitle = szTitulo ;
            ofn.nMaxFileTitle = _MAX_FNAME + _MAX_EXT ;
            ofn.lpstrDefExt = "bmp" ;

```

LONG

```

        return 0 ;

    case WM_SIZE:
        cxCliente = LOWORD (lParam) ;
        cyCliente = HIWORD (lParam) ;
        return 0 ;

    case WM_COMMAND:
        hMenu = GetMenu (hjan) ;

        switch (wParam)
        {
            case IDM_ABRIR:
                if (GetOpenFileName (&ofn))
                {
                    if (lpMbid != NULL)
                    {
                        GlobalFreePtr (lpMbid) ;
                        lpMbid = NULL ;
                    }

                    lpMbid = LerDib (szNomeArq) ;

                    if (lpMbid == NULL)
                        MessageBox (hjan, szNomeAplic, "Nao consegui abrir o arquivo do MBID",
                            MB_ICONEXCLAMATION | MB_OK) ;

                    InvalidateRect (hjan, NULL, TRUE) ;
                }

                return 0 ;

            case IDM_CONTAR:
                FuncCont(lpMbid);

                return 0;

            case IDM_REAL:
            case IDM_EXPANDIR:
                CheckMenuItem (hMenu, wExibe, MF_UNCHECKED) ;
                wExibe = wParam ;
                CheckMenuItem (hMenu, wExibe, MF_CHECKED) ;

```



```

        InvalidateRect (hjan, NULL, TRUE);
        return 0;
    }
    break;

case WM_PAINT:
    hdc = BeginPaint (hjan, &ps);

    if (lpMbid != NULL)
    {
        lpMbidBits = PegaEnderecoMbid (lpMbid);
        cxMbid = LarguraMbid (lpMbid);
        cyMbid = AlturaMbid (lpMbid);

        SetStretchBltMode (hdc, COLORONCOLOR);

        if (wExibe == IDM_REAL)
            SetDIBitsToDevice (hdc, 0, 0, cxMbid, cyMbid, 0, 0,
                               0, cyMbid, (LPSTR) lpMbidBits,
                               (LPBITMAPINFO) lpMbid,
                               DIB_RGB_COLORS);

        else
            StretchDIBits (hdc, 0, 0, cxCliente, cyCliente,
                           0, 0, cxMbid, cyMbid,
                           (LPSTR) lpMbidBits,
                           (LPBITMAPINFO) lpMbid,
                           DIB_RGB_COLORS, SRCCOPY);
    }

    EndPaint (hjan, &ps);
    return 0;
case WM_DESTROY:
    if (lpMbid != NULL)
        GlobalFreePtr (lpMbid);

    PostQuitMessage (0);
    return 0;
}
return DefWindowProc (hjan, mensa, wParam, lParam);
}

```

Apêndice 9 - A Transformada de Hough para retas

Parametrização por coordenadas cartesianas

Se uma imagem é representada por pontos (x_i, y_i) onde $i = 1, 2, 3, \dots, n$, e n é o número de pontos da imagem, uma reta na imagem poderá ser representada pela equação 1.

$$y(x) = -ax + b \quad (01)$$

Onde a e b são ditos parâmetros desta função. Se representarmos o parâmetro b em função do parâmetro a , obteremos a equação 2.

$$b(a) = xa + y \quad (02)$$

Esta equação nos diz que para um determinado ponto (x, y) na imagem, existe uma reta no espaço de parâmetros, pois conhecendo os valores de x e y é possível obter valores lineares de b em função de a . Ou seja, uma reta no espaço de parâmetros corresponde a um ponto na imagem.

Por exemplo, se tivermos a reta da figura 1 dada pela equação 3.

$$y(x) = -1,3x + 8 \quad (03)$$

Seus parâmetros a e b são respectivamente 1,3 e 8 e correspondem a um ponto no espaço de parâmetros como ilustrado na figura 2.

Uma propriedade importante que se obtém desta inversão é que um ponto no espaço de parâmetros (a, b) irá corresponder a uma reta na imagem. Como pode ser percebido na figura 3, o cruzamento entre duas retas é um ponto, então o cruzamento entre duas retas no espaço de parâmetros corresponde a uma reta na imagem.

Fig. 1- Gráfico da reta na imagem.

Fig. 2 - Gráfico do ponto correspondente no espaço de parâmetros.

Ao se analisar uma imagem, cada ponto dela pode ser parametrizado, obtendo-se no espaço de parâmetros tantas retas quantas forem o número de pontos na imagem. Se um grande número de retas no espaço de parâmetros se interceptam num mesmo ponto como ilustrado na figura 4, existe uma forte evidência de uma reta na imagem, pois, neste caso, existe um grande número de pontos na imagem que fazem parte da reta que tem os parâmetros a e b .

Fig. 3 - Gráfico da interseção entre duas retas no espaço de parâmetros.

Fig. 4 - Gráfico da interseção entre muitas retas no espaço de parâmetros indicando a presença de uma reta na imagem com parâmetros a e b respectivamente iguais a 1,3 e 8.

Parametrização por coordenadas polares

Usando as coordenadas cartesianas podem-se obter retas no espaço de parâmetros, uma para cada ponto da imagem. A intersecção dessas retas indica que muitos desses pontos estão contidos numa mesma equação de reta na imagem com os parâmetros a e b interceptados pelas retas no espaço de parâmetros.

Considerando agora o ângulo φ ilustrado na figura 5, entre uma reta na imagem e o eixo das abcissas. Como o ponto de cruzamento da reta com o eixo das

ordenadas é obtido igualando-se o valor de x a 0(zero), e o ponto de cruzamento da reta com o eixo das abcissas é obtido igualando-se o valor de y a 0(zero), teremos x da equação 4 obtido a partir da equação 1.

$$y(x) = -ax + b$$

$$\text{se } x = 0,$$

$$y(x) = b$$

$$\text{e se } y(x) = 0,$$

$$-ax + b = 0$$

$$x = \frac{b}{a} \quad (04)$$

Fig. 5 - Gráfico da reta. A reta faz um ângulo φ com o eixo das abcissas.

Então a tangente do ângulo que a reta faz com o eixo das abcissas será dado pela equação 5 ou 6.

$$\operatorname{tg} \varphi = \frac{b}{\left(\frac{b}{a}\right)} \quad (05)$$

$$\operatorname{tg} \varphi = a \quad (06)$$

E a equação da reta poderá ser escrita na forma da equação 7.

$$y(x) = -x \operatorname{tg} \varphi + b \quad (07)$$

Quando a reta for paralela ao eixo das ordenadas, o ângulo φ é igual a n vezes 90° sendo n um número inteiro ímpar, sua tangente terá valor no infinito, portanto utilizando as coordenadas retangulares, não é possível parametrizar retas paralelas ao eixo das ordenadas. Isto porém pode ser resolvido usando-se coordenadas polares.

Passando a equação $y(x) = -ax + b$ para as coordenadas polares,

Fig. 6 - Gráfico da reta com o ângulo θ e amplitude ρ representados.

verifica-se primeiramente a identidade dos ângulos. Da figura 6, a soma dos ângulos internos de um triângulo é igual a 180° ($\theta + \varphi + 90^\circ = 180^\circ$). Então, a soma $\theta + \varphi = 90^\circ$. Como a soma $\theta + \phi = 90^\circ$, conclui-se então que $\varphi = \phi$ que é o complementar de θ , nesse caso

$$\operatorname{sen} \varphi = \cos \theta \quad e$$

$$\cos \varphi = \operatorname{sen} \theta$$

desta maneira o seno e o cosseno de φ serão dados pelas equações 8 e 9 respectivamente.

$$\operatorname{sen} \varphi = \cos \theta = \frac{\rho}{\left(\frac{b}{a}\right)} = \frac{a\rho}{b} \quad e \quad (08)$$

$$\cos \varphi = \operatorname{sen} \theta = \frac{\rho}{b} \quad (09)$$

O valor de b , é calculado, então, pela equação 10.

$$b = \frac{\rho}{\operatorname{sen} \theta} \quad (10)$$

Então a é obtido pela equação 11.

$$\begin{aligned} \cos \theta &= \frac{\rho}{b} \cdot a = \frac{\rho}{\left(\frac{\rho}{\operatorname{sen} \theta}\right)} \cdot a = a \operatorname{sen} \theta \quad e \\ a &= \frac{\cos \theta}{\operatorname{sen} \theta} \end{aligned} \quad (11)$$

Substituindo-se a e b na equação da reta obtemos a equação 12.

$$y(x) = - \left(\frac{\cos \theta}{\operatorname{sen} \theta} \right) x + \frac{\rho}{\operatorname{sen} \theta} \quad (12)$$

Trocando a posição de $\sin\theta$ e fazendo ρ em função de θ , têm-se como resultado a equação 13.

$$\rho(\theta) = x \cos\theta + y \sin\theta \quad (13)$$

Que é a equação paramétrica da reta em coordenadas polares.

Esta equação pode ser usada para qualquer reta na imagem, pois sempre há um valor finito para $\rho(\theta)$.

Dado um ponto (x,y) na imagem, pode-se obter o gráfico de $\rho(\theta)$ no espaço de parâmetros atribuindo valores para θ . Como este gráfico é cossenoidal, para que não haja repetição do valor de $\rho(\theta)$, limita-se θ no intervalo de $[0^\circ, 180^\circ]$ ou de $[-90^\circ, 90^\circ]$.

Portanto são observadas as seguintes propriedades entre o plano da imagem e o paramétrico.

- Um ponto no plano da figura corresponde a uma curva senoidal no plano paramétrico.
- Um ponto no plano paramétrico corresponde a uma reta no plano da imagem.
- Pontos de uma mesma reta no plano da imagem corresponde a um ponto de intercessão de curvas senoidais no espaço paramétrico.
- Pontos de uma mesma curva no plano paramétrico corresponde a um ponto no plano da imagem.

Tomando como exemplo a equação da reta $y(x) = -1,3x + 8$ pode-se determinar os seguintes cinco pontos: $(0,44, 7,52)$, $(1,37, 6,19)$, $(3,15, 3,19)$, $(4,58, 2)$, $(5,85, 0,28)$. Analisando estes pontos no espaço de parâmetros usando a equação paramétrica polar 13. Cada um destes pontos irá gerar uma equação paramétrica polar e de cada uma dessas equações será obtido um gráfico cossenoidal no espaço de parâmetros.

Fig. 7 - Gráfico das cossenoides correspondentes aos pontos (x_i, y_i) .

Na figura 7 pode-se notar a intersecção entre as curvas que acontece no ponto (ρ, θ) igual a $(4,92, 37,4)$. Como os parâmetros do ponto de intersecção correspondem aos parâmetros da equação da reta na imagem, podemos aplicar a equação 13

$$\rho(\theta) = x \cos \theta + y \sin \theta$$

e obter a equação 14.

$$y = \frac{(\rho - x \cos \theta)}{\sin \theta} \quad (14)$$

aplicando os valores de ρ e θ do ponto de intersecção e obtendo y em função de x , fica

$$y(x) = \frac{(4,92 - x \cos 37,4)}{\sin 37,4}$$

assim teremos

$$y(x) = -1,3x + 8,1$$

que é aproximadamente a equação da reta de onde foram retirados os 5(cinco) pontos do exemplo.

O exemplo acima nos mostra que podemos, através do método de Hough, recuperar a equação de uma reta na imagem analisando apenas os pontos da imagem. É interessante notar que retas de maior comprimento irão contribuir com um número maior de pontos e serão mais facilmente detectadas, pois a determinação da existência de uma reta na imagem se dá pela contagem do número de curvas no espaço de parâmetros que se interceptam num determinado ponto (ρ, θ) .