UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS
DEPARTAMENTO DE FÍSICA E INFORMÁTICA

**METODOLOGIA PARA A DETECÇÃO DO
MOVIMENTO UTILIZANDO A TÉCNICA DO
FLUXO ÓTICO.**

OK

Miguel A. Seminario Patiño

Tese apresentada ao Instituto de
Física de São Carlos - USP para a
obtenção do Título de Doutor em
Ciências (Física Aplicada).

Orientador: Prof. Dr. Valentin Obac Roda.

São Carlos

1997

IFSC-USP SERVIÇO DE BIBLIOTECA E
INFORMAÇÃO

Seminario Patiño, Miguel A.

**Metodologia para a Detecção do Movimento Utilizando a Técnica do Fluxo Ótico / Miguel A. Seminario Patiño - São Carlos 1997.
187p.**

Tese (Doutorado) - Instituto de Física de São Carlos, 1997.

Orientador: Prof. Dr. Valentin Obac Roda.

**1. Fluxo Ótico, 2. Velocidade Ótica, 3 Movimento, 4. Visão Com-
tacional, 5. Brilho da Imagem. I. Título.**

MEMBROS DA COMISSÃO JULGADORA DA TESE DE DOUTORADO DE **MIGUEL ALFONSO SEMINÁRIO PATINO** APRESENTADA AO INSTITUTO DE FÍSICA DE SÃO CARLOS, UNIVERSIDADE DE SÃO PAULO, EM 31/03/1997.

COMISSÃO JULGADORA:

Prof. Dr. Valentin Obac Roda (IFSC-USP)

Prof. Dr. Luciano da Fontoura Costa (IFSC-USP)

Prof. Dr. Roland Koberle (IFSC-USP)

Prof. Dr. Adilson Gonzaga (EESC-USP)

Dr. Paulo Estevão Cruvinel (EMBRAPA)

"Entendendo o passado se conquista o futuro"

Seminario

À minha mãe Ofelia.

Aos meus irmãos Moises,
Gerardo, Rigoberto,
Ofelia e Francisco.

À minha companheira Vilma

AGRADECIMENTOS

- Ao Professor Dr. Valentin Obac Roda, meu orientador, pelo apoio acadêmico e compreensão
- Ao CNPq pela bolsa de estudos que permitiu a realização do trabalho.
- Ao Instituto de Física de São Carlos pela oportunidade e condições para realizar o doutoramento.
- À EMBRAPA São Carlos por ter gentilmente cedido as imagens de percolação utilizadas no trabalho..
- Aos colegas deste campus universitário que de uma ou outra forma contribuíram para o desenvolvimento do presente trabalho. De forma especial a Rafael S, Ali F. T., Paulo A. M. M., Regina C., Marcelo F., Ivan G., Patrícia M., Luiz Antônio B., Virgílio A., Celso F, Adolfo P., e Galo S.
- Ao Valdir, Bruno e Cláudio pelo apoio técnico.
- Aos meus familiares pelo apoio e compreensão.
- À minha companheira Vilma pelo seu carinho.

RESUMO

O presente trabalho foi motivado pela grande importância que tem o movimento na percepção do espaço. Essa função é comum nos sistemas de visão natural já que a sobrevivência de algumas espécies é função desse fenômeno físico. Obviamente, a detecção do movimento no sistema de visão humana é de vital importância.

Visão computacional é uma área de pesquisa de forte interesse atual. Em visão computacional o movimento nas imagens é recuperado a partir de uma sequência de imagens. Neste campo a análise do movimento dos objetos pelo processamento de sequência de imagens tem tido aplicações em diversas áreas, desde o monitoramento de tráfego veicular até nas áreas biomédica e militar. Também, uma sequência de imagens pode proporcionar informação das estruturas e comportamento dos objetos que pertencem à cena do mundo 3D.

O objetivo do presente trabalho foi desenvolver uma metodologia computacional aplicada à detecção do movimento utilizando a técnica do Fluxo Ótico. Uma das características do Fluxo Ótico é que o campo de velocidades dos objetos que se movem na imagem pode ser estimado utilizando a informação local das mudanças do brilho da imagem. Aplicou-se a consideração de que só ocorre mudança de brilho nas regiões da imagem onde ocorreu movimento e que brilho varia suavemente em quase toda a imagem.

Realizou-se uma implementação computacional com a qual a estimativa do Fluxo Ótico é bem sucedida para uma sequência de imagens compatíveis de entrada. Executou-se uma operação de pre-processamento para filtrar e determinar as bordas das imagens. Utilizou-se operadores derivativos para calcular as derivadas numéricas que permitem determinar a direção e a intensidade das velocidades. Aplicou-se a técnica de relaxação as velocidades calculadas para obter uma melhor aproximação. Como consequência da metodologia obteve-se resultados de boa qualidade na forma de diagramas vetoriais indicativos do movimento (Diagramas de Agulha) que permitem por inspeção visual a detecção do movimento. O algoritmo é robusto no que refere-se à manipulação das imagens, dados e cálculos.

Para mostrar o bom desempenho da técnica apresenta-se os resultados de uma série de experiências que correspondem a diferentes formas de movimento: translação, rotação, translação-rotação, divergência, utilizando diversos tipos de cenas sintéticas e reais.

ABSTRACT

The motivation for the present work was the great importance of the movement in space perception. Movement detection is a normal function in natural vision systems, the survival of some animal species depends on it. In human vision movement detection is an essential feature.

Computer vision is a strong contemporary research area. In computer vision systems movement is recovered from a sequence of image frames. Motion analysis using frame sequences has important applications, going from vehicular traffic monitoring to biomedical and military applications. A sequence of frame images can also supply information about the structure and behavior of objects belonging to the 3D world.

The purpose of the present work was to develop a computer methodology for motion analysis using the optical flow methodology. The optical flow technique estimates the velocity field of moving objects in a sequence of images considering the local information of brightness change. It has been considered that the brightness changes occurs only because of the moving objects and is smooth over all the image.

The computer methodology developed for the optical flow estimation performed reasonably well for a compatible sequence of image frames. A pre-processing was used to filter and extract the edges of the images. Derivative operators were used to find the numeric derivatives of the brightness which allow to determine the value and the direction of the velocities. A relaxation technique was used to provide a better approximation of the velocities. As a result of the developed methodology we obtained from a sequence of two images, vector diagrams (needle diagrams) that allow by visual inspection the detection of motion. The algorithm is robust regarding to the image manipulation, data and numeric calculations.

To show the good performance of the technique we present the results of some experiments with different motion types: translation, rotation, translation and rotation and divergence, using artificial and real images.

ÍNDICE

CAPITULO 1	1
INTRODUÇÃO	1
1.1.- Cenas Dinâmicas: uma Revisão	1
1.1.1.- Objetivos	1
1.1.2.- Histórico	1
1.1.3.- Aplicações	2
1.2.- Desenvolvimento dos Estudos	3
1.2.1.- Método baseados na posição de imagens captadas de diferentes perspectivas	4
1.2.2.- Métodos Baseados na Técnica do Fluxo Ótico	5
1.3.- Percepção do movimento	9
1.4.- Objetivos do trabalho	10
1.5.- Estrutura do trabalho	11
1.6.- Conclusões	11
 CAPITULO 2	 14
DETERMINAÇÃO DO FLUXO ÓTICO	14
2.1.- Introdução	14
2.2.- Brilho da Imagem	16
2.3.- Velocidade Ótica	19

2.4.- Fluxo Ótico	22
2.5.- Estimativa do Fluxo Ótico	24
2.6.- Conclusões	26
 CAPITULO 3	 31
PROGRAMA PARA A DETERMINAÇÃO DO FLUXO ÓTICO ..	31
3.1.- Introdução	31
3.2.- Processos	33
3.2.1.- Fases de Inicialização	33
3.2.2.- Fase de Pré-processamento	42
3.2.2.1.- Determinação das Máscaras	48
3.2.2.2.- Operação Convolução e a DOG	49
3.2.3.- Fase da Estimativa do Fluxo Ótico	51
3.2.3.1.- Determinação do melhor casamento entre pixels	51
3.2.3.2.- Operação Cálculo da Raiz Mínima Quadrada	51
3.2.3.3.- Determinação das Velocidades	58
3.2.3.4.- Operação Relaxação	61
3.2.4.- Fases de Finalização	62
3.3.- Conclusões	65
 CAPITULO 4	 69
RESULTADOS EXPERIMENTAIS E ANÁLISE	69
 CAPITULO 5	 82
CONCLUSÕES E SUGESTÕES DE TRABALHOS FUTUROS ..	82
5.1.- Conclusões	82
5.2.- Sugestões para trabalhos futuros	84
 BIBLIOGRAFIA	 85

APÊNDICE A

Característica da *Work Station* 94

APÊNDICE B

Programa para Estimar o Fluxo Ótico 95

CAPITULO 1

INTRODUÇÃO

1.1.- Cenas Dinâmicas: uma Revisão

1.1.1.- Objetivos

O objetivo do presente capítulo é apresentar uma revisão da literatura na área de padrões visuais dinâmicos com ênfase na técnica do Fluxo Ótico. Discute-se algumas idéias fundamentais usadas por especialistas. Considera-se também, as correntes de abordagem na análise de cenas dinâmicas. As duas correntes fundamentais, segundo Young e Fu [71], são os métodos baseados na posição de imagens captadas de diferentes perspectivas, e os métodos baseados na técnica do Fluxo Ótico. De qualquer forma, o motivo é delinear um critério geral sobre as formas de desenvolvimento das metodologias e técnicas na área de Reconhecimento de Padrões Visuais Dinâmicos dentro do contexto de Visão Computacional. Esta breve revisão tenta fazer também uma avaliação do conhecimento existente na área de cenas dinâmicas. Também, esta pesquisa bibliográfica tenta identificar e avaliar os trabalhos existentes com o fim de saber e/ou identificar o que já foi feito.

1.1.2.- Histórico

Segundo Martin e Aggarwal [37], o sistema de visão humana tem capacidade para captar e transmitir informação complexa de forma eficiente. Marr [36] e Ullman [67] asseveram que nos sistemas biológicos o olho não é um simples transdutor, já que admite estímulos visuais

e transmite estímulos neuronais codificados. Ballard e Brown [08] explicam que o processo de transferência de informação envolve a correlação e integração tanto das características temporais como as espaciais. Marr [36] e Horn [28] argumentam que a Percepção Visual Humana serve como base para determinar as características desejáveis na implementação de Sistemas de Visão Artificial.

A extração da informação das cenas dinâmicas é um grande desafio para os pesquisadores na implementação de sistemas de visão artificial. Segundo as contribuições da área pode-se perceber o interesse dos cientistas, fundamentalmente em desenvolver sistemas para detectar, representar, armazenar e manipular as características espaciais de uma cena. Deve-se observar, também, que os métodos de análise dos eventos consideram só uma cena estática para levantar as partes de uma imagem e a sua estrutura. Em muitos casos o problema de analisar as características temporais do estímulo visual tem sido colocado de lado em estudos posteriores, isto principalmente por causas das limitações tecnológicas. O processamento de dados em sistemas biológicos é desenvolvido e executado de uma maneira complexa, poder-se-ia dizer de forma adaptativa. A percepção do movimento é extremamente importante para a sobrevivência de algumas espécies animais.

1.1.3.- Aplicações

A análise do movimento através de seqüência de imagens, ou coleções de quadros de imagens, é aplicado a diversas áreas do saber, como por exemplo: medicina, navegação autômata, tomografia, comunicações e TV, dança e coreografia, meteorologia e animação (Aggarwal [05], Aggarwal e Badler [06] e Snyder [58]). Deve-se observar que a motivação para esse interesse torna-se evidente se alguém examina em detalhe qualquer dessas contribuições. Fazer um análise 3D em diversas aplicações é necessário. Usa-se este tipo de análise para localizar e rastrear objetos móveis nas tarefas de vigilância, guiamento e monitoramento de tráfego veicular. Assim, diversos exemplos são indicativos pelo interesse pelo movimento. As contribuições podem ser encontradas, já como imagens que variam no tempo, ou como análise de cenas dinâmicas; Aggarwal e Duda [07], Doorn e Koenderink [19], Jain [30], Mitiche [40][41], Nagel [43], Tsotsos et.al. [66] e Webb e Aggarwal [69].

1.2.- Desenvolvimento dos Estudos

Costuma-se classificar os trabalhos desenvolvidos para o análise computacional das cenas em análise de cenas estáticas e análise de cenas dinâmicas.

Em muitos problemas básicos, considera-se que a visão dinâmica pode ser melhor entendida como uma seqüência de quadros de imagem estáticos. Desta forma, a análise de cenas dinâmicas é baseada na análise de cenas estáticas. Segundo Ballard e Brown [08], esta forma de abordagem é uma maneira discreta de análise, onde o período de amostragem deve ser suficientemente pequeno para que não haja perda de informação. Em outras palavras, a análise tem que ser muito rápida para que os resultados sejam considerados como seqüências em tempo real. Não deve ser entendido que a análise estática é fundamental, senão que a corrente dos trabalhos sob essa forma de análise é mais simples comparado com a análise de cenas dinâmicas uma vez que a análise de cenas em movimento é muito complexa e difícil de resolver.

Igualmente, pode-se conceituar a análise de imagens que variam no tempo como o processamento de uma seqüência de imagens ordenadas no tempo. Nas contribuições científicas se considera a captação da informação, considerando uma seqüência de quadros de imagens do evento, dado que a informação requerida não pode ser obtida partindo apenas de uma imagem.

Young e Fu [71], apresentam as principais linhas de pesquisa sobre a análise de cenas dinâmicas, como segue:

a) segmentação de cenas dinâmicas: consiste em dividir uma imagem em regiões fixas e não fixas, ou seja, encontrar regiões que estejam em movimento em cada elemento da seqüência de imagens.

b) detecção e análise de oclusão: oclusão ocorre quando os objetos se obstruem, e interferem na Percepção Visual, dos mesmos, em uma cena. Uma das metas da análise da oclusão é a determinação dos planos ou camadas de objetos na sua vizinhança.

c) extração de informações 3D partindo de configurações 2D: refere-se as medidas relativas das distâncias e movimento tridimensionais das imagens no espaço.

Considera-se como trabalhos precursores no estudo de imagens 3D, os estudos de imagens de camadas de nuvens em movimento captadas pelos satélites, Adiv [03][04]. Estes estudos foram motivados pelo desejo de analisar o movimento 2D. Alguns pesquisadores

consideram modelos abstratos de movimento 2D, usando polígonos e/ou figuras curvilíneas, Aggarwal e Duda [07]. O estudo dos eventos que possuem estruturas 3D é de grande interesse na atualidade, mas a obtenção dos parâmetros e sua forma 3D é bastante complexa.

Considerando o desenvolvimento das contribuições científicas na área; a análise de cenas dinâmicas pode ser abordada de duas formas:

- i) Métodos baseados na posição de imagens captadas de diferentes perspectivas, e
- ii) Métodos baseado na técnica do Fluxo Ótico.

1.2.1.- Método baseados na posição de imagens captadas de diferentes perspectivas.

A estimativa da posição de um objeto e o deslocamento do observador são questões a resolver quando se muda de perspectiva. O objetivo é observar os pontos relevantes do objeto partindo de duas imagens do mesmo objeto. Na atualidade o problema é de grande interesse., por essa razão existem inúmeras contribuições que buscam resolver o problema.

Roach e Aggarwal [56], estudaram o problema da determinação do movimento, da estrutura 3D partindo de uma seqüência de imagens 2D. Modelaram matematicamente o evento com sistemas de equações não lineares. A solução para este caso específico, consiste em resolver o conjunto de equações, usando o método dos mínimos quadrados. O sistema desenvolvido é muito sensível, o que implica grande incidência do ruído. Usaram-se algoritmos iterativos para resolver numericamente o sistema de equações impondo condições iniciais.

Tsai e Huang [61], apresentaram um método para estimar os parâmetros do movimento 3D de um corpo plano rígido, considerando duas perspectivas seqüenciais no tempo, ou quadros de imagem. Consideraram, também, um conjunto de oito parâmetros que podem ser determinados resolvendo o conjunto de equações lineares. O método proposto requer uma solução polinomial de sexta ordem de uma variável. Tsai e Huang [62] tiveram a necessidade de determinar quatro pontos em três posições, entretanto, Tsai e Huang [63][65], mostraram a estimação 3D do movimento de uma trajetória plana rígida, contudo, os parâmetros são determinados, unicamente, partindo de quatro pontos correspondentes as imagens. No artigo, também, realizaram uma outra análise considerando três observações, e provaram que dados quatro pontos, correspondentes à perspectiva de três observações, a solução é única.

Fang e Huang [23], estudaram o teorema da singularidade para dar solução ao conjunto de equações não lineares obtidas em trabalhos anteriores. A análise realizada é orientada para determinar os parâmetros do movimento 3D, quando se tem um ângulo de rotação pequeno. Também, apresentaram métodos para resolver o conjunto de equações não lineares aplicando o Critério de Newton Modificado, e o Método de Levenberg-Marquardt, para resolver o problema dos mínimos quadrados não lineares.

Yen e Huang [70] estudaram a forma como determinar uma estrutura, e o movimento 3D de um corpo rígido considerando uma seqüência de três quadros de imagem, e usando linhas retas. Entretanto, Longuet-Higgins [34] estudou o caso de objetos que possuem superfícies curvas, e mostrou uma técnica baseada no uso de relações de projeção para quatro pontos no espaço.

Meiri [38], pesquisou a reconstrução do movimento 3D de um objeto, partindo de imagens. Também, fornece as equações que descrevem a relação entre o número requerido de pontos observados no objeto, e o número de quadros de imagem. Discute-se a escala, e a invariância do procedimento.

Deve-se observar, que as propostas matemáticas descritas conduzem, geralmente, a um sistema de equações complicadas, de difícil solução; além disso, assume-se diversas considerações e um grande número de pontos.

1.2.2.- Métodos Baseados na Técnica do Fluxo Ótico

Existem diversas formas para interpretar o Campo do Fluxo Ótico. A maioria destas formas assumem que a informação está contida no campo, e que os modelos que detectam o movimento são restritos a corpos rígidos. Os trabalhos desenvolvidos têm tomado conta que a cena é constituída de um objeto em movimento, e que o observador é estacionário. Deve-se observar que o meio ou fundo é estacionário, ver Bruss e Horn [14], Lawton [32], e Tsai e Huang [61]. Neumann [47], pondera em seu estudo, cenas com múltiplas objetos em movimento, e investiga formas ou técnicas para decompor o Fluxo Ótico do conjunto correspondente. Assim, estes autores introduziram, também, uma metodologia para recuperar a estrutura 3D dos objetos.

Fang e Huang [21][22], Nagel [42], Nagel e Neumann [44], mostram um conjunto de equações não lineares com parâmetros do movimento desconhecidos. As formas propostas para resolver estas equações são métodos iterativos com soluções numéricas e condições iniciais. Mostra-se, também, que os resultados podem ser melhorados usando um conjunto de vetores de fluxo, aumentando o tamanho do campo que contem os vetores. Os pesquisadores assumem que os objetos se movimentam de maneira independente. De interesse é a obtenção dos resultados quando se trabalha com uma seqüência de imagens completa.

Longuet-Higgins [34], e Tsai e Huang [64], desenvolveram uma técnica computacional simples baseada na resolução do sistema de equações lineares. Formularam, também, condições de singularidade das soluções. Potter [50], reporta as dificuldades devido ao ruído.

Como se pode observar, os algoritmos desenvolvidos são sensíveis ao ruído. Uma das causas da sensibilidade ao ruído é devido à escolha dos critérios de otimização. Os trabalhos de Roach e Aggarwal [56] e Bruss e Horn [14] mostram as aproximações mais apropriadas para tratar o ruído.

Aplica-se o critério dos mínimos quadrados para minimizar a diferença entre os dados medidos e os valores correspondentes preestabelecidos da estrutura do movimento 3D. A aplicação do critério conduz a um sistema de equações não lineares. O conjunto de equações proposto é resolvido usando técnicas numéricas. Dessa forma calculam-se os parâmetros do movimento e os valores de profundidade. Porém, por causa do ruído, a recuperação da informação 3D a partir do fluxo de campo é instável.

Em alguns trabalhos se considera que o movimento é de translação pura, o que implica que todos os valores do fluxo estejam em direção, ou partindo de um ponto no plano de imagem. A determinação do ponto conhecido como Foco de Expansão. Poucas técnicas são baseadas nessa observação; ver Marr [36], e Prazdny [54].

Webb e Aggarwal [69] reportaram alguns resultados baseados em imagens reais. Não obstante permite-se o movimento restringido do sensor, e assume-se que o meio é constituído de

superfícies orientadas planas. Dessa forma, o algoritmo é baseado na procura do Foco de Expansão e das distâncias das superfícies das cenas.

Prazdny [53] descreve um método que se fundamenta na decomposição do campo de velocidades em componentes de translação e rotação. Calcula-se o Foco de Expansão, e o campo de translação que lhe correspondem, também, calcula-se o erro relativo baseado na componente rotacional. Dessa maneira, obtêm-se a função erro dos três parâmetros de rotação, além disso é determinada a resolução minimizando a função. A saber, Jain [30] reporta dificuldades na aplicação de critérios semelhantes a dados com ruído.

Riegger e Lawton [55] desenvolveram uma técnica simples e robusta para calcular os parâmetros do movimento em um meio estacionário com descontinuidades significativas na profundidade. Longuet-Higgins [34], desenvolveram um algoritmo baseado nas diferenças entre vetores, e as correspondentes descontinuidades no fluxo de campo, orientados a partir do Foco de Expansão do campo de translação. A proposta é função da precisão dos valores dos vetores do fluxo perto dos contornos oclusos.

Segundo a literatura pesquisada, a maioria dos métodos apresentados é baseada na análise local de objetos que se movimentam independentemente. Enfatizam, também que existe um único objeto, interpretação rígida, conforme com todos os vetores em um conjunto dado; entretanto, é assumida uma interpretação correta e os vetores são agrupados em seu conjunto. Por causa da sua natureza puramente local essa proposta é muito sensível a erros no fluxo de campo.

Segundo Ballard e Brown [08], a Transformação de Hough é um artifício matemático que pode ser utilizado para obter os parâmetros do movimento partindo do fluxo do campo. A Transformação de Hough é relativamente insensível ao ruído. Fennema e Thompson [24] fazem uso da Transformação de Hough em cenas que possuem objetos em movimento. Porém, a técnica de Hough tem um custo computacional elevado e baixa resolução, em outras palavras a técnica requer muita memória e elevado tempo de computação para calcular os parâmetros espaciais. Análise detalhada pode ser encontrado em: Adiv [01][02], Ballard e Sabbah [09], Coward et. al. [15], Duda e Hard [18], Grimson e Huttenlocher [27], Maitre [35] e O'Rourke [48].

Mitiche [39], Nakayama e Loomis [45], Lawton e Rieger [33], Prazdny [52][54], e Thompson e Barnard [60] utilizam a técnica do Fluxo Ótico com o objetivo de desenvolver estudos que expliquem o movimento, e estruturas dos objetos no espaço. Também, os trabalhos de Thompson [59] e Potter [50] mostram estudos fundamentados no Fluxo Ótico, e segmentam cenas de objetos em movimento. Paquin e Dubois [49] fazem uso do Fluxo Ótico para descrever as características das camadas do meio.

Em sua pesquisa Adiv [03], mostra uma nova aproximação para interpretar o campo do Fluxo Ótico. A proposta é baseada em dois níveis de imagens principais. No primeiro nível o Fluxo Ótico é dividido em segmentos concatenados de vetores de fluxo; aqui cada vetor é ligado com o movimento rígido de uma superfície plana áspera. Entretanto, no segundo nível os segmentos são agrupados sob a hipótese de que são para objetos simples em movimento. Adiv [04] estuda as cenas dinâmicas com o intuito de obter a estrutura, e o movimento 3D partindo do Fluxo Ótico. Apresenta, também, os problemas que podem acontecer, como é o caso da presença do ruído no campo do Fluxo Ótico. Mostra-se, ademais, a forma de como se pode estimar os parâmetros do movimento, e o problema das perturbações do meio quando existem objetos movendo-se independentemente.

Duncan e Chou [20], expõem um método para a detecção do movimento considerando uma seqüência de imagens. Demonstraram que a intensidade de cada pixel é relacionada com a segunda derivada temporal de uma função gaussiana. Indica-se, também, que as derivadas temporais e espaciais, da função, podem ser usadas para calcular a componente do Fluxo Ótico.

Por outro lado, Kearney et. al. [31], considera que as múltiplas observações de uma cena podem fornecer informação importante da estrutura, e comportamento dinâmico de um corpo 3D. No artigo se estuda a importância das diferentes técnicas para estimar o Fluxo Ótico e se enfatiza que o Fluxo Ótico é função dos gradientes de intensidade da imagem.

Schunck [57], em sua pesquisa, descreve que o Fluxo de Imagem é o campo de velocidade no plano de imagem causado pelos movimentos aparentes. Demonstra também, que podem ser usados alguns algoritmos para descrever a velocidade do Fluxo de Imagem, e apresentam resultados experimentais para casos de testes simulados e de imagens reais.

Segundo Verri e Poggio [68], o campo do movimento de um campo vetorial 2D é a projeção no plano de imagem do campo de velocidade 3D de uma cena em movimento. Consideram ao Fluxo Ótico como estimação do movimento do campo no qual pode ser observado, partindo da variação da primeira ordem dos modelos de intensidade de imagem. Também estudam as propriedades quantitativas do campo do movimento, as mesmas que podem ser obtidas partindo do Fluxo Ótico.

1.3.- Percepção do Movimento

Pode-se conceituar o movimento como o estado em que um corpo muda constantemente de posição em relação a um outro corpo ou ponto de referência. Existem basicamente dois tipos de movimento: o de translação e o de rotação, Gregory [26]. Os corpos em movimento representam eventos de diversos tipos, por exemplo, um objeto em movimento pode representar perigo iminente para um ser vivo sendo sua detecção vital para a sua sobrevivência, o mesmo objeto em repouso não representa contingência alguma.

A percepção do movimento é um processo fundamental ao homem. A percepção pode ser executada pelos sentidos, mas fundamentalmente pela visão. Nos órgãos da visão, os olhos, existem detetores especiais do movimento. Os receptores do movimento são os responsáveis de indicar ao cérebro a presença dos corpos que se movimentam em nosso entorno.

Com a visão, o homem é capaz de detectar múltiplas características físicas dos corpos como: a forma, a cor, a textura e o movimento. Essas características são detectadas em primeiro momento na superfície retiniana. Cada um dos detetores retinianos cumpre com determinadas funções, por exemplo os bastonetes são sensíveis aos níveis de cinza. Também, a superfície retiniana é particionada segundo a sua funcionalidade. Assim, na parte central da retina, área denominada de FOVEA, se encontra a maior quantidade de cones e bastonetes, enquanto que na borda acham-se as células orientadas à detecção do movimento.

Pôr outro lado, os extremos do Campo de Visão são percebidos pelas bordas da retina. Todo evento que se execute nos extremos do Campo de Visão é indicado através do processo atento e que merece ser atendível. Se o observador tem interesse sobre dito evento, girará o

olho orientando-o em direção ao local onde se está desenvolvendo a cena, com o intuito de captar mais informação, centrando a FOVEA na cena que lhe atraiu, tentando identificar o objeto.

O assinalamento do movimento pode ser percebido de duas formas, Gregory [26]. Primeiro, quando o olho está estático e o corpo se movimenta no Campo de Visão. Neste caso se forma uma imagem que percorre toda a superfície retiniana desde o aparecimento até o desaparecimento no Campo de Visão, e o assinalamento do movimento do corpo é originário das células que cumprem com aquela função. Segundo, quando o olho está em movimento e o corpo se desloca frente ao Campo de Visão. Neste caso o olho rastreia o objeto em todo o seu percurso e a imagem que se forma na superfície retiniana se mantém praticamente estática. O assinalamento do movimento não provem propriamente do objeto em movimento senão do fundo com sinais que exploram a retina durante seu percurso. Também, se assinala o movimento com os receptores-extensores dos músculos oculares. Portanto existem duas formas de percepção do movimento, uma proveniente da superfície retiniana e outra originaria dos músculos oculares quando os olhos se movimentam na cavidade ocular. Na detecção do movimento ambos os processos são cooperativos.

1.4.- Objetivos do trabalho

O principal objetivo do presente trabalho foi o desenvolvimento de uma metodologia na área de processamento de imagens, aplicada especificamente à detecção do movimento usando a técnica do Fluxo Ótico. Outro objetivo foi o de aplicar alguns critérios que relacionem os sistemas de visão natural e visão sintética, assim, aplicar os argumentos de Marr - Hildreth Marr [36], no que se refere à construção de máscaras para executar a convolução no processo de detecção de bordas. Foi também aplicada a raiz mínima quadrada na estratégia de casamento dos pixels de Anandan, Barron [11], com o objetivo de encontrar o melhor casamento entre pixels. Para o melhoramento dos resultados, aplicou-se a técnica de relaxamento de Horn-Schunck [29]. O programa desenvolvido foi aplicado a diversos tipos de eventos registrados através de uma sequência de dois quadros de imagens de entrada.

Com os argumentos e com as técnicas propostas, dispõe-se de recursos computacionais adequados é possível obter resultados claros e de boa qualidade na forma de um diagrama

vetorial de velocidades (diagrama de agulhas) a partir de duas imagens de entrada. Os diagramas de agulha permitem por inspeção visual detectar o movimento dos objetos da cena.

1.5.- Estrutura do trabalho

O trabalho é constituído de cinco capítulos e dois apêndices.

No Capítulo 1 apresenta-se as justificativas e uma revisão bibliográfica comenta

No Capítulo 2 desenvolve-se a determinação do Fluxo Ótico.

No Capítulo 3 explica-se o desenvolvimento do programa.

No Capítulo 4 apresenta-se os resultados obtidos na execução do programa aplicado a uma seqüência de duas imagens de entrada. Também, analisa-se os resultados dos ensaios.

O Capítulo 5 contem as conclusões do trabalho e as sugestões para dar-lhe continuidade.

No Apêndice A apresenta-se as características da máquina utilizada no trabalho.

No Apêndice B exhibe-se o programa para a detecção do movimento.

1.6.- Conclusões

Através do presente capítulo foi descrito um conjunto de contribuições pertencentes à área da Percepção Visual de cenas dinâmicas. Foram consideradas duas formas de abordagem do problema: a seqüência de imagens e o Fluxo Ótico.

No caso dos métodos para detetar o movimento através de seqüência de imagens, foram descritas as técnicas, e as restrições feitas pelos autores considerando que o mundo é composto por objetos rígidos que não mudam de forma com o tempo. Mostrou-se também que a otimização e critérios utilizados conduzem a bons resultados mas ainda persiste o problema do ruído.

No estudo das contribuições percebeu-se que os modelos matemáticos são um complexo sistemas de equações integro-diferenciais de difícil resolução. Também se observou o uso de critérios e métodos matemáticos, como por exemplo o método dos mínimos quadrados do erro para otimizar resultados.

Em alguns artigos são apresentados estudos onde a solução ao problema de cenas dinâmicas é baseado no princípio de conservação de distância, opção válida para corpos rígidos. Nos estudos também se observou a necessidade e/ou hipótese de ter um mínimo de pontos em imagens consecutivas e que a correspondência entre eles é conhecida. Também se detectou que existe uma técnica de abordagem que permite ao pesquisador determinar a posição relativa da perspectiva.

Por outro lado, nas contribuições que desenvolvem a técnica do Fluxo Ótico, usam-se as Velocidades Óticas com o intuito de resolver os problemas da posição dos objetos e/ou a s profundidade.

A técnica utilizada para resolver o problema do ruído e da oclusão é Transformada de Hough, mas esta técnica é de custo elevado já que requer muita memória e tempo de computação.

Apresentou-se a percepção do movimento. Mostrou-se que os sistemas de visão natural tem duas formas de assinalar o movimento. Uma forma com sinais originárias das retinas e uma outra forma com sinais provenientes dos músculos oculares. Ambas as formas se complementam na função de detectar o movimento.

Foram também apresentados diversos argumentos da área com o objetivo de entender a forma de desenvolvimento da técnica do Fluxo Ótico para a detecção do movimento a partir de uma seqüência de quadros de imagens de entrada. Também foi exposta a forma de detecção do movimento dos sistemas de visão natural e sobre a importância do movimento na vida real.

Levando em consideração o conjunto de argumentos mostrados nesta capítulo, o objetivo deste trabalho foi o de implementar um algoritmo computacional para detetar o movimento a partir de uma sequência de dois quadros de imagens.

CAPITULO 2

DETERMINAÇÃO DO FLUXO ÓTICO

2.1.- Introdução

O presente capítulo tem como objetivo apresentar alguns conceitos fundamentais para a determinação do Fluxo Ótico. Os conceitos abordados são o Brilho da Imagem e a Velocidade Ótica. Também é desenvolvido o equacionamento para a estimativa do Fluxo Ótico.

Nos eventos que acontecem no espaço 3D existem duas partes bem definidas: o observador e o objeto. Do relacionamento entre eles surgem três casos: observador estático - objeto em movimento, observador em movimento - objeto estático e observador e objeto em movimento. Nos casos citados o movimento é relativo.

O objeto pertence a uma cena de um evento que acontece no espaço 3D do mundo real. O observador é um sistema ótico que registra as cenas 3D em uma superfície de imagem denominada plano de imagem. Deve-se notar que o observador pode ser uma câmara, ou algum outro dispositivo, no qual se registrem as imagens correspondentes às cenas. As mudanças do brilho nas imagens podem ser usadas para detectar o movimento relativo e/ou formas dos objetos. De qualquer forma, muita informação pode ser extraída a partir das imagens que variam no tempo.

Campo de Velocidade ou **Campo de Movimento** é definido como sendo a projeção em perspectiva da velocidade de um objeto do espaço 3D sobre o plano de imagem, Horn [28], Horn e Schunck[29], Prazdny [54]. Os métodos propostos para estimar o campo de velocidade, métodos do gradiente e métodos da filtragem espaço-tempo estão centrados nas mudanças do brilho da imagem. O campo do fluxo que acontece pela mudança do brilho é referido como **Fluxo Ótico** ou **fluxo de imagem**, Horn e Schunck [29], Nagel [42], Verri e Poggio [68] e Young e Fu [71]. Visto que as velocidades são um conceito meramente geométrico e já que o Fluxo Ótico é estimado em base nas mudanças do brilho, então o Fluxo Ótico é igual ao campo de velocidades, Horn [28].

Considerando uma câmara e um objeto em um sistema de coordenadas cartesianas, de acordo com a figura 2.1, quando o objeto se movimenta dentro do campo de visão da câmara, ou quando a câmara se movimenta frente a um objeto fixo acontecem mudanças no brilho da imagem. As mudanças que ocorrerem no brilho da imagem podem ser usadas para recuperar o movimento relativo, e/ou as forma dos objetos.

Figura 2.1.- Observador (câmara) e objeto no espaço 3D.

Originalmente o Fluxo Ótico foi determinado como a solução de uma equação restrita, obtida considerando que o brilho da imagem é estacionário em cada um dos pontos da imagem, Horn e Schunk [29].

2.2.- Brilho da Imagem

O brilho é uma palavra chave em processamento de imagens e pode ter dois tipos de significado. Um deles trata do brilho da imagem e o outro do brilho da cena. Em Horn [28], se introduz o termo de iluminância como o equivalente do brilho da imagem e o termo luminância para brilho da cena.

Define-se iluminância como a potência por unidade de área de energia radiante que incide sobre uma superfície. Ver figura 2.2.

A iluminância é denominada como E , e é definida como:

$$E = \frac{\Delta Q}{\Delta A} \quad \text{Eq. 2.1}$$

onde: ΔQ é a potência de energia radiante que incide sobre a superfície infinitesimal ΔA .

Figura 2.2.- Iluminância: energia por unidade de área que incide sobre uma superfície.

A formação de uma imagem na superfície sensível câmera (plano de imagem) é função direta do brilho. Assim, o brilho em uma imagem é relacionado com o fluxo de energia emitida pelo conjunto de pontos ou pela superfície do objeto observado. Deve-se observar que diferentes pontos do objeto possuem diferentes brilhos. Essa diferença de brilho é função da iluminação e da forma como os objetos refletem a luz. Utiliza-se o termo luminância para identificar esse fenômeno físico. A luminância é definida como a potência por unidade de área emitida dentro do ângulo sólido de uma superfície. Ver figura 2.3.

A luminância é denominada de L , e é definida como:

$$L = \frac{\Delta^2 Q}{\Delta A \Delta \omega} \quad \text{Eq. 2.2}$$

onde: ΔQ é a potência emitida por uma superfície infinitesimal de área ΔA dentro de um ângulo sólido infinitesimal $\Delta \omega$.

Portanto, brilho da imagem pode ser considerado como a medida da iluminância da cena sobre o plano de imagem. Tendo a iluminância (brilho da imagem) e a luminância (brilho da cena) pode-se obter uma equação que as relacione. Para entender esse relacionamento veja-se a figura 2.4.

Figura 2.3.- Luminância: potência emitida por unidade de área dentro de um cone de direções que tem unidades de ângulo sólido.

Figura 2.4.- Geometria do sistema ótico

A figura 2.4 mostra a projeção em perspectiva, base para obter a expressão do brilho, assim:

$$E(x(t), y(t), t) = L(\rho(\mathbf{p}, t), \mathbf{N}_{sc}(\mathbf{P}), \mathbf{N}_{fl}(\mathbf{P})) \frac{\pi}{4} \left(\frac{D}{f} \right)^2 (\mathbf{P} \cdot \mathbf{z}^c)^4 / (\mathbf{P} \cdot \mathbf{P})^2 \quad \text{Eq. 2.3}$$

onde:

$\mathbf{P}$ é o vetor do ponto P à origem do sistema de coordenadas.

E é a iluminância (brilho da imagem),

L é a luminância (brilho da cena),

$\rho(\mathbf{p}, t)$ é o albedo, o qual descreve a variação espacial da textura da superfície pintada da cena observada,

t é o tempo,

$\mathbf{N}_{fl}(\mathbf{P})$ é o vetor unitário o qual identifica a fonte de luz (fl),

$\mathbf{N}_{sc}(\mathbf{P})$ é o vetor unitário normal à superfície da cena (sc),

D é o diâmetro da lente, e

$(\mathbf{P} \cdot \mathbf{z}^c)^4$ é o ângulo α entre o vetor $\mathbf{P}$ e o eixo z .

A luminância pode ser equacionada como:

$$L(\rho(\mathbf{p}, t), N_{sc}(\mathbf{P}), N_{fl}(\mathbf{P})) = \rho(\mathbf{p}, t) R(N_{sc}(\mathbf{P}), N_{fl}(\mathbf{P})) \quad \text{Eq. 2.4}$$

onde: $R(N_{sc}(\mathbf{P}), N_{fl}(\mathbf{P}))$ é a Reflectância.

Da Eq. 2.4, tem-se que a luminância é igual ao produto do albedo pela reflectância. Note-se que as variações da luminância considera as mudanças da orientação da superfície em função da iluminação da fonte de luz. Ambos os termos estão em função da coordenadas espaciais 3D do ponto $\mathbf{P}$.

Considerando a geometria do modelo apresentado, o sistema ótico pode ser calibrado. Nesse caso a irradiância pode ser considerada igual à radiância, assim:

$$E(x(t), y(t), t) = L(\rho(\mathbf{p}, t), N_{sc}(\mathbf{P}), N_{fl}(\mathbf{P})) \quad \text{Eq. 2.5}$$

Desse ponto de vista, considerando as grandezas pode-se definir a luminância L como a potência da energia irradiada pela superfície dos objetos de uma cena encerrada pelo ângulo sólido nessa direção. Finalmente a iluminância ou brilho da imagem E pode ser definida como a razão entre a potência recebida por um elemento de imagem e o seu tamanho. Portanto, o brilho da imagem não é uma propriedade pontual, é a densidade da energia usualmente expressada em watts/m^2 , Horn [28].

2.3.- Velocidade Ótica

Considerando que temos um sistema de visão no qual se registrem através de imagens cenas do mundo 3D. Existe mudança do brilho da imagem quando um objeto que pertence à cena se move dentro do plano de imagem da câmera. A mudança do brilho da imagem pode ser utilizada para detectar o movimento e/ou formas dos objetos.

Entende-se como velocidade óticas o conjunto de vetores de velocidade dos pontos projetados sobre a superfície do sistema de visão. A velocidade ótica é referida também como

campo do movimento, ou campo de velocidades, Horn [28], Horn e Schunck [29], Nagel [42], Prazdny [54], Verri e Poggio [68] e Young e Fu [71].

Primeiramente vamos examinar as equações fundamentais. Ver figura 2.5. O modelo apresentado é um sistema de visão com projeção central. Este modelo geométrico tem sido aplicado por ser adequado à maioria das câmaras comuns. Por simplicidade e sem perder a generalidade, o centro de projeção se encontra na origem do sistema de coordenadas cartesiano, $S : (O, X, Y, Z)$, e o plano de imagem é o sistema de coordenadas $s : (o, x, y)$.

O plano de imagem é centrado no eixo-Z e perpendicular a este, e seus eixos x e y são paralelos aos eixos X e Y , respectivamente.

Da figura 2.4, tem-se um ponto P no espaço 3D, definido pelo vetor:

$$P = (X, Y, Z) \quad \text{Eq. 2.6}$$

No plano de imagem existe um ponto p , imagem do ponto P , que pode ser definido como:

$$p = (x, y, z) \quad \text{Eq. 2.7}$$

Figura 2.5 Geometria da câmara e movimento aparente.

f é a distância focal e $'$ é a derivada.

Como o ponto $\mathbf{p}$ é imagem ponto $\mathbf{P}$, então a relação entre ambos os vetores é:

$$\mathbf{p} = \frac{f}{Z} \mathbf{P} \quad \text{Eq. 2.8}$$

Portanto, o campo de velocidade, ou seja, projeção do movimento 3D sobre o plano de imagem, é obtido derivando e executando manipulações algébricas da Eq. 2.8, assim:

$$\mathbf{p}' = \frac{f}{Z} \left[\mathbf{P}' - \mathbf{P} \left(\mathbf{P}' \cdot \mathbf{z}^c \right) \frac{1}{Z} \right] \quad \text{Eq. 2.9}$$

Em geral, o movimento 3D do ponto $\mathbf{P}$ pode ser modelado em função das componentes de translação e rotação, pelo que:

$$\mathbf{P}' = \mathbf{T} + \mathbf{W} \times \mathbf{P} \quad \text{Eq. 2.10}$$

onde:

$$\mathbf{T} = (T_x, T_y, T_z) \quad \text{Eq. 2.11}$$

é a componente translação, e

$$\mathbf{W} = (W_x, W_y, W_z) \quad \text{Eq. 2.12}$$

é a componente de rotação.

Deve-se observar que ambas as componentes, a de translação e a de rotação são componentes instantâneas. Também é considerado que o sistema de visão é estacionário e que o objeto se movimenta no campo de visão.

Substituindo as equações 2.6, 2.7, 2.10, 2.11, e 2.12 na equação. 2.9, obtem-se as duas equações escalares que relacionam as componentes p'_1 e p'_2 das velocidades óticas do ponto $\mathbf{p}$ no plano de imagem com as componentes do ponto $\mathbf{P}$ do espaço 3D, ponto em movimento, assim:

$$p'_1 = \frac{f}{Z} T_x - \frac{x}{Z} T_y + f W_y - y W_z - \frac{x}{f} (y W_x - x W_y) \quad \text{Eq. 2.13}$$

$$p'_2 = \frac{f}{Z} T_Y - \frac{x}{Z} T_X + xW_Z - fW_X - \frac{y}{f} (yW_X - xW_Y) \quad \text{Eq 2.14}$$

2.4.- Fluxo Ótico

Existe mudança do padrão do brilho em uma imagem quando os objetos que pertencem à cena se movem dentro do campo de visão. Entende-se como Fluxo Ótico como o movimento aparente do padrão do brilho, mas isso não sempre é válido, Horn [28]. O Fluxo Ótico, também é referido como Fluxo de Imagem, Prazdny [54] e Schunck [57].

Para esclarecer os conceitos de Fluxo Ótico (Fluxo de Imagem) e velocidade ótica (campo do movimento) consideremos que o objeto da cena, no espaço 3D é uma esfera uniforme. A esfera é colocada dentro do campo de visão da câmera e consideremos também que o espaço da cena tem iluminação constante. Ver figura 2.6.

Figura 2.6.- Esfera girando sobre seu eixo e sem translação, e fonte de luz estática.

Na montagem apresentada na figura 2.6, se observa que existe variação espacial do brilho já que o objeto da cena é curvo. Note-se que se forma uma sombra mas esta não se

movimenta na superfície, portanto as imagens captadas pelo sistema de visão não mudam no tempo. No evento o campo de velocidade é diferente de zero, enquanto que o Fluxo Ótico é zero. O acontecido no evento é devido as condições da montagem onde a iluminação é constante e o objeto, a esfera uniforme, só tem movimento de rotação e não de translação.

Em outro caso, ilustrado na figura 2.7, a sombra que se forma na superfície da esfera uniforme se desloca, portanto na superfície do sistema de visão se formam diferentes imagens em diferentes instantes de tempo. Neste caso o campo de velocidade é zero e o Fluxo Ótico é diferente de zero. O acontecido é devido a que a fonte de luz é a que se desloca, se bem que a esfera uniforme permanece estática, o que da como consequência a formação de diferentes imagens.

Figura 2.7.- Esfera uniforme estática, e fonte de luz se transladando.

Como se pode observar em certas condições de montagem o campo de velocidade e o Fluxo Ótico são iguais. Existem, na realidade outros casos para explicar este fenômeno como a ocorrência de imagens diferentes e sombras.

Portanto, note-se que só o Fluxo Ótico é acessível e que não é muito diferente ao campo de velocidade, exceto em condições especiais. De qualquer forma, essa diferença permite estimar o movimento relativo através das mudanças do brilho da imagem.

De qualquer forma no movimento aparente do padrão do não é trivial detectar ou relacionar os pixel que se correspondem entre ambas as imagens.

Da figura 2.8, tem-se um ponto P que pertence à primeira imagem; o brilho daquele ponto é E no instante t . Seja P' o ponto que lhe corresponde na segunda imagem no tempo $t + \delta t$. Existiu mudança do brilho nesse intervalo de tempo. A forma como o brilho varia não é previsível. Tipicamente existem pontos próximos a P que possuem o mesmo brilho. Consideremos que a mudança do brilho não tem descontinuidades, ou seja que varia de forma contínua na parte da imagem de interesse e que o ponto P se situa no contorno C . No tempo $t + \delta t$ existiria um contorno C' semelhante e um ponto P' . Mas, a incógnita é qual é a correspondência entre o ponto P do contorno C com o ponto P' do contorno C' . Essa relação é difícil de se estabelecer já que ambos os contornos não possuem a mesma forma. Note-se que o Fluxo Ótico não é determinado só pela informação local da mudança da imagem.

a) Imagem no tempo t .

b) Imagem no tempo $t + \delta t$

Figura 2.8.- Relação entre pixel P que se situa no contorno C e o pixel P' que se situa no contorno C' , de difícil solução

2.5.- Estimativa do Fluxo Ótico

Considerando-se que o observador é um sistema de visão artificial, por exemplo, uma câmara comum, e o objeto pertence à cena do mundo real. Na superfície de imagem do

sistema de visão se registra uma imagem correspondente ao evento que ocorreu no mundo real.

Suponhamos que se tem uma imagem qualquer. Na imagem vamos considerar uma pequena superfície, ou região, com brilho uniforme. De fato nessa pequena área pode-se obter um modelo vetorial e registrar as mudanças do brilho da imagem, a que corresponde ao evento observado, Ben-Tzvi et. al. [12], Ben-Tzvi e Sandler [13], Del Bimbo et. al. [16], Del Bimbo et. al. [17], Horn [28], Horn e Schunck [29], Marr [36], Nesi et.al. [46] e Prazdny [54].

Seja $E(x, y, t)$ a iluminância do ponto (x, y) no instante t em um segmento de imagem. Se $u(x, y)$, e $v(x, y)$, são as componentes das velocidades óticas, com componentes x e y do vetor do Fluxo Ótico nesse ponto, então espera-se que E seja igual no tempo $t + \delta t$, quando $\delta t \rightarrow 0$, no ponto $(x + \delta x, y + \delta y)$. Deste modo, tem-se:

$$\delta x = u \delta t \quad \text{Eq. 2.15}$$

$$\delta y = v \delta t \quad \text{Eq. 2.16}$$

pelo que:

$$E(x + u\delta t, y + v\delta t, t + \delta t) = E(x, y, t) \quad \text{Eq. 2.17}$$

onde: δt é um pequeno intervalo de tempo.

Para determinar u e v , não basta a restrição anterior, também temos que considerar que as velocidades óticas não são descontínuas.

Se o brilho da imagem varia suavemente em relação a x , y e t , então a Eq. 2.17, pode ser expandida. Aplicando a série de Taylor, obtendo-se:

$$E(x, y, t) + \delta x \frac{\partial}{\partial x} E + \delta y \frac{\partial}{\partial y} E + \delta t \frac{\partial}{\partial t} E + \varepsilon = E(x, y, t) \quad \text{Eq. 2.18}$$

onde ε contem os termos de mais alto ordem em δx , δy e δt .

Suprimindo $E(x, y, t)$, e dividindo por δt na Eq. 2.13, tem-se:

$$\frac{\delta x}{\delta t} \frac{\partial}{\partial x} E + \frac{\delta y}{\delta t} \frac{\partial}{\partial y} E + \frac{\partial}{\partial t} E + F(\varepsilon) = 0 \quad \text{Eq. 2.19}$$

No limite, quando $\delta t \rightarrow 0$, tem-se:

$$\frac{\partial}{\partial x} E \frac{dx}{dt} + \frac{\partial}{\partial y} E \frac{dy}{dt} + \frac{\partial}{\partial t} E = 0 \quad \text{Eq. 2.20}$$

A Eq. 2.20, é a expressão da equação onde o brilho da imagem é considerado estacionário com relação ao t , o qual é justo a expansão da equação:

$$\frac{d}{dt} E = 0 \quad \text{Eq. 2.21}$$

Conforme Eq. 2.20 e Eq. 2.21, pode-se escrever que:

$$\frac{d}{dt} E = \frac{\partial}{\partial x} E \frac{dx}{dt} + \frac{\partial}{\partial y} E \frac{dy}{dt} + \frac{\partial}{\partial t} E = 0 \quad \text{Eq. 2.22}$$

Fazendo:

$$\frac{dx}{dt} = u \quad \text{Eq. 2.23}$$

$$\frac{dy}{dt} = v \quad \text{Eq. 2.24}$$

$$\frac{\partial}{\partial x} E = E_x \quad \text{Eq. 2.25}$$

$$\frac{\partial}{\partial y} E = E_y \quad \text{Eq. 2.26}$$

$$\frac{\partial}{\partial t} E = E_t \quad \text{Eq. 2.27}$$

Da Eq. 2.23, Eq. 2.24, ... Eq. 2.27 na Eq. 2.22, tem-se:

$$E_x u + E_y v + E_t = 0 \quad \text{Eq. 2.28}$$

onde u e v representam as componentes do vetor de velocidade local V ao longo de x e y , respectivamente. E_x , E_y , e E_t são as derivadas respectivas e são determinadas a partir da imagem.

A Eq. 2.28, é conhecida como “Equação Restrita do Fluxo Ótico” (*Optical Flow Constraint Equation - OFCE*), visto que é expressa sobre as restrição das componentes de u e v do Fluxo Ótico. As soluções da equação são referidas como Fluxo Ótico, e representa somente uma aproximação do campo de velocidade p' .

A figura 2.9, ilustra a restrição do fluxo de velocidade local, resolvida pela Eq. 2.28. Os valores de (u, v) , que satisfazem a equação do Fluxo Ótico Restrito, caem na linha reta do Espaço de Velocidade. Todo o que se faz é identificar a Linha Restrita.

A Eq. 2.28, pode ser rescrita como:

$$(E_x, E_y) \bullet (u, v) = -E_t \quad \text{Eq. 2.29}$$

A Eq. 2.28, isolada, não fornece de forma total as componentes do Fluxo Ótico, Horn e Schunck [29]. De qualquer forma, a equação pode ser usada diretamente só para determinar a componente paralela do Fluxo Ótico através da gradiente do brilho da imagem ∇E , Ben-Tzvi et. al. [12]. Assim:

$$V = - \frac{E_t}{||\nabla E||} \frac{\nabla E}{||\nabla E||} \quad \text{Eq. 2.30}$$

onde: $\frac{\nabla E}{||\nabla E||}$ é o vetor unitário, identifica a direção de ∇E se $\nabla E \neq 0$, e $dE/dt = 0$.

Figura 2.9.- Espaço de Velocidade (u, v) . Ilustra-se restrição da velocidade do fluxo local expressada pela Eq. 2.28.

Por outro lado, a componente do movimento na direção da gradiente do brilho $(E_x, E_y)^T$, é dada pela relação $E_t / (E_x^2 + E_y^2)^{1/2}$. A componente do Fluxo Ótico sobre os ângulos retos a esta direção não pode ser determinada, ou seja ao longo dos contornos que possuem o mesmo brilho. Isso é uma desvantagem, conhecida como “problema de abertura” (problema, também, presente na visão humana). Este problema é relacionado à impossibilidade de recuperar informação univocamente na direção do movimento, quando é observado um objeto através de uma abertura que é menor que o tamanho do objeto. O problema de abertura é ilustrado na figura 2.10.

Pode-se observar na figura 2.10, que se temos modelos e/ou objetos de uma cena determinada, os elementos de referência não são suficientes para ver a componente transversal do movimento. Também, é possível perceber e estimar a componente da velocidade aparente segunda Eq. 2.30.

Visto que o Fluxo Ótico é considerado como a equação de uma linha restrita, no Espaço de Velocidade; isso implica que a Eq. 2.28 pode ter inúmeras soluções. Observe-se,

também, que onde é possível definir no mínimo duas linhas restritas diferentes, o problema de abertura não aparece. Ver figura 2.11.

Figura 2.10.- Visualização do problema de abertura

No processo de estimar o Fluxo Ótico, surgem interrupções na velocidade local, devido as descontinuidades do brilho da imagem. Estes problemas são causados pelo ruído, pelas oclusões entre objetos em movimento, pelas formas dos modelos , e/ou pelas altas velocidades.

Figura 2.11 O problema de abertura associado à linha restrita no Espaço de Velocidade.

2.6.- Conclusões

Neste capítulo foi mostrado o conceito de velocidade ótica que alguns autores denominam de campo de velocidade, campo de movimento ou também de campo de deslocamento. De qualquer forma, são termos equivalentes, já que tratam de explicar o mesmo fenômeno. Também, foi apresentado o equacionamento do Fluxo Ótico Restrito.

Demostrou-se que as velocidades óticas são referidas como o conjunto de vetores que se formam na superfície do sistema de visão, correspondente em seu conjunto ao movimento do objeto da cena quando existe mudança de posição e/ou rotação. Aqui são representadas as velocidades óticas como as componentes do espaço de velocidade, mas, como é de esperar estas velocidades estão em função das componente x e y do plano de imagem.

No equacionamento do campo pode-se observar um problema que é a profundidade Z . Esta variável converte o sistema de equações em não lineares. Argumenta-se que para obter um sistema razoável de equações são necessários pelo menos cinco pontos de trabalho. Temos duas equações e como se precisam de cinco pontos, o sistema aumenta a dez, ficando a solução mais complexa. Como se pode notar cada ponto adicionado aumenta em dois o número de equações. Rigorosamente, a solução é muito difícil se temos em conta que cada uma das equações do sistema é função de Z .

De qualquer forma, se tem um sistema de equações não lineares, e a sua solução numérica é possível só com condições iniciais, e fazendo uma boa aproximação usando a técnica de agrupamento. Aqui, se deve anotar que este sistema, sem essas condições é instável.

Fazer uma boa aproximação não é muito prático. A maioria das contribuições, no campo, tem feito esforços desenvolvendo estratégias para obter uma boa solução. Outros estudos tratam de eliminar o fator Z , o pelo menos que seus efeitos sejam não perceptíveis. Existem outros fatores que devem ser tomados em consideração na formulação, como por exemplo: princípios de visão, integração espacial/temporal.

CAPITULO 3

PROGRAMA PARA A DETERMINAÇÃO DO FLUXO ÓTICO

3.1.- Introdução

Neste capítulo descrevemos o programa desenvolvido para a determinação do Fluxo Ótico através da análise de duas imagens de entrada sequenciais no tempo. Cada um dos quadros de imagem representa uma cena que ocorreu no espaço 3D. O objetivo do programa é detectar o movimento dos objetos na cena.

Quando o programa é executado, computa as velocidades óticas mediante a análise da intensidade e fornece como saída um diagrama vetorial que mostra a direção da velocidade em diferentes pontos da imagem.

Os processos do programa podem ser agrupados em quatro partes: inicialização, pré-processamento, estimativa do campo de velocidades e finalização.

Na parte de inicialização geram-se as condições iniciais para executar o resto do programa. No pré-processamento das imagens executa-se a detecção de bordas utilizando o método de diferença de gaussianos aplicando máscaras geradas com o critério de Marr-Hildreth, Marr [36]. Neste processo utiliza-se a convolução discreta. Os dados resultantes deste pré-processamento são base para o cálculo do Fluxo Ótico.

Para estimar o campo de velocidades devemos obter os vetores velocidade dos objetos em movimento na cena, esses vetores são obtidos verificando-se a mudança de brilho dos pixels correspondentes de acordo com a estratégia de Anandan, Barron [10]. Para este processo são utilizadas duas imagens sequenciais no tempo. O processo começa encontrando o melhor casamento de brilho entre um pixel da primeira imagem, pixel de referência, com os pixels na vizinhança do pixel de mesmas coordenadas na segunda imagem, incluindo o próprio. Para determinação da vizinhança considera-se uma máscara 3x3 centrada no pixel de referência. O melhor casamento é obtido encontrando a mínima diferença de brilho entre os pixels das duas imagens aplicando o critério da raiz mínima quadrada. Os cálculos são para todos os pixels da imagem. Na parte final de cada cálculo registra-se o deslocamento do pixel da máscara. Neste ponto do processo é possível ter uma indicação grosseira se ocorreu movimento e qual a tendência de sua direção. Caso haja diferença de brilho entre os pixels das duas imagens detecta-se que ocorreu movimento, e se o pixel de melhor casamento na segunda imagem não for o de mesmas coordenadas da primeira imagem pode-se ter uma idéia da direção do movimento em valores discretos de 45°: 45°, 90°, 135°, 180°, 225°, 270°, 315°. O passo seguinte consiste em determinar com maior precisão a direção e o valor da velocidade considerando a superfície de diferenças de brilho entre as duas imagens, em torno do pixel de melhor casamento e aplicando as derivadas numéricas para determinar os máximos e mínimos dessa superfície. Utilizando as derivadas numéricas tem-se uma indicação do valor da velocidade pela rapidez da mudança do brilho e o ângulo que indica a direção do movimento é estimado de forma contínua com valores reais. As derivadas numéricas são calculadas aplicando o operador de “*beaudet*”; estes operadores são definidos como um arranjo matricial de multiplicadores e são aplicados a imagens que representam cenas de objetos com movimento de rotação e translação, Beaudet [11]. No final desta fase executa-se a relaxação das velocidades resultantes com o objetivo de estimar uma média das velocidades calculadas.

Na finalização são gravados os dados resultantes do cálculo da velocidade em um arquivo de saída, a partir do qual é gerado o diagrama vetorial que mostra a existência ou não do movimento no evento analisado.

O programa foi implementado utilizando a linguagem C Padrão ANSI. O programa é estruturado em 12 fases, de acordo com figura 3.1. Ao término de cada fase imprime-se na tela mensagens, outorgando dessa forma segurança na execução do programa.

3.2.- Processos

Nesta seção será descrita a função de cada um dos processos.

3.2.1.- Fases de Inicialização

As fases consideradas nesta parte são as fases 01 a 08. São fases nas quais se geram as condições iniciais de trabalho. O objetivo destas fases é inicializar as variáveis, ponteiros, verificar argumentos, inicializar operadores, gerar máscaras, ler e registrar os quadros de imagens de entrada.

A Fase 01 imprime a mensagem de início da execução do programa. Ver figura 3.2.

A Fase 02 tem como objetivo verificar os argumentos da linha de comando. Ver figura 3.3. A função primeiramente verifica o número apropriado dos argumentos coletados através da Linha de Comando. Se os argumentos estão fora da faixa requerida pelo programa, então a execução da função termina. O número mínimo de argumentos obrigatório é 7, (0-6), e são:

- a) nome do programa executável,
- b) caminho onde se encontram os dados de entrada,
- c) caminho onde se registraram os dados de saída,
- d) nome dos arquivos - imagens sem extensão,
- e) opção obrigatória dos quadros de imagens,
- f) número, extensão, da primeira imagem, e
- g) número, extensão, da segunda imagem.

Logo que os argumentos da Linha de Comando são verificados, a função executa um laço para registrar os dados atribuindo-os as variáveis respectivas. Deve-se observar que se os quadros de imagens não são sucessivos, a execução do programa é terminada. Executa-se, também, um laço para atribuir por omissão as condições iniciais de trabalho.

Figura 3.1.- Diagrama de fluxo do programa principal.

Figura 3.2.- Diagrama de fluxo da Fase 01

Figura 3.3.- Diagrama de fluxo resumido da Fase 02

Podem ser utilizadas duas opções não obrigatórias na linha de comando, sempre que se cumpram as condições de uso. A verificação das opções é realizada através de um laço encadeado. O total dos argumentos da linha de comando são 12.

Este processo utiliza duas funções de complemento: uma que imprime o tipo de erro que acontece na execução do programa e uma outra que informa ao usuário a forma correta de escrever na linha de comando.

A Fase 03 tem como finalidade verificar dados/informação introduzidos através da Linha de Comando e calcular o valor do divisor entre a diferença dos valores da extensão dos arquivos - imagens. Ver figura 3.4.

O processo da fase 04 tem como objetivo produzir o arquivo de saída no qual se registram os dados resultantes do cálculo do Fluxo Ótico. Também a finalidade desta fase é determinar o nome completo dos arquivos de trabalho. Ver figura 3.5.

A Fase 05 tem por finalidade inicializar os ponteiros com “*NULL*”, as máscaras para o cálculo da convolução discreta e os operadores Beaudet para o cálculo das derivadas numéricas. Ver figura 3.6

A inicialização das máscaras está relacionada com a fase 09, na qual se determinam as bordas por diferença de gaussianos. Esta parte está descrita junto com a fase 09.

Por outro lado, na parte que corresponde a inicializar os operadores Beaudet, foram inicializados operadores 3x3, segundo Beaudet [11]. Ver figura 3.7. O intuito deste tipo de operador é executar as derivadas numéricas de primeiro e de segunda ordem e as derivadas parciais.

O processo da Fase 06 tem finalidade calcular o valor do arranjo em função do número de linhas e colunas das imagens. A informação do tamanho da imagem é obtida através da Linha de Comando ou no cabeçalho da própria imagem. O valor calculado é o

Figura 3.4.- Diagrama de fluxo do processo da FASE 03

Figura 3.5.- Diagrama de fluxo do processo da FASE 04

Figura 3.6.- Diagrama de fluxo do processo da Fase 05

$$\frac{1}{6} \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

a) Para calcular as primeiras derivadas parciais em relação a x e a transposta para y

$$\frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \\ -2 & -2 & -2 \\ 1 & 1 & 1 \end{bmatrix}$$

b) Para calcular as segundas derivadas parciais em relação a x e a transposta para y

$$\frac{1}{4} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

c) Para calcular a segunda derivada parcial em relação a x y.

Figura 3.7.- Operador Beaudet 3x3 para o cálculo das derivadas numéricas.

valor da resolução o qual se aplica para o arranjo de dados e atribuições e/ou alocação dinâmica de memória. Ver figura 3.8.

A Fase 07 tem por objetivo criar os endereços para manipular dados e cálculos. Estes endereços são atribuídos aos ponteiros respectivos, criados no programa principal. Ver figura 3.9. Os ponteiros com os endereços atribuídos são aplicados para apontar dados, calcular o valor da resolução, valor do numero de linhas - colunas, reservar memória de forma dinâmica.

A finalidade da Fase 08 é ler os dados contidos nos quadros de imagens de entrada e registra-los em um arranjo de dados. A leitura das imagens é seqüencial, ou seja, primeiro é lida a imagem 1 e após a imagem 2. Ver figura 3.10.

3.2.2.- Fase de Pré-processamento

A Fase 09 tem como objetivo determinar as bordas das imagens utilizando a diferença de dois gaussianos (DOG). Os dados resultantes desta operação são base para o cálculo do Fluxo Ótico. Ver figura 3.11.

Na literatura especializada podem-se encontrar diversas formas para detectar bordas como por exemplo: método do gradiente, método de encaixamento por áreas e/ou métodos que usam gabaritos de casamento. Todos esses métodos executam a primeira derivada. As bordas que se obtém utilizando métodos com a primeira derivada não são satisfatórias quando os objetos que pertencem à cena tem bordas com características de brilho tipo rampa. Este tipo de bordas são típicas quando se tem superfícies com declive iluminadas com fontes de luz comuns.

No presente trabalho utiliza-se o método da segunda derivada com objetivo de eliminar essa desvantagem. Observe-se que um sinal tipo rampa ao ser derivada passa a ser um degrau e ao ser executada a segunda derivada passa a ser um sinal com um impulso positivo no início da borda e um impulso negativo no final da borda. Ver figura 3.12. Para o caso a detecção de borda pode ser realizada de duas maneiras, uma forma é aplicando um detetor de cruzamento por zero e uma outra forma executando o Laplaciano do Gaussiano.

Figura 3.8.- Diagrama de fluxo resumido do processo da Fase 06.
LC: Linha de Comando.

Figura 3.9.- Diagrama de fluxo reduzido do processo da FASE 07

Figura 3.10.- Diagrama de fluxo reduzido do processo da FASE 08

Figura 3.11.- Diagrama de fluxo simples do processo da FASE 09.
DOG: Diferença de Gaussianos

a) Borda tipo rampa

b) Aplicando a primeira e segunda derivada

Figura 3.12.- Mudança do tipo de sinal ao aplicar as derivadas de forma sucessiva.

Marr [36], sugere a proposta conhecida como de Marr - Hildreth, que consiste em aplicar o Laplaciano do Gaussiano. Argumenta-se que este método representa fielmente o sistema de visão humana.

A definição do Laplaciano do Gaussiano é:

$$\nabla^2 G(r) = -\frac{1}{2\pi\sigma^4} \left(2 - \frac{r^2}{\sigma^2}\right) \text{EXP}\left[-\frac{r^2}{2\sigma^2}\right] \quad \text{Eq. 3.1}$$

onde:

$$r^2 = x^2 + y^2$$

O método consiste em aplicar quatro máscaras, isso sugere o uso de diferentes valores de σ . Assim, existiriam máscaras grandes e pequenas. As bordas de valores elevados são detectadas por todas as máscaras e os detalhes pelas máscaras pequenas.

As máscaras podem ser maiores do que 25x25, mas isso gera um custo computacional elevado. A solução do problema é usar a DOG e máscaras pequenas. A DOG é definida como:

$$DOG(\sigma_1, \sigma_2) = \frac{1}{\sqrt{2\pi}\sigma_1} \exp\left[-\frac{x^2}{2\sigma_1^2}\right] - \frac{1}{\sqrt{2\pi}\sigma_2} \exp\left[-\frac{x^2}{2\sigma_2^2}\right] \quad \text{Eq. 3.2}$$

A relação entre sigmas é dada segundo o critério de Marr-Hildreth, Marr [36], assim:

$$\frac{\sigma_2}{\sigma_1} \approx 1.6 \quad \text{Eq. 3.3}$$

É de interesse usar máscaras pequenas já que esse tipo de máscaras detectam tanto as bordas de valores elevados como as pequenas. Uma outra razão é que foi usada a convolução discreta.

3.2.2.1.- Determinação das Máscaras

As máscaras a gerar são máscaras 1D. Na sua geração levou-se em consideração o critério de Marr-Hildreth, Marr [36]. A implementação das máscaras foi realizada através de uma série de tentativas, testou-se diversos valores de σ_1 mantendo a relação σ_2/σ_1 próxima a 1.6. As máscaras foram implementadas considerando-se uma situação de compromisso, considerou-se $\sigma_1 = 2.5$ como valor de projeto. Também por compromisso de projeto considerou-se a relação dos sigmas igual a 1.5, através da qual se obteve $\sigma_2 = 3.75$.

O tamanho da máscara foi calculado considerando as relações:

$$k = 2 \sigma_i + 1 \quad \text{Eq. 3.4}$$

onde, $i = 1, 2$, e

$$-k/2 < x < k/2 \quad \text{Eq. 3.5}$$

Utilizando o operador de molde converte-se esse número real em inteiro e contando a passagem por zero tem-se que a máscara 1 possui 7 elementos e a máscara dois 9 elementos.

A relação utilizada para calcular os valores das máscaras é:

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} \text{EXP}\left[-\frac{x^2}{2\sigma^2}\right] \quad \text{Eq.3.6}$$

Para obter os valores da máscara 1 se trabalho com $\sigma_1 = 2.5$ e $-3 < x < 3$, e para a máscara dois com $\sigma_2 = 3.75$, $-4 < x < 4$. Os valores obtidos mostram-se na figura 3.13.

3.2.2.2.- Operação Convolução e a DOG

Executa-se a convolução discreta a qual consiste em convolucionar uma imagem $f_i(x,y)$, de tamanho $L \times C$ com uma máscara $f_m(x,y)$, de tamanho $M \times N$ para produzir uma imagem convolucionada, Gonzalez e Woods [25] e Pratt [51], assim:

$$f_m(x,y) * f_i(x,y) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f_m(m,n) f_i(x-m, y-n) \quad \text{Eq. 3.7}$$

```
mscr[ 0 ] = 0.077674  
mscr[ 1 ] = 0.115877  
mscr[ 2 ] = 0.147308  
mscr[ 3 ] = 0.159577  
mscr[ 4 ] = 0.147308  
mscr[ 5 ] = 0.115877  
mscr[ 6 ] = 0.077674
```

total valores → 7

soma valores → 0.841295

a) Máscara 1

```
mscr[ 0 ] = 0.059323  
mscr[ 1 ] = 0.078941  
mscr[ 2 ] = 0.096814  
mscr[ 3 ] = 0.109425  
mscr[ 4 ] = 0.113984  
mscr[ 5 ] = 0.109425  
mscr[ 6 ] = 0.096814  
mscr[ 7 ] = 0.078941  
mscr[ 8 ] = 0.059323
```

total valores → 9

soma valores → 0.802989

b) Máscara 2

Figura 3.13.- Valores das máscaras 1 e 2

No trabalho realizou-se a convolução 2D utilizando mascaras 1D. Cada pixel da imagem é convolucionado com cada uma das máscaras. Ver figura 3.14. No processo primeiramente se executa a convolução discreta usando a máscara 1, os dados resultantes desses cálculos são registrados de forma adequada. Logo é executada novamente a convolução aplicando a máscara 2, de forma semelhante os dados são registrados. Após de ter ambos conjuntos de dados covolucionados estima-se a diferença dos valores utilizando a equação 3.2. Produzem-se dessa forma o conjunto de dados base para o estimar o Fluxo Ótico.

3.2.3.- Fase da Estimativa do Fluxo Ótico

A finalidade da Fase 10 é estimar o Fluxo Ótico. Ver figura 3.15. A função que tem por objetivo estimar o fluxo de velocidades é ilustrado na figura 3.16.

3.2.3.1.- Determinação do melhor casamento entre pixels.

O objetivo desta operação é estimar o melhor casamento de brilho entre os pixels dos dois quadros de imagem. O pixel da imagem 1 é o pixel de referência entretanto o pixel da imagem dois é o pixel centrado em uma vizinhança. A operação é ilustrada na figura 3.17. Compara-se cada pixel da primeira imagem com os pixels na vizinhança do pixel de mesmas coordenadas na segunda imagem. Para encontrar o melhor acoplamento utiliza-se o critério da raiz mínima quadrada considerando-se a média dos brilhos no entorno do pixel. O valor retornado do cálculo é registrado e passado por um limiar que garanta a minimização. Logo são gravadas as coordenadas que indicam o deslocamento onde se encontra o melhor acoplamento. Ao término do processo de casamento esta função retorna endereços, os quais são utilizados pela função que determina o fluxo ótico. Os endereços permitem acessar o valor de deslocamento das coordenadas da máscara onde se encontra o valor minimizado. O valor registrado é somado as coordenadas dos quadros de imagens nos cálculos das derivadas numéricas. Este processo está incluído na rotina que determina o melhor acoplamento como se ilustra na figura 3.18.

3.2.3.2.- Operação Cálculo da Raiz Mínima Quadrada

Na figura 3.19, mostra-se o diagrama de fluxo da função que calcula o valor da raiz mínima quadrada. Deve-se fazer notar que em imagens de tamanho $M \times N$ o valor mínimo

Executar convolução 2D utilizando máscara 1D

Figura 3.14.- Operação convolução. Cada pixel da imagem é convolucionado com a máscara e o resultado é armazenado.

Figura 3.15.- Diagrama de fluxo reduzido do processo da Fase 10

Figura 3.16.- Diagrama de fluxo da função calcular fluxo. Onde CNL é controle número de linhas, CNC é controle número colunas, NL é no número de linhas, CC é o número de colunas.

Figura 3.17.- Diagrama em blocos da procura do melhor acoplamento. Onde rmq: raiz mínima quadrada.

Figura 3.18.- Diagrama de fluxo reduzido da função que estima o melhor casamento entre os pixels. Onde: CNL é controle número de linhas, CNC é controle número colunas, NL é no número de linhas, CC é o número de colunas, e rmq é a raiz mínima quadrada.

Figura 3.19.- Diagrama em blocos da operação de estimar o valor da raiz mínima quadrada.

quadrado, entre duas imagens pode ser expressada como a raiz mínima quadrada do valor médio sobre o arranjo $M \times N$, Gonzalez e Woods [25], assim:

$$s = \left\{ \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [f_1(x, y) - f_2(x, y)]^2 \right\}^{1/2} \quad \text{Eq. 3.8}$$

onde f_1 representa a imagem 1 e f_2 a imagem 2. No trabalho se executa o cálculo tendo o pixel da primeira imagem como referência e o pixel da segunda imagem com uma vizinhança que varre uma área 7×7 . A área 7×7 foi determinada em função da máscara utilizada na determinação das bordas. Ver figura 3.20.

3.2.3.3.- Determinação das Velocidades

Na figura 3.21, mostra-se o diagrama em blocos reduzido do processo que calcula o fluxo de velocidades. Para calcular o fluxo de velocidades primeiramente são calculadas as derivadas numéricas aplicando os operadores de “*beaudet*”, Beaudet [11]. O operador é um arranjo matricial de multiplicadores de imagens que é representado como $[O_{ij}]$; com este arranjo executa-se a operação, $\sum_{ij} O_{ij} A_{ij}$, onde $[A_{ij}]$ é um arranjo matricial especificado pela superfície centrada no pixel de melhor casamento. O arranjo matricial A_{ij} representa a superfície 3×3 , cujos valores são preenchidos estimando a raiz mínima quadrada da melhor aproximação entre os pixels correspondentes entre ambas as imagens.

Os operadores derivativos são identificados com os subscritos como D_x e D_y para a derivada parcial de primeira ordem em relação a x e y ; D_{xx} e D_{yy} para a derivada parcial de segunda ordem em relação a x e y e D_{xy} para a derivada parcial de segunda ordem em relação a xy , respectivamente (Ver figura 3.7).

Executada a operação entre ambos os arranjos obtém-se as curvaturas do gradiente direcional. Estes dados são utilizados para extrair os “picos” através das curvaturas principais. Calculadas as derivadas numéricas e segundo Barron et. al. [10] e Beaudet [11], estima-se as curvaturas mínimas e máximas ao longo de $(E_{\min}, E_{\max})$, assim:

Figura 3.20.- Diagrama de fluxo da função que calcula a raiz mínima quadrada

Figura 3.21.- Diagrama de fluxo reduzido do processo para determinar as velocidades.

$$R_{\min} = \frac{1}{2}[(D_{xx} - D_{yy}) - \sqrt{(D_{xx} - D_{yy})^2 + 4D_{xy}^2}] \quad \text{Eq. 3.9}$$

$$R_{\max} = \frac{1}{2}[(D_{xx} - D_{yy}) + \sqrt{(D_{xx} - D_{yy})^2 + 4D_{xy}^2}] \quad \text{Eq. 3.10}$$

Onde o radical representa a excentricidade da curvatura na direção da gradiente. Se o radical é negativo então faze-se $R_{\min}$ e $R_{\max}$ iguais a zero.

O cálculo do ângulo é realizado utilizando o valor de $R_{\min}$ referido ao eixo x positivo, assim:

$$\theta = \text{tg}^{-1}(R_{\min} - D_{xx}, D_{xy}) \quad \text{Eq. 3.11}$$

A rotação do vetor com coordenadas x, y é executado ao longo de E_x , pelo que:

$$X = D_x \cos(\theta) + D_y \sin(\theta) \quad \text{Eq. 3.12}$$

$$Y = -D_x \sin(\theta) + D_y \cos(\theta) \quad \text{Eq. 3.13}$$

Finalmente determinam-se os valores das velocidades resultantes em uma vizinhança do pixel de referência, se $R_{\min}$ e $R_{\max}$ forem diferentes de zero:

$$E_x = \frac{X}{R_{\min}} \quad \text{Eq. 3.14}$$

$$E_y = \frac{Y}{R_{\max}} \quad \text{Eq. 3.15}$$

3.2.3.4.- Operação Relaxação

A parte final deste processo corresponde à função que realiza a relaxação das velocidades estimadas. O objetivo da operação é aplicar um alisamento aos valores das velocidades obtidas. Os cálculos são realizados utilizando um mínimo de 10 iterações. Na

figura 3.22, apresenta-se o diagrama de fluxo do método de relaxação aplicando a técnica de Horn - Schunck, Horn - Schunck [29].

Existem duas equações de velocidade, u e v , (Equações 2.13 e 2.14), para cada ponto na imagem. Resolver estas duas equações simultaneamente é muito complexo pelo método da eliminação de Gauss- Jordan, já que a matriz é muito grande, devido a que o número de linhas - colunas é o dobro dos pixels da imagem. Um outro método é o método de Gauss - Seidel, com características semelhantes.

No presente trabalho adotou-se a técnica iterativa de Horn-Shunck. Essa técnica consiste em calcular numericamente um novo conjunto de velocidades (u^{n+1} , v^{n+1}) a partir dos cálculos estimados e da media das velocidades calculadas (u^n , v^n), assim:

$$u^{n+1} = \tilde{u}^n - \frac{E_x[E_x \tilde{u}^n + E_y \tilde{v}^n]}{\alpha^2 + E_x^2 + E_y^2} \quad \text{Eq. 3.16}$$

$$v^{n+1} = \tilde{v}^n - \frac{E_y[E_x \tilde{u}^n + E_y \tilde{v}^n]}{\alpha^2 + E_x^2 + E_y^2} \quad \text{Eq. 3.17}$$

onde α^2 é o fator de ponderação.

As médias de u e v são calculadas utilizando a máscara mostrada na figura 3.23, Horn [28]. Assim:

$$\tilde{u} = \frac{1}{12}[E_{i-1,j+1} + E_{i+1,j+1} + E_{i-1,j-1} + E_{i+1,j-1}] + \frac{1}{6}[E_{i,j+1} + E_{i-1,j} + E_{i+1,j} + E_{i,j-1}] \quad \text{Eq. 3.18}$$

$$\tilde{v} = \frac{1}{12}[E_{i-1,j+1} + E_{i+1,j+1} + E_{i-1,j-1} + E_{i+1,j-1}] + \frac{1}{6}[E_{i,j+1} + E_{i-1,j} + E_{i+1,j} + E_{i,j-1}] \quad \text{Eq. 3.19}$$

3.2.4.- Fases de Finalização

Esta quarta parte do programa consiste nas fases 11 e 12.

Figura 3.22.- Diagrama de fluxo reduzido do processo de relaxação. Onde: CNI é contador numero de iterações, e NI é numero de iterações.

$1/12$	$1/6$	$1/12$
$1/6$		$1/6$
$1/12$	$1/6$	$1/12$

Figura 3.23.- Máscara para estimar a média das velocidades. Apresenta-se os pesos adequados com os quais os valores podem ser multiplicados.

A objetivo da Fase 11 é registrar o fluxo estimado em um arquivo de saída de formato normalizado. Ver figura 3.24.

A Fase 12 tem como objetivo imprimir na tela a mensagem de fim da execução do programa. Ver figura 3.25.

3.3.- Conclusões

No presente capítulo foi apresentado a estrutura do programa. Observou-se que este está constituído de 12 fases e é totalmente estruturado. Após adquirir informação através da Linha de Comando o programa gera as condições iniciais de trabalho. O programa é capaz de acessar informação do número de linhas colunas no cabeçalho das imagens. Processa informação e gera dados para a continuação dos processos. Após as imagens terem sido lidas e registradas em um arranjo é executada a diferenciação gaussiana para determinar-se as bordas. Aplica-se a convolução discreta. Nesta seção utilizou-se o critério de Marr - Hildreth.

Logo foi estimado o Fluxo Ótico, considerando a estratégia de casamento de pixels de Anandan. Utilizou-se o critério da raiz mínima quadrada para encontrar a melhor casamento entre os pixels de duas imagens. Depois, foram calculadas as velocidades óticas. Ao final do processo , os dados resultantes obtidos foram atenuados, aplicando a técnica de relaxação de Horn - Schunck.

Finalmente os resultados foram gravados em um arquivo de saída de formato "*Burkitt*". O programa finalizou com a impressão da mensagem de fim de execução.

Deve-se observar que no programa foram usadas algumas funções comuns, por exemplo aquela que concatena dois "*strings*" de entrada em um só "*string*" de saída e a função que converte um inteiro a "*string*".

No que se refere à alocação de memória, o que foi feito foi reservar memória em função do tamanho das imagens. A alocação de memória dessa forma passou ser dinâmica. Também, o programa é capaz de trabalhar com imagens até de 1024x1024 pixels e não necessariamente quadradas.

Figura 3.24.- Diagrama de fluxo reduzido para o processo da Fase 11

Figura 3.25.- Diagrama de fluxo da Fase 12

Os diagramas de fluxo apresentados foram reduzidos, deliberadamente, com o intuito de torná-los didáticos. Para melhor entendimento de cada um dos processos deve-se referenciar ao programa, colocado no Apêndice B. Outro ponto que deve ser salientado e que tentou-se utilizar os melhores recursos da linguagem C para ganhar velocidade na execução do programa o qual foi desenvolvido partindo do geral para o particular e aplicando o método científico de Prova - e - Erro.

CAPITULO 4

RESULTADOS EXPERIMENTAIS E ANÁLISE

As imagens utilizadas nos ensaios foram imagens monocromáticas de formato normalizado do tipo “*rasterfile*” (“*Work Station SUN*”). Este tipo de formato é constituído de três partes: a primeira é o cabeçalho de 32 “*bytes*”, a segunda um conjunto de valores para o “*colormap*”, no nosso caso vazio por trabalhar com imagens monocromáticas e finalmente o conjunto de pixels das imagens armazenados como “*unsigned char*”, ou seja, dados do tipo caracter sem sinal, arrançados de forma sequencial.

No cabeçalho se encontra a informação do número de colunas e linhas da imagem, nessa ordem. Assim, se não se conhece o tamanho das imagens o programa é capaz de obter essa informação do próprio cabeçalho. A leitura dos arquivos-imagens se executa um por vez atribuindo os valores do numero de linhas e colunas às variáveis respectivas, logo comparam-se se são compatíveis desse ponto de vista. O programa pode trabalhar com imagens cujo tamanho chegue até 1024x1024 pixels.

Como saída se obteve um arquivo de formato normalizado do tipo “*burkitt*”, através do qual se obtêm o Diagrama Vetorial. Esse tipo de arquivo é estruturado de duas partes: a primeira é o cabeçalho no qual se gravam os dados do número de colunas e filas do tamanho

original da imagem, logo se registra a informação do tamanho resultante da imagem, tamanho reduzido do número de colunas e filas da imagem eliminando os pixels de borda da imagem; depois se gravam os dados que indicam o valor de deslocamento a partir do qual se começa a registrar os dados do corpo da imagem. Os dados a registrar é na ordem número de colunas e filas. A segunda parte do arquivo corresponde aos dados do corpo da imagem.

O arranjo das imagens de entrada que foram testadas são as seguintes:

- a) quadro preto - translação ascendente (imagem sintética),
- b) quadro branco - translação descendente (imagem sintética),
- c) disco - rotação (imagem real),
- d) tráfego - translação rotação (imagem real),
- e) cilindro - divergência (imagem real),
- f) frente - onda senoidal (imagem sintética), e
- g) percolação - translação vertical (imagem real).

O computador utilizado no desenvolvimento do trabalho foi uma estação de trabalho “*SUN*”. A CPU desta máquina é Sun versão 4, cuja velocidade é de 65 MHz, e o volume da RAM é de 16384 K. No Apêndice A mostram-se as características gerais da máquina.

A continuação descreveremos cada um dos ensaios. A forma de colocar o tamanho das imagens é LINHAS x COLUNAS x OITO BITS. Os oito bits indicam os 256 níveis de cinza.

a) quadro_p.4 e .5

A figura 4.1 mostra na parte superior imagem inicial (esquerda) e final (direita) de uma seqüência de dois quadros de imagens sintéticas. O tamanho das imagens é de 100x100x8. Trata-se de um quadro preto, com bordas borrosas, se movendo-se em um fundo branco. O quadro se deslocam diagonalmente de forma ascendente com uma determinada velocidade. Na parte inferior da mesma figura se apresenta o Diagrama Vetorial correspondente ao evento. Por inspeção do Diagrama Vetorial observamos o sentido dos vetores os quais indicam o movimento diagonal ascendente à direita do objeto. O tempo de execução do programa para este ensaio foi de 8 minutos.

b) quadro_b.7 e .9

A figura 4.2 mostra a imagem inicial e final de uma seqüência de dois quadros de imagens sintéticas. O tamanho das imagens é de 100x100x8. Trata-se de um quadro branco, com bordas bem definidas, se movendo em um fundo preto. O quadro branco se desloca de forma descendente com uma determinada velocidade. Na mesma figura se apresenta o Diagrama Vetorial correspondente ao evento. Por inspeção do diagrama notamos no sentido dos vetores o movimento descendente à esquerda do objeto. O tempo de execução do programa para este ensaio foi de 8 minutos.

c) disco.3 e .4

A figura 4.3 mostra a imagem inicial e final de uma seqüência de dois quadros de imagens reais. O tamanho das imagens é de 240x320x8. O evento da cena trata de um disco com movimento de rotação em sentido antihorário. Na parte inferior da figura mostra-se o Diagrama Vetorial correspondente, no qual por inspeção podemos observar o movimento dado pelos vetores. O tempo de execução do programa para este ensaio foi de 58 minutos.

d) tráfego.1 e .3

A figura 4.4 apresenta a imagem inicial e final de uma seqüência de dois quadros de imagens reais. O tamanho das imagens é de 190x240x8. O evento corresponde a uma cena de tráfego veicular. Na parte inferior da figura se mostra o Diagrama Vetorial correspondente ao ensaio. No diagrama podemos observar o movimento de quatro objetos, três dos quais correspondem ao movimento de veículos, um que está contornando a esquina, outro que se move de direita para esquerda e um outro que se move de esquerda para direita; o quarto movimento corresponde a uma pessoa que está caminhando sobre a calçada. O tempo de execução para este ensaio foi de 32 minutos aproximadamente.

d) cilindro.16 e .18

Na parte superior da figura 4.5, apresenta-se a imagem inicial e final de uma seqüência de dois quadros de imagens reais. O tamanho das imagens é de 300x300x8. O evento trata de uma cena onde um cilindro (lata de refrigerante), fica mais próximo da câmara, ocasionando um movimento divergente. Na parte inferior da figura mostra-se o

Diagrama Vetorial correspondente ao ensaio e por inspeção observamos o sentido do movimento. Note-se a divergência dos vetores, os quais indicam que o objeto fica cada vez mais perto da câmara e em seu conjunto saem de um Foco de Expansão. O tempo de execução para este ensaio foi de 73 minutos.

d) frente.8 e .9

Na parte superior da figura 4.6 apresenta-se a imagem inicial e final de uma seqüência de dois quadros de imagens sintéticas. O tamanho das imagens é de 100x100x8. A cena é um evento que executam duas senoides superpostas, realizando um movimento de frente de ondas. Na parte inferior da figura está apresentado o Diagrama Vetorial correspondente à cena; note-se o frente de vetores, como era de se esperar já que se trata de frente de pixels em movimento. O tempo de execução do programa para este ensaio foi de 10 minutos.

d) percolação.2 e .3 e .14 e 15

As duas primeiras imagens da figura 4.7a mostram a imagem inicial e final de uma seqüência de dois quadros de imagens reais. O tamanho das imagens é de 260x96x8. O cena corresponde a um estudo do fenômeno de percolação de água em solos. No evento o movimento é gerado pela ação da gravidade. Na parte direita da figura apresenta-se o Diagrama Vetorial respectivo. Observa-se o sentido para baixo dos vetores consequência do movimento vertical da água. O tempo de execução do programa para este ensaio foi de 16 minutos. De forma semelhante se ensaio com a seqüência de imagens .14 e .15, correspondente a ação intermédia da percolação; ver figura 4.7b.

Das figuras apresentadas, figuras 4.1, 4.2, ... , 4.7, tem-se duas partes uma é a seqüência das imagens de entrada e a outra correspondente ao Diagrama Vetorial de saída. O programa para a determinação do fluxo ótico utiliza apenas duas imagens de entrada, imagens arranjadas de forma seqüencial no tempo. As imagens apresentadas pertencem, na realidade, a uma seqüência completa da cena, por exemplo a seqüência da percolação de água em solos possui um conjunto de imagens.

Nos Diagramas de Agulhas pode-se observar os diversos tipos de movimento, correspondentes aos eventos das cenas citadas. Assim, tem-se movimentos de translação

ascendente de esquerda para direita e movimento de translação descendente de direita para esquerda, indicando em seu conjunto um deslocamento de forma diagonal. Tem-se, também o movimento de rotação em sentido antihorário. Tem-se o movimento translação-rotação, detectando o movimento combinado. Para o caso do movimento divergente o diagrama mostra um conjunto de vetores que indicam o tipo de movimento. No caso da frente de ondas se observa um resultado de boa qualidade. Finalmente tem-se o fenômeno da percolação de água em solos.

Em todos os ensaios o programa foi executado com 10 iterações. O desvio padrão usado foi de 2.5, o que deu máscaras adequadas para o cálculo da convolução discreta; no processo de executar a Diferença de Gaussianos.

O tempo de execução do programa é função do valor do desvio padrão, o número de iterações, o tamanho das imagens, os operadores para o cálculo das derivadas numéricas. No caso dos quadros preto e branco cujos tamanhos são de 100x100x8, com um σ de 2.5, com 10 iterações o programa levou 8 minutos para ser executado, entretanto no caso do disco, cujo tamanho das imagens são de 240x320x8, com σ de 2.5 e com 10 iterações, o tempo para execução do programa foi 58 minutos.

Figura 4.1.- Na parte superior apresenta-se a cena de um quadro negro se mexendo em um fundo branco. Na parte inferior mostra-se o Diagrama - Vetorial referente ao evento. (Imagem sintética).

Figura 4.2.- Na parte superior apresenta-se a cena de um quadro branco se mexendo em um fundo preto. Na parte inferior mostra-se o Diagrama Vetorial referente ao ensaio. (Imagem sintética).

Figura 6.3.- Na parte superior apresenta-se a cena de um disco rodando. Na parte inferior mostra-se o Diagrama Vetorial correspondente ao evento. (Imagem captada no nosso laboratório utilizando uma câmara de vídeo tipo camcorder).

Figura 4.4.- Na parte superior apresenta-se uma cena de tráfego veicular. Na parte inferior mostra-se o Diagrama Vetorial referente ao ensaio. (Imagem trazida via rede Internet do endereço: <ftp.eecs.umich.edu>).

Figura 4.5.- Na parte superior apresenta-se a seqüência de dois quadros de imagens referente ao evento divergente. Na parte inferior mostra-se o Diagrama Vetorial correspondente ao ensaio. (Imagem trazida via rede Internet do endereço: <ftp.eecs.umich.edu>).

Figura 4.6.- Na parte superior apresenta-se um evento de um frente de onda senoidal. Na parte inferior mostra-se o Diagrama Vetorial correspondente ao ensaio. (Imagem sintética).

Figura 4.7a.- As duas primeiras imagens mostram a ação da percolação de água em solos. As imagens correspondem à parte inicial do evento. A terceira imagem mostra o Diagrama Vetorial referente ao ensaio. Imagens da percolação gentilmente cedidas pelo CNPDIA - EMBRAPA São Carlos.

Figura 4.7b.- As duas primeiras imagens representam o evento intermédio da ação de percolação de água em solos. A terceira imagem mostra o Diagrama Vetorial do ensaio correspondente. Imagens da percolação gentilmente cedidas pelo CNPDIA - EMBRAPA São Carlos.

CAPITULO 5

CONCLUSÕES E SUGESTÕES DE TRABALHOS FUTUROS

5.1.- Conclusões

No presente trabalho foi desenvolvido uma metodologia para estimar o Fluxo Ótico a partir de duas imagens de entrada. Aplicou-se estratégia de casamento de pixels de Anandan, considerando a raiz mínima quadrada para encontrar o melhor valor de acoplamento entre os pixels. Com a finalidade de melhorar os resultados obtidos na estimativa do fluxo foi aplicada a técnica de relaxação de Horn-Schunck.

Deve-se ressaltar, também, que no processo de desenvolvimento do programa, foram construídas duas máscaras gaussianas. Efetuou-se dessa forma considerando os argumentos de Marr - Hildreth, já que foi argumentado que o operador apropriado para detectar bordas é o Laplaciano do Gaussiano. Também afirmou-se que este método é um bom modelo do processamento do sistema de visão humana. Observou-se no desenvolvimento da metodologia que as máscaras de valor elevado são uma desvantagem pelo alto custo computacional, aprofundando o problema já que no trabalho se implementou a convolução discreta. Recomendou-se pois o uso da diferença de dois gaussianos, denominado Diferença Gaussiana - DOG. No caso aplicou-se o critério de Marr-Hildreth, no qual, a relação entre σ_2/σ_1 seria aproximadamente de 1.6. No projeto usou-se o valor de 1.5, obtendo-se bons resultados.

Os resultados obtidos nos ensaios são de qualidade, como se pode notar nos Diagramas Vetoriais claros, figuras 4.1, 4.2. ... , 4.7. Observa-se que através de inspeção visual de ditos diagramas tem-se a confirmação de que existiu movimento do objeto na cena e o sentido dos vetores em seu conjunto dá a orientação do deslocamento deste.

No que se refere as seqüências das imagens trabalhou-se com duas classes; as chamadas imagens sintéticas e imagens reais. Utilizou-se imagens que respondem a diferente tipos de eventos, assim, tem-se movimento de translação (quadro negro e quadro branco), rotação (disco), translação-rotação (tráfego veicular), divergência (cilindro), frente de onda (senóide) e percolação de água em solos (movimento combinado).

Dos Diagramas Vetoriais correspondentes, pode-se observar que o programa teve baixo desempenho nas cenas de rotação e divergência, nos outros eventos os resultados obtidos foram bons.

No referente ao tempo de execução do programa pose-se notar que este é diretamente proporcional ao tamanho das imagens, do numero de iterações, dos operadores para o cálculo das derivadas numéricas, das máscaras gaussianas e das características da máquina e/ou dos requerimentos do usuário.

O trabalho foi desenvolvido em um ambiente *SUN*. Os formatos dos imagens de entrada foram do tipo “*ras*”, e do arquivo de saída tipo “*burkitt*”. O arquivo de saída foi convertido a “*ps*”.

A metodologia adotada foi a do ensaio-erro e a analítica.

As aplicações da pesquisa são bastante amplas e abrangentes. As áreas nas quais o trabalho pode ser aplicado são diversas, como por exemplo: medicina, biologia, navegação, tomografia, comunicações, meteorologia, animação, rastreamento de objetos, guiamento, monitoramento de tráfego veicular e física dos solos.

5.2.- Sugestões para trabalhos futuros

O presente trabalho abre perspectivas de continuidade em pesquisa na área de detecção do movimento utilizando a técnica do fluxo ótico. Entre os possíveis tópicos que podem ser abordados como continuação do trabalho, podemos citar:

- a) Aplicar o programa a sequência de imagens onde tenham-se objetos ocluídos.
- b) Realizar Histogramas e cálculos estatísticos,
- c) Usar diversos métodos matemáticos no processamento das imagens,
- d) Aplicações diretas em áreas específicas,
- e) Utilizar mecanismos de software e processamento de imagens por tempo real para aumentar a velocidade para a determinação do fluxo ótico.

BIBLIOGRAFIA

- [01] ADIV, G. Recovering 2-D motion parameters in scenes containing multiple objects.
Proc. DARPA Image Understanding Workshop, Arlington: 285-292, 1983.

- [02] ADIV, G. Recovering motion parameters in scenes containing multiple moving objects. *Proc. IEEE Conference Computation. Vision Pattern Recognition*.
Washington, DC: 399-400, 1983.

- [03] ADIV, G. Determining three-dimensional motion and structure from optical flow generated by several moving objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7 (4) : 525-542, July 1985.

- [04] ADIV, G. Inherent ambiguities in recovering 3-D motion and structure from a noise flow field. . *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11 (5) : 477-489, May 1989.

- [05] AGGARWAL, J. K. (Guest ed.). Motion and time varying imagery. *Computer Vision, Graphics and Image Processing*. 21 : 1-2, 1983.

- [06] AGGARWAL, J. K. and BADLER (Guest eds.). Special issue on motion and time-varying imagery. *IEEE Transactions on Pattern Analysis a Machine Intelligence*, 2 (6) : 493-494, November 1980.

- [07] AGGARWAL, J. K. and DUDA, R. D. Computer analysis of poligonal images. *IEEE Transactions on Computer*. 24 (10) : 445-522, October 1975.

- [08] BALLARD, D. H. and BROWN, C. M. *Computer Vision*. New Jersey, Prentice Hall, 1982. 523p.
- [09] BALLARD, D. H. and SABBAH, D. Viewer independent shape recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5 (6) : 653-660, 1983.
- [10] BARRON, J. L.; FLEET, D. J.; BEAUCHEMIN, S. S. AND BURKITT, T.A. Performance of optical flow techniques. *CVPR*. 236-242, 1992.
- [11] BEAUDET, P. R. Rotationally invariant image operators. *Proceedings of the 4th International Joint Conference on Pattern Recognition*, Kyoto, Japan, 579-583, 1978.
- [12] BEN-TZVI, D.; DEL BIMBO, A. and NESI, P. Optical flow constraint line parametrization. *Pattern Recognition*, 26 (10) : 1549-1561, 1993.
- [13] BEN-TZVI, D. and SANDLER, M. B. A combinational Hough transform. *Pattern Recognition Letters*, 11(3) : 167-174, March 1993.
- [14] BRUSS, A. R. and HORN, B. K. P. Passive navigation. *Computer Vision, Graphic and Image Processing*, 21 : 3-20, 1983.
- [15] COWARD, A E.; SNYDER, W. E. and RUEDGER, W. H. The detection of unresolved targets using the Hough transform. *Computer Vision, Graphics and Image Processing*, 21: 222-238, 1983.
- [16] DEL BIMBO, A.; NESI, P. and SANZ, J. L. Optical flow computation using extended constraints. *Tech. Rep. Dipartamenti de Sistemi e Informatica, Facolta' di Ingegneria, Universite' di Firenze*, DSI-RT, 19/92.

- [17] DEL BIMBO, A.; NESI, P. and SANZ, J. L. Analysis of optical flow constraints. *Tech Rep. Dipartimento de De Sistemi e Informatica, Facolta' di Ingegneria, Universite' di Firenze*, November 92.
- [18] DUDA, R. O. and HART, P. E. Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15 (1) : 11-15, 1972.
- [19] DOOR, A. J. van. and KOENDERINK, J. J. The estructure,of the humam motion detections system. *IEEE Transactions on Systems, Man and Cybernetics*, 13 (5) : 916-922, September 1983.
- [20] DUNCAN, J. H. and CHOU, T. -C. On the detection of motion and the computation of optical flow. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14 (3) : 346-351, March 1992.
- [21] FANG, J. -Q. and HUANG, T. S. Solving three-dimensional small-rotation motion equations. *Proc. IEEE Conference of Computer Vision and Pattern Recognition*, Washington, DC, USA: 253-258, 1983.
- [22] FANG, J. -Q. and HUANG, T. S. Estimating 3-D movement of a rigid object: experimental results. *Proc. Int. Joint Conference of Artificial Intelligence*, Karlsruhe, West Germany: 1035-1037, 1983.
- [23] FANG, J. -Q. and HUANG, T. S. Solving three-dimensional small-rotation motion equations: uniqueness, algorithms and numerical results. *Computer Vision, Graphics and Imaging Processing*, 26: 183-206, 1984.
- [24] FENEMA, C. L. and THOMPSON, W. B. Velocity determination in scenes containing several moving objects. *Computer Vision Graphics and Image Processing*, 9: 301-315, April 1979.
- [25] GONZALEZ, R. C. and WOODS. R. E. *Digital Image Processing*. USA, Addison-

Wesley, 1987, 503p.

- [26] GREGORY, R. L. *Olho & Cérebro*. Rio de Janeiro, Brasil, Zahar. 1979. 251p.
- [27] GRIMSON, W. E. L. and HUTTENGLOCHER, D. P. On sensivity of Hough transform for object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12 (3) : 255-274, 1990.
- [28] HORN, B. K. P. *Robot Vision*. Massachussets, USA, MacGraw-Hill, 1986. 509p.
- [29] HORN, B. K. P. and SCHUNK, B. G. Determining optical flow. *Artificial Intelligence*, 17 : 185-203, 1981.
- [30] JAIN, R. Dynamic scene analysis using pixel-based processes. *IEEE Computer Society*, 14 (8) : 12-18, August 1981.
- [31] KEARNEY, J. K.; THOMPSON, W. B. and BOLEY, D. L. Optical flow estimation: on error analysis of gradient-based methods with local aproximation. *IEEE Transactions on Pattern analysis and Machine Intelligence*, 9 (2) : 229-244, March 1987.
- [32] LAWTON, D. T. Motion analysis via local translational processing. *Proc. Workshop Computer vision*, Rindge, NH: 59-72, 1982.
- [33] LAWTON, D. T. and RIEGER, J. H. The use od difference field in processing sensor motion. *Proceeding Image Understanding, Workshop*, Washington, DC, USA: 77-83, 1983.
- [34] LONGUET-HIGGINS, H. C. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293: 133-135, 1981.
- [35] MAITRE, H. Contribution to the prediction of performances of the Hough transform.

IEEE Transactions on Pattern Analysis and Machine Intelligence, 8 (5) : 669 - 674, September 1986..

- [36] MARR, D. *Vision*. New York, USA, W. H. Freeman, 1982. 597p.
- [37] MARTIN, W. N. and AGGARWAL, J. K. Dynamic scene analysis: a survey. *Computer Graphics and Image, Processing* 7 : 356-374, 1978.
- [38] MEIRI, A. Z. On monocular perception of 3D moving object. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 2 (6) : 582-583, November 1980.
- [39] MITICHE, A. Computation of optical flow and rigid motion. *Proceedings of Workshop on Computer Vision: Representation and Control Annapolis, Maryland*: 63-71, 1984.
- [40] MITICHE, A . On combining stereopsis and kineopsis for space perceptions. *Proceedings of 1st. Conference on Artificial Intelligence Applications*, Denver, Colorado: 156-160, 1984.
- [41] MITICHE, A. A comment "on kineopsis and computations of structure and motions". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11 (5) : 540-541, 1989.
- [42] NAGEL, H, -H. On the derivation of 3-D rigid point configurations from image sequences. *Proc. IEEE Conference of Pattern Recognition and Image Processing*, Dallas, Tx : 103-108, 1981.
- [43] NAGEL, H. -H. Representation of moving rigid objects based on visual observation. *IEEE computer Society*, 14 (8) : 29-39, August 1981.
- [44] NAGEL, H. -H. and NEUMANN, B. On 3-D reconstruction from two perspective views. *Proc. int. Joint Conference Artificial Intelligence*, Vancouver, Canada:

661-663, 1981.

- [45] NAKAYAMA, K. and LOOMIS, J. M. Optical velocity patterns, velocity-sensitive neurons, and space perceptions: a hypothesis. *Perception*. 3 : 69-80, 1974.
- [46] NESI, P.; DEL BIMBO, A. and BEN-TZVI, D. Algorithms for optical flow estimation in real-time on connection machine-2. *Tech. Rep. Dipartimento de Sistemi e Informatica, Facolta' di Ingegneria, Universite' di Firenze, DSI-RT, 24/92, Italy, 1992.*
- [47] NEUMANN, B. Motion analysis of image sequences for object grouping and reconstruction. *Proc. Int. Conference of Pattern Recognition*, Maimi, FL: 1262-1265, 1980.
- [48] O'ROURKE, J. Motion detection using Hough techniques. *Proc. IEEE Conference of Pattern Recognition and Image Processing*, Dallas Tx: 82-87, 1981.
- [49] PAQUIN, R. and DUBOIS, E. A spacto-temporal gradient method for estimating the displacement field in time-varying imagery. *Computer Graphics and Image Processing*, 21: 205-221, 1983.
- [50] POTTER, J. L. Scene segmentation using motion information. *Computer Vision, Graphics and Image Processing*, 6 : 558-581, 1977.
- [51] PRATT, K. P. *Digital Image Processing*. John Wiley & Sons, New York.1978. 750p.
- [52] PRAZDNY, K. Egomotion and relative depth map from optical flow. *Biology and Cybernetic*, 36 : 87-102, 1980.
- [53] PRAZDNY, K. Determining the instantaneuos direction of motion from optical flow generated by a curvilinearly moving observer. *Proc. IEEE Conference of Pattern Recognition and Image Processing*, Dallas, Tx: 109-114, 1981.

- [54] PRAZDNY, K. On the information in optical flow. *Computer Vision, Graphics and Image Processing*, 22 : 239-259. 1983.
- [55] RIEGER, J. H. and LAWTON, D. T. Determining the instantaneous axis of translation from optical flow generated by arbitrary sensor motion. *Workshop Motions: Representation, Perception*, Toronto, Canada: 33-41, 1983.
- [56] ROACH, J. W. and AGGARWAL, J. K. Determining the movement of objects from a sequence of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2 : 554-562, November 1980.
- [57] SCHUNCK, B. G. Image flow segmentation and estimation by constraint line clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11 (10) : 1010 - 1027, October 1989.
- [58] SNYDER, W. E. (Guest ed.). Computer analysis of time-varying images. *IEEE Computer Society*, 14 (8) : 7-9, August 1981.
- [59] THOMPSON, W. B. Combining motion and contrast for segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2 (6) : 543-549, November 1980.
- [60] THOMPSON, W. B. and BARNARD, S. T. Lower-level estimation and interpretation visual motion. *Computer*. 14 : 20-28, August 1981.
- [61] TSAI, R. Y. and HUANG, T. S. Estimating three-dimensional motion parameters of a rigid planar patch. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 29 (6) : 1147-1152, December 1981.
- [62] TSAI, R. Y. and HUANG, T. S. Uniqueness and estimation of three-dimensional motion parameters of a rigid planar patch from three perspective views. *Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing*, Paris,

France: 834-838, 1982.

- [63] TSAI, R. Y. and HUANG, T. S. Estimating three-dimensional motion parameters of a rigid planar patch, II: singular value decomposition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 30 (4) : 525-534, August 1982.
- [64] TSAI, R. Y. and HUANG, T. S. Uniqueness and estimation of three-dimensional motion parameters of a rigid objects with curved surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6 : 13-27, January 1984.
- [65] TSAI, R. Y. and HUANG, T. S. Estimating three-dimensional motion parameters of a rigid planar patch, III: finite point correspondence and the three-view problem. *IEEE Transaction on Acoustics, Speech and Signal Processing*, 32 (2) : 213-220, April 1984.
- [66] TSOTSOS, J.; MYLOPOULOS, J.; COVEY, D. and ZUCKER, S. W. A framework for visual motion understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2 (6) : 563-573, November 1980.
- [67] ULLMAN, S. Analysis if visual motion by biological and computer systems. *IEEE Computer Society*, 14 (8) : 57-69, August 1981.
- [68] VERRI, A. and POGGIO, T. Motion field and optical flow: qualitative properties. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11 (5) : 490-498, May 1989.
- [69] WEBB, J. A. and AGGARWAL, J. K. Visually interpreting the motion of objects in space. *IEEE Computer Society*, 14 (8) : 40-46, August 1981.
- [70] YEN, B. L. and HUANG, T. S. Determining 3-D motion structure of a rigid body over 3 frames using straight line correspondences. *Proceedings of the IEEE Conference of Computer Vision and Pattern Recognition*, Washington, DC: 267-272, 1983.

- [71] YOUNG, T. I. and FU, K. -S. (eds.). *Handbook of Pattern Recognition and Image Processing*. San Diego, Academic Press, 1986. 705p.

Apêndice A

Característica da *Work Station SUN*

CPU	Sun Versão 4
Frequência	65 MHz
RAM	16 384 KBytes
HD	3 693 332 KBytes
Swap	67 600 KBytes
Dump	67 588 KBytes

Apêndice B

Programa para Estimar o Fluxo Ótico

```

/*****
FINALIDADE:
  Estimar o Fluxo Otico a partir de duas imagens de entrada. Usa-se:
  -> estrategia de Anandan para o casamento de pixels,
  -> argumento de Marr-Hildreth para executar a DOG,
  -> metodo da raiz minima quadrada para encontrar melhor casamento
  -> tecnica de relaxacao de Horn-Schunck para atenuar o fluxo,
  -> operadores de Beaudet para calcular as derivadas numericas
  -> metodologia computacional de Beauchemin
*****/
/* INCLUSAO BIBLIOTECAS */
#include <math.h>
#include <stdio.h>
#include <fcntl.h>

/* CONSTANTES QUE CONTEM INFORMACAO DOS QUADROS DE IMAGENS */
#define NUM_QUA    2    /* numero de quadros de imagens */

#define N3        8
#define N4        16
#define N5        32
#define N6        64
#define N7        128
#define N8        256
#define N9        512
#define N10       1024

#define B_CABE    32    /* numero de bytes do cabecalho da imagem */

/* CONSTANTES PARA A IMPLEMENTACAO DO ALGORITMO */
#define FALSO      0
#define VERDDR     1
#define LER_CABE   12
#define ITER_ATENR  10
#define CUMP_VTR    3
#define COMPR_STR  100
#define TAM_CARACT  4
#define FORMATO_X   8
#define FORMATO_Y   8
#define FORMAT      4
#define NUM_BYTES   4
#define AREA        3
#define VTR_ACU_RMQ AREA*AREA
#define PI          3.14159265
#define LIM_MIN     50
#define CONST_DIV   2.0
#define EXPONNT     2.0
#define ALFA        5.0
#define SIGMA       2.5
#define APRX_M_H    1.4

/* MENSAGENS DE INICIO E FIM DA EXECUCAO DO PROGRAMA */
char *mensa_inici = "FLUXO OTICO : Inicio Execucao";
char *mensa_final = "FLUXO OTICO : Fim da Execucao";

/* NOVO NOME PARA O TIPO DE DADO char */
typedef char string[COMPR_STR];

/* NOMEANDO VARIAVEIS ESTRUTURADAS SEGUNDO O TIPO DE DADOS A USAR */
typedef struct dado_rasterf {
    int var_art, numr_linh, numr_coln;
} rasterf_dado;

typedef struct dado_area {
    float AREA_DD[AREA][AREA];
} area_dado;

typedef struct dado_mascara {
    int intr_mscr;
    float flut_mscr;
    float mscr[N10];
} mascara_dado;

```

```

mascara_dado mascr_1, mascr_2; /* variaveis externas */

```

```

typedef struct dado_operador_BDT {
    int intr_oprd_BDT;
    float flut_oprd_BDT;
    float OPRD_BDT[AREA][AREA];
} operador_BDT_dado;

```

```

/* variaveis externas */

```

```

operador_BDT_dado var_oper_BDT_x, var_oper_BDT_y, var_oper_BDT_xx, var_oper_BDT_yy, var_oper_BDT_xy;

```

```

typedef struct dado_vtr_deslocmt {
    float vtr_desl_x, vtr_desl_y;
} vtr_deslocmt_dado;

```

```

typedef vtr_deslocmt_dado camp_desl_8[N3*N8];
typedef vtr_deslocmt_dado camp_desl_16[N4*N4];
typedef vtr_deslocmt_dado camp_desl_32[N5*N5];
typedef vtr_deslocmt_dado camp_desl_64[N6*N6];
typedef vtr_deslocmt_dado camp_desl_128[N7*N7];
typedef vtr_deslocmt_dado camp_desl_256[N8*N8];
typedef vtr_deslocmt_dado camp_desl_512[N9*N9];
typedef vtr_deslocmt_dado camp_desl_1024[N10*N10];

```

```

typedef struct dado_param {
    float parm_angl, parm_cur_min, parm_cur_max;
} param_dado;

```

```

typedef param_dado param_8[N3*N3];
typedef param_dado param_16[N4*N4];
typedef param_dado param_32[N5*N5];
typedef param_dado param_64[N6*N6];
typedef param_dado param_128[N7*N7];
typedef param_dado param_256[N8*N8];
typedef param_dado param_512[N9*N9];
typedef param_dado param_1024[N10*N10];

```

```

typedef float img_8[N3*N3];
typedef float img_16[N4*N4];
typedef float img_32[N5*N5];
typedef float img_64[N6*N6];
typedef float img_128[N7*N7];
typedef float img_256[N8*N8];
typedef float img_512[N9*N9];
typedef float img_1024[N10*N10];

```

```

typedef struct dado_base {
    int resol, tam_x, tam_y;
    camp_desl_1024 *p_vetri;
    param_1024 *p_param;
    img_1024 *p_gaussn, *p_laplcn;
} base_dado, *p_base_dado;

```

```

/* =====> PROGRAMA <==> PRINCIPAL <===== */
main( contador_arg, mtrz_ptr_arg )
int contador_arg;
char *mtrz_ptr_arg[];
{
    p_base_dado registroC_img1, registroD_img1, registroC_img2, registroD_img2,
        nom_arq_reg1, nom_arq_reg2;
    string rotr_ent, rotr_sai, nom_arq_ent, nom_arq_sai, nom_comp_arq_img1, nom_comp_arq_img2;
    unsigned char CABE_IMG[B_CABE];
    int qua_num_1, qua_num_2, num_iter_atenr, dado_num_lin_col, coluna, linha, num_qua, tam_arj=0,
        x[NUM_QUA], y[NUM_QUA];
    float val_div_flux=0.0, sigma;

    /* FASE 01 : imprime mensagem de inicio de execucao do programa */
    inicio( );

    /* FASE 02 : verifica validade argumentos linha de comando */
    ProcssVeriArgu( &contador_arg, mtrz_ptr_arg, nom_arq_ent, nom_arq_sai, &qua_num_1, &qua_num_2,
        &num_iter_atenr, &dado_num_lin_col, &coluna, &linha, &num_qua, &sigma );

    /* FASE 03 : verifica dados obtidos na linha comando e calcula valor divisor fluxo */
    ProcssVeriDadCalDiv( &num_qua, &qua_num_1, &qua_num_2, &val_div_flux );

    /* FASE 04 : manipula arquivos */
    ProcssTrabArq( mtrz_ptr_arg, rotr_ent, rotr_sai, nom_arq_ent, nom_arq_sai, &qua_num_1, &qua_num_2,
        &num_iter_atenr, nom_comp_arq_img1, nom_comp_arq_img2 );

    /* FASE 05 : inicializa ponteiros-mascara-operadores */
    ProcssInicPtrMscrOpr( &registroC_img1, &registroC_img2, &registroD_img1, &registroD_img2,
        &nom_arq_reg1, &nom_arq_reg2, &mascr_1, &mascr_2, &sigma,
        &var_oper_BDT_x, &var_oper_BDT_xx, &var_oper_BDT_xy );

    /* FASE 06 : calcula valor do arranjo */
    ProcssCalcTamArj( &tam_arj, &coluna, &linha, &dado_num_lin_col, x, y,
        nom_comp_arq_img1, nom_comp_arq_img2 );

    /* FASE 07 : gere enderecos para registrar dados e calculos */
    ProcssCriaPtrBase( &registroC_img1, &registroC_img2, &registroD_img1, &registroD_img2,
        &tam_arj, x, y );

    /* FASE 08 : le - registra dados imagens entrada */
    ProcssLerRegDaEnt( nom_comp_arq_img1, nom_comp_arq_img2, CABE_IMG,
        registroD_img1->p_gaussn, registroD_img2->p_gaussn, &linha,
        &coluna, &tam_arj );

    /* FASE 09 : estima diferenca de gaussianos */
    ProcssDOG( &registroD_img1, &registroD_img2 );

    /* FASE 10 : estima Fluxo Otico */
    ProcssEstimaFluxo( &registroC_img1, &registroC_img2, &registroD_img1, &registroD_img2,
        &nom_arq_reg1, &nom_arq_reg2, &num_iter_atenr );

    /* FASE 11 : registra dados resultantes do Fluxo no arquivo de saida */
    ProcssRegiFluxoArq( &nom_arq_reg1, &nom_arq_reg2, nom_arq_sai, &val_div_flux );

    /* FASE 12 : imprime mensagem fim execucao do programa */
    fim( );
}

```

```

/*##### DECLARACAO DE FUNCOES #####*/
/*#####*/
NOME:
    inicio()

PROCESSO:
    Imprimir na tela mensagem de inicio da execucao do programa
    #####*/
    inicio()
    {
        printf( "\n\n%s\n", mensa_inici );
        printf("\n");
        printf( "\tFASE 01: Inicio do Programa\n" );
    }

/*#####*/
NOME:
    ProcssVeriArgu( contador_arg, mtrz_ptr_arg, nom_arq_ent, nom_arq_sai,
        qua_num_1, qua_num_2, num_iter_atenr,
        dado_num_lin_col, coluna, linha, p_num_qua )

PARAMETROS:
    contador_arg: contador do numero de argumentos
    mtrz_ptr_arg: matriz-ponteiro aponta valor argumentos
    nom_arq_ent: nome - arquivo de entrada
    nom_arq_sai: nome - arquivo de velocidade
    qua_num_1: quadro 1
    qua_num_2: quadro 2
    num_iter_atenr: numero de iteracoes de atenuacao
    dado_num_lin_col: arquivo de entrada sem cabecalho
        coluna: numero de colunas do quadro-imagem (arquivo de entrada)
        linha: numero de linhas do quadro-imagem (arquivo de entrada)
    num_qua: indicador do numero de quadros

PROCESSO:
    verificar a validade dos argumentos da linha de comando
    #####*/
    ProcssVeriArgu( p_contador_arg, mtrz_ptr_arg, nom_arq_ent, nom_arq_sai,
        p_qua_num_1, p_qua_num_2, p_num_iter_atenr,
        p_dado_num_lin_col, p_coluna, p_linha, p_num_qua, p_sigma )
    int *p_contador_arg;
    char *mtrz_ptr_arg[];
    string nom_arq_ent, nom_arq_sai;
    int *p_qua_num_1, *p_qua_num_2,
        *p_num_iter_atenr, *p_dado_num_lin_col, *p_linha, *p_coluna, *p_num_qua;
    float *p_sigma;
    {
        string roteiro_sai_fo, pas_str;
        int i=0;

        printf( "\n\tFASE 02: Processo verificar argumentos linha comando\n" );

        /* verificando numero apropriado de argumentos */
        if( (*p_contador_arg <= 6) || (*p_contador_arg > 13) ){
            erro(1);
        }

        i = 4;

        if( i < *p_contador_arg ){

            if( strcmp( "-QuIm", mtrz_ptr_arg[i] ) == 0 ){
                if( i + 2 < *p_contador_arg ){

                    sscanf( mtrz_ptr_arg[i+1], "%d", p_qua_num_1 );
                    sscanf( mtrz_ptr_arg[i+2], "%d", p_qua_num_2 );

                    /* atribuindo condicoes iniciais ate que opcoes sejam usadas */
                    *p_num_qua = VERDDR;
                    *p_num_iter_atenr = ITER_ATENR;
                    *p_dado_num_lin_col = 0;
                    *p_linha = 0;
                    *p_coluna = 0;
                    *p_sigma = SIGMA;
                }
            }
        }
    }

```

```

strcpy( nom_arq_ent, "Indefinido" );
strcpy( nom_arq_sai, "Indefinido" );

i += 3;
}
else{
    erro(1); /* erro: numero total de argumentos */
}
}
else{
    erro(2); /* erro: opcao invalida */
}
}
else{
    erro(11); /* erro indefinido */
}

while( i < *p_contador_arg ){

    if( strcmp( "-NLC", mtrz_ptr_arg[i] ) == 0 ){
        if( i + 2 < *p_contador_arg ){

            sscanf( mtrz_ptr_arg[i+1], "%d", p_linha );
            sscanf( mtrz_ptr_arg[i+2], "%d", p_coluna );

            *p_dado_num_lin_col = VERDDR;

            i += 3;
        }
        else{
            erro(1);
        }
    }
    else{
        if( strcmp( "-ltAt", mtrz_ptr_arg[i] ) == 0 ){
            if( i + 1 < *p_contador_arg ){
                sscanf( mtrz_ptr_arg[i+1], "%d", p_num_iter_atenr );

                i += 2;
            }
            else{
                erro(1);
            }
        }
        else{
            erro(2); /* erro: opcao invalida */
        }
    }
}

printf( "\tFASE 02: Processo verificar argumentos linha comando terminado\n" );

fflush( stdout );
}

/*****
NOME:
    erro( num_err )

PARAMETRO:
    num_err: numero do erro.

PROCESSO:
    informar que tipo de erro tem acontecido na execucao do programa
*****/
erro( num_err )
int num_err;
{
    switch( num_err ){
        case 1 : printf( "erro = %d\t: Erro numero de argumentos\n", num_err );
                  break;
        case 2 : printf( "erro = %d\t: Opcao invalida\n", num_err );
                  break;
    }
}

```

[illegible]


```
pp_qua_num_1, pp_qua_num_2, pp_num_iter_atenr )
```

PARAMETROS:

```
mtrz_ptr_arg: matriz-ponteiro aponta valor argumentos
rotr_ent: roteiro do diretorio dos dados de entra
rotr_sai: roteiro do diretorio para registrar arquivo saida
nom_arq_ent: nome - arquivo de entrada
nom_arq_sai: nome - arquivo de saida
qua_num_1: quadro 1
qua_num_2: quadro 2
num_iter_atenr: numero de iteracoes de atenuacao
```

PROCESSO:

```
gerar arquivo de saida para o fluxo otico
*****/
ger_arq_sai( mtrz_ptr_arg, rotr_ent, rotr_sai, nom_arq_ent, nom_arq_sai,
             pp_qua_num_1, pp_qua_num_2, pp_num_iter_atenr )
char *mtrz_ptr_arg[];
string rotr_ent, rotr_sai, nom_arq_ent, nom_arq_sai;
int **pp_qua_num_1, **pp_qua_num_2, **pp_num_iter_atenr;
{
    string roteiro_sai_fo, pas_str;

    /* objetivo: formar mtrz_ptr_arg[1] */
    concatenar( mtrz_ptr_arg[1], "/", rotr_ent );

    /* objetivo: formar mtrz_ptr_arg[1]/mtrz_ptr_arg[3] */
    concatenar( rotr_ent, mtrz_ptr_arg[3], nom_arq_ent );

    /* objetivo: formar roteiro -Qulm n:n */
    strcpy( roteiro_sai_fo, "-Qulm" );
    intr_a_str( **pp_qua_num_1, pas_str );
    concatenar( roteiro_sai_fo, pas_str, roteiro_sai_fo );
    concatenar( roteiro_sai_fo, ":", roteiro_sai_fo );
    intr_a_str( **pp_qua_num_2, pas_str );
    concatenar( roteiro_sai_fo, pas_str, roteiro_sai_fo );

    /* objetivo: formar -Qulm n:n -ItAt n */
    concatenar( roteiro_sai_fo, "-ItAt", roteiro_sai_fo );
    intr_a_str( **pp_num_iter_atenr, pas_str );
    concatenar( roteiro_sai_fo, pas_str, roteiro_sai_fo );

    /* objetivo: formar mtrz_ptr_arg[2] */
    concatenar( mtrz_ptr_arg[2], "/", rotr_sai );

    /* objetivo: formar mtrz_ptr_arg[2]/mtrz_ptr_arg[0] */
    concatenar( rotr_sai, mtrz_ptr_arg[0], nom_arq_sai );

    /* objetivo: formar mtrz_ptr_arg[2]/mtrz_ptr_arg[0]. */
    concatenar( nom_arq_sai, ".", nom_arq_sai );

    /* objetivo: formar mtrz_ptr_arg[2]/mtrz_ptr_arg[0].mtrz_ptr_arg[3] */
    concatenar( nom_arq_sai, mtrz_ptr_arg[3], nom_arq_sai );

    /* objetivo: formar mtrz_ptr_arg[2]/mtrz_ptr_arg[0].mtrz_ptr_arg[3]FO */
    concatenar( nom_arq_sai, "FO", nom_arq_sai );

    /* objetivo: formar mtrz_ptr_arg[2]/mtrz_ptr_arg[0].mtrz_ptr_arg[3]FO -Qulm n:n -ItAt n */
    concatenar( nom_arq_sai, roteiro_sai_fo, nom_arq_sai );
}

/*****
NOME:
concatenar( str_ent_1, str_ent_2, str_sai )

PARAMETROS:
str_ent_1: string de entrada 1
str_ent_2: string de entrada 2
str_sai: string de saida, concatena string 1 e 2, nessa ordem

PROCESSO:
concatenar os strings de entrada <str_ent_1> e <str_ent_2> no string de saida <str_sai>
*****/
concatenar( str_ent_1, str_ent_2, str_sai )
```

```

string str_ent_1, str_ent_2, str_sai;
{
    string str_intrfc;

    int i, j;

    for( i = 0; str_ent_1[i] != '\0'; i++ )
        str_intrfc[i] = str_ent_1[i];

    for( j = i; str_ent_2[j - i] != '\0'; j++ )
        str_intrfc[j] = str_ent_2[j - i];

    str_intrfc[j] = '\0';

    for( i = 0; i <= j; i++ ){
        str_sai[i] = str_intrfc[i];
    }
}

/*****
NOME:
    intr_a_str( val_intr_ent, str_sai )

PARAMETROS:
    val_intr_ent: valor do inteiro de entrada
    str_sai: string de saida

PROCESSO:
    converter valores inteiros em strings
*****/
intr_a_str( val_intr_ent, str_sai )
int val_intr_ent;
string str_sai;
{
    int i, j, val_resto, var_aux;
    string t;

    if( val_intr_ent < 0 ){
        var_aux = VERDDR;
    }
    else{
        var_aux = FALSO;
    }

    if( val_intr_ent == 0 ){
        str_sai[0] = '0';
        str_sai[1] = '\0';
    }
    else{
        i = 0;

        while( abs( val_intr_ent ) > 0 ){
            val_resto = abs( val_intr_ent ) % 10;
            t[i] = (char)( (int)'0' + val_resto );
            val_intr_ent = val_intr_ent / 10;
            i++;
        }

        if( var_aux ){
            t[i] = '-';
            i++;
        }

        for( j = 0; j < i; j++ ){
            str_sai[j] = t[i - j - 1];
        }

        str_sai[i] = '\0';
    }
}

/*****
NOME:
    pro_nom_arq( p_qua_num_1, p_qua_num_2, nom_arq_ent,

```



```

inic_oper_BDT( &p_var_oper_BDT_x, &p_var_oper_BDT_xx, &p_var_oper_BDT_xy );

printf( "\tFASE 05: Processo inicializar ponteiros, mascara, operador BDT terminado\n" );

fflush( stdout );
}

/*****
NOME:
inic_ptr( pp_registroC_img1, pp_registroD_img1, pp_registroC_img2, pp_registroD_img2,
          pp_nom_arq_reg1, pp_nom_arq_reg2 )

PARAMETROS:
pp_registroC_img1: ponteiro ponteiro - para calculos imagem 1
pp_registroD_img1: ponteiro ponteiro - para calculos imagem 1
pp_registroC_img2: ponteiro ponteiro - para trabalhar dados imagem 2
pp_registroD_img2: ponteiro ponteiro - pra trabalhar dados imagem 2
pp_nom_arq_reg1: ponteiro ponteiro - aponta nome arquivo registro 1
pp_nom_arq_reg2: ponteiro ponteiro - aponta nome arquivo registro 2

PROCESSO:
inicializar ponteiros com NULL
*****/
inic_ptr( pp_registroC_img1, pp_registroD_img1, pp_registroC_img2, pp_registroD_img2,
          pp_nom_arq_reg1, pp_nom_arq_reg2 )
p_base_dado **pp_registroC_img1, **pp_registroD_img1,
             **pp_registroC_img2, **pp_registroD_img2,
             **pp_nom_arq_reg1, **pp_nom_arq_reg2;
{
    inic_pontr( &pp_registroC_img1, &pp_registroD_img1 );

    inic_pontr( &pp_registroC_img2, &pp_registroD_img2 );

    inic_pontr( &pp_nom_arq_reg1, &pp_nom_arq_reg2 );
}

/*****
NOME:
inic_pontr( ppp_registroC, ppp_registroD )

PARAMETROS:
ppp_registroC: ponteiro do ponteiro ponteiro - aponta registro C
ppp_registroD: ponteiro do ponteiro ponteiro - aponta registro D

PROCESSO:
Inicializar com NULL os ponteiros registroC / D
inicializar com NULL os ponteiros nom_arq_reg_C / D
*****/
inic_pontr( ppp_registroC, ppp_registroD )
p_base_dado ***ppp_registroC, ***ppp_registroD;
{
    ***ppp_registroC = (p_base_dado)NULL; /* '\0' */
    ***ppp_registroD = (p_base_dado)NULL; /* '\0' */
}

/*****
NOME:
genr_mscr_gaussn_1( pp_mascr_1, pp_sigma )

PARAMETROS:
pp_mascr_1: ponteiro ponteiro - aponta mascara a
pp_sigma: ponteiro ponteiro - aponta valor desvio padrao

PROCESSO:
inicializar mascara 1
*****/
genr_mscr_gaussn_1( pp_mascr_1, pp_sigma )
mascara_dado **pp_mascr_1;
float **pp_sigma;
{
    int cont_mscr, faixa;

    faixa = (int)( **pp_sigma * 2 + 1 );

```

```

if( faixa % 2 == 0 ){
    faixa++;
}

for( cont_mscr= -(int)(faixa/2.0); cont_mscr <= (int)(faixa/2.0); cont_mscr++){

    (**pp_mascr_1).mscr[ cont_mscr + (int)(faixa/2.0) ] =
        exp( -pow( (float)cont_mscr, 2.0 )/( 2.0 * pow( **pp_sigma, 2.0 )))/( sqrt( (double)(2*PI) ) * **pp_sigma );

    (**pp_mascr_1).flut_mscr += (**pp_mascr_1).mscr[ cont_mscr + (int)(faixa/2.0) ];
}

(**pp_mascr_1).intr_mscr = faixa;
}

/*****
NOME:
    genr_mscr_gaussn_2( pp_mascr_2, pp_sigma )

PARAMETROS:
    pp_mascr_2: ponteiro ponteiro - aponta mascara a
    pp_sigma: ponteiro ponteiro - aponta valor desvio padrao

PROCESSO:
    inicializar mascara 2, tendo em conta a paroposta de Marr-Hildreth onde a relacao do
    padrao deve ser de ( sigma-a/sigma-b ) = 1.6
*****/
genr_mscr_gaussn_2( pp_mascr_2, pp_sigma )
mascara_dado **pp_mascr_2;
float **pp_sigma;
{
    int cont_mscr, faixa;
    float tx_sigma;

    tx_sigma = APRX_M_H * **pp_sigma;

    faixa = (int)( tx_sigma * 2 + 1 );

    if( faixa % 2 == 0 ){
        faixa++;
    }

    for( cont_mscr= -(int)(faixa/2.0); cont_mscr <= (int)(faixa/2.0); cont_mscr++){

        (**pp_mascr_2).mscr[ cont_mscr + (int)(faixa/2.0) ] =
            exp( -pow( (float)cont_mscr, 2.0 )/( 2.0 * pow( tx_sigma, 2.0 )))/( sqrt( (double)(2*PI) ) * tx_sigma );

        (**pp_mascr_2).flut_mscr += (**pp_mascr_2).mscr[ cont_mscr + (int)(faixa/2.0) ];
    }

    (**pp_mascr_2).intr_mscr = faixa;
}

/*****
NOME:
    inic_oper_BDT( pp_var_oper_BDT_x, pp_var_oper_BDT_xx, pp_var_oper_BDT_xy )

PARAMETROS:
    pp_var_oper_BDT_x: pp - aponta operador 3x3 para derivadas primeiro ordem respecto x
    pp_var_oper_BDT_xx: pp - aponta operador 3x3 para derivadas segundo ordem respecto x
    pp_var_oper_BDT_xy: pp - aponta operador 3x3 para derivadas parciais respecto x e y

FINALIDADE :
    inicializar operador de imagens definidos pelo BEAUDET a aplicar nas derivadas numericas
*****/
inic_oper_BDT( pp_var_oper_BDT_x, pp_var_oper_BDT_xx, pp_var_oper_BDT_xy )
operador_BDT_dado **pp_var_oper_BDT_x, **pp_var_oper_BDT_xx, **pp_var_oper_BDT_xy;
{
    (**pp_var_oper_BDT_x).OPRD_BDT[0][0] = -1.0;
    (**pp_var_oper_BDT_x).OPRD_BDT[1][0] = 0.0;
    (**pp_var_oper_BDT_x).OPRD_BDT[2][0] = 1.0;

    (**pp_var_oper_BDT_x).OPRD_BDT[0][1] = -1.0;
    (**pp_var_oper_BDT_x).OPRD_BDT[1][1] = 0.0;

```

[illegible]

```

switch( *p_dado_num_lin_col ){
/* procura valor numero filas-colunas no cabecalho imagem */
case 0 : /* procura valor numero fila-coluna no cabecalho imagem 1 */
    tam_raster( nom_comp_arq_img1, &x0, &y0, &p_tam_arrj );

    x[0] = x0;
    y[0] = y0;

    /* procura valor numero fila-coluna no cabecalho imagem 2 */
    tam_raster( nom_comp_arq_img2, &x1, &y1, &p_tam_arrj );

    x[1] = x1;
    y[1] = y1;

    /* verifica valores numero filas-colunas dos quadros de imagens */
    tam_valido( y, x );

    *p_linha = y0;
    *p_coluna = x0;

    break;

/* procura valor numero de linha-coluna atraves linha de comando */
case 1 : pp_tam_arrj = &p_tam_arrj;

    calc_tam_arrj( &p_linha, &p_coluna, &pp_tam_arrj );

    x[0] = *p_coluna;
    x[1] = *p_coluna;
    y[0] = *p_linha;
    y[1] = *p_linha;

    break;

default : erro(6); /* erro indefinido */
}

printf( "\tFASE 06: Processo calcular tamanho arranjo terminado\n" );

fflush( stdout );
}

/*****
NOME:
    tam_raster( nom_arq_raster, p_x, p_y, pp_tam_arrj )

PARAMETROS:
    nom_arq_raster: nome do arquivo - formato raster
    p_x: ponteiro - aponta numero de colunas do arquivo formato raster
    p_y: ponteiro - aponta numero de linhas do arquivo formato raster
    pp_tam_arrj: ponteiro ponteiro - aponta tamanho do arranjo ( potencia de 2 )

PROCESSO:
    abrir arquivo-imagem formato raster
    verificar abertura de arquivo
    ler cabecalho do arquivo para obter o valor numero fila-coluna
    chamar funcao para estimar o tamanho adequado do arranjo
    (tamanho arranjo deve ser potencia de 2)
*****/
tam_raster( nom_arq_raster, p_lin, p_col, pp_tam_arrj )
string nom_arq_raster;
int *p_lin, *p_col, **pp_tam_arrj;
{
    rasterf_dado cabe_img;
    int num_qua_img_ent, val_max;

    cabe_img.numr_linh = 0;
    cabe_img.numr_coln = 0;

    /* abrindo arquivo */
    num_qua_img_ent = open( nom_arq_raster, O_RDONLY );

    /* verificando abertura do arquivo */
    if( num_qua_img_ent < 0 ){

```

```

    erro(5);          /* erro abertura arquivo */
}

/* verificando - lendo cabeçalho (32bytes) */
if( read( num_qua_img_ent, &cabe_img, LER_CABE ) == LER_CABE ){
    *p_col = cabe_img.numr_coln;
    *p_lin = cabe_img.numr_linh;

    /* chamando funcao para o calculo do arranjo */
    calc_tam_arrj( &p_lin, &p_col, &pp_tam_arrj );
}
else{
    erro(4);          /* erro leitura cabeçalho */
}

close( num_qua_img_ent );
}

/*****
NOME:
    tam_valido( linhas, colunas )

PARAMETROS:
    linhas : tamanho da imagem quadros 1 e 2
    colunas : tamanho da imagem quadros 1 e 2

PROCESSO:
    verificar compatibilidade do tamanho dos quadros de imagens 1 e 2 ( linhas-colunas )
*****/
tam_valido( linhas, colunas )
int linhas[NUM_QUA], colunas[NUM_QUA];
{
    if( linhas[0] != linhas[1] || colunas[0] != colunas[1] ){
        erro(8);      /* tamanho da imagem nao-compativel */
    }
}

/*****
NOME:
    calc_tam_arrj( pp_num_linha, pp_num_coluna, ppp_tam_arrj )

PARAMETROS:
    pp_num_linha: ponteiro ponteiro - aponta numero de linhas do arquivo raster;
    pp_num_coluna: ponteiro ponteiro - aponta numero de colunas do arquivo raster;
    ppp_tam_arrj: ponteiro do ponteiro ponteiro - aponta tamanho do arranjo (potencia de 2)

PROCESSO:
    calcular o tamanho adequado do arranjo
    ( tamanho do arranjo deve ser potencia de 2 )
*****/
calc_tam_arrj( pp_num_linha, pp_num_coluna, ppp_tam_arrj )
int **pp_num_linha, **pp_num_coluna, ***ppp_tam_arrj;
{
    int val_max;

    /* asseguramento do valor maximo */
    if( **pp_num_coluna > **pp_num_linha ){
        val_max = **pp_num_coluna;
    }
    else{
        val_max = **pp_num_linha;
    }

    /* escolhendo tamanho do arranjo */
    if( val_max <= N10 && val_max > N9 ){
        ***ppp_tam_arrj = N10;      /* N10=1024 */
    }
    else{
        if( val_max <= N9 && val_max > N8 ){
            ***ppp_tam_arrj = N9;      /* N9=512 */
        }
        else{
            if( val_max <= N8 && val_max > N7 ){
                ***ppp_tam_arrj = N8;      /* N8=256 */
            }
        }
    }
}

```



```

    }
    else{
        if( val_max <= N7 && val_max > N6 ){
            ***ppp_tam_arrj = N7;          /* N7=128 */
        }
        else {
            erro(7);          /* tamanho da imagem deve estar entre 1024 e 128 */
        }
    }
}
}
}

/*~~~~~*/
NOME:
ProcssCriaPtrBase( p_registroC_img1, p_registroC_img2, p_registroD_img1, p_registroD_img2,
    p_tam_arrj, x, y )

PARAMETROS:
p_registroC_img1: ponteiro - para calculos imagem 1
p_registroC_img2: ponteiro - para calculos imagem 2
p_registroD_img1: ponteiro - para dados imagem 1
p_registroD_img2: ponteiro - para dados imagem 2
p_tam_arrj: ponteiro - aponta valor tamanho arranjo
x: tamanho da imagem em x dos quadros 1 e 2
y: tamanho da imagem em y dos quadros 1 e 2

PROCESSO:
criar ponteiros base para manipular calculos e dados imagens
~~~~~*/
ProcssCriaPtrBase( p_registroC_img1, p_registroC_img2, p_registroD_img1, p_registroD_img2,
    p_tam_arrj, x, y )
p_base_dado *p_registroC_img1, *p_registroC_img2, *p_registroD_img1, *p_registroD_img2;
int *p_tam_arrj;
int x[NUM_QUA], y[NUM_QUA];
{
    int x_0, x_1, y_0, y_1;

    x_0=x_1=y_0=y_1=0;

    printf( "\n\tFASE 07: Processo criar base ponteiros\n" );

    x_0 = x[0];
    y_0 = y[0];
    x_1 = x[1];
    y_1 = y[1];

    /* para imagem 1 */
    cria_ptr_base( &p_registroC_img1, &p_registroD_img1,
        *p_tam_arrj, &x_0, &y_0 );

    /* para imagem 2 */
    cria_ptr_base( &p_registroC_img2, &p_registroD_img2,
        *p_tam_arrj, &x_1, &y_1 );

    printf( "\tFASE 07: Processo criar base ponteiros terminado\n" );

    fflush( stdout );
}

/*~~~~~*/
NOME:
cria_ptr_base( pp_registroC_img, pp_registroD_img,
    resl_bas, p_tam_img_x_bas, p_tam_img_y_bas )

PARAMETROS:
pp_registroC_img: ponteiro - ponteiro aponta registro A da imagem
pp_registroD_img: ponteiro - ponteiro aponta registro B da imagem
resol_bas: valor resolucao base
p_tam_img_x_bas: ponteiro - aponta tamanho imagem em x da base
p_tam_img_y_bas: ponteiro - aponta tamanho imagem em y da base

PROCESSO:
produzir ponteiros para apontar base arranjo dados

```

```

    aplicar valor resolucao base
    fixar base registros A e B
    *****/
cria_ptr_base( pp_registroC_img, pp_registroD_img,
               resl_bas, p_tam_img_x_bas, p_tam_img_y_bas )
p_base_dado **pp_registroC_img, **pp_registroD_img;
int resl_bas, *p_tam_img_x_bas, *p_tam_img_y_bas;
{
    p_base_dado gera_end_base_reg_infr(),
    regis_ender_reser;
    int x_bas, y_bas;

    x_bas = *p_tam_img_x_bas; /* artificio */
    y_bas = *p_tam_img_y_bas; /* artificio */

    /* para armazenar endereco gerado */
    regis_ender_reser = gera_end_base_reg_infr( &resl_bas, &x_bas, &y_bas );

    /* para registrar enderecos */
    registro_end_resv( regis_ender_reser, &pp_registroC_img, &pp_registroD_img );

    /* para reservar memoria */
    resrv_memr_end_base( regis_ender_reser );
}

/*****
NOME:
    p_base_dado gera_end_base_reg_infr( p_resl_bs, p_x_bs, p_y_bs )

PARAMETROS:
    p_resl_bs: ponteiro - aponta resolucao base
    p_x_bs: ponteiro - aponta tamanho imagem base
    p_y_bs: ponteiro - aponta tamanho imagem base

PROCESSO:
    gera endereco para reservar memoria,
    registrar
        valor da resolucao, e
        tamanho numero linhas-colunas
    *****/
p_base_dado gera_end_base_reg_infr( p_resl_bs, p_x_bs, p_y_bs )
int *p_resl_bs, *p_x_bs, *p_y_bs;
{
    p_base_dado ender_reser;

    /* gera endereco */
    ender_reser = (p_base_dado)malloc( sizeof(base_dado) );

    if( ender_reser == NULL ){
        erro(11);          /* erro de alocao, ponteiro nulo */
    }
    else{
        /* registra dado com endereco gerado */
        ender_reser->resol = *p_resl_bs;
        ender_reser->tam_x = *p_x_bs;
        ender_reser->tam_y = *p_y_bs;
    }
    return( ender_reser );
}

/*****
NOME:
    registro_end_resv( registro_reserva, ppp_registroC, ppp_registroD )

PARAMETROS:
    reser_guar: ponteiro da reserva a guardar
    ppp_registroC: ponteiro do ponteiro ponteiro aponta registro A
    ppp_registroD: ponteiro do ponteiro ponteiro apontada registro B

PROCESSO:
    atribuir endereco gerado em ponteiros ( registro-C e D )
    *****/
registro_end_resv( registro_reserva, ppp_registroC, ppp_registroD )
p_base_dado registro_reserva, ***ppp_registroC, ***ppp_registroD;

```

[illegible]

```

{
printf( "\n\tFASE 08: Processo ler-registrar dados entrada\n" );

/* imagem 1 */
le_reg_dado_img( nom_comp_arq_img1, CABE_IMG, arrj_img_B1, &p_lin, &p_col, &p_tam_arrj );

/* imagem 2 */
le_reg_dado_img( nom_comp_arq_img2, CABE_IMG, arrj_img_B2, &p_lin, &p_col, &p_tam_arrj );

printf( "\tFASE 08: Processo ler-registrar dados entrada terminado\n" );

fflush( stdout );
}

/*****
NOME:
le_reg_dado_img( nome_arq, CABA_IMG, arrj_img, pp_linh, pp_coluna, pp_reso_arr )

PARAMETROS:
nome_arq: nome do arquivo
CABE_IMG: matriz cabecalho da imagem (formato raster-file)
arr_img: arranjo-matriz
pp_linh: ponteiro-ponteiro aponta numero de linhas
pp_coluna: ponteiro-ponteiro aponta numero de colunas
pp_reso_arr: ponteiro-ponteiro aponta valor resolucao arranjo

PROCESSO:
ler arquivo (formato rasterfile) e registrar os seus elementos no arranjo-matriz <arr_img[]>
*****/
le_reg_dado_img( nome_arq, CABE_IMG, arrj_img, pp_linh, pp_coluna, pp_reso_arr )
string nome_arq;
unsigned char CABE_IMG[B_CABE];
float arrj_img[N10*N10];
int **pp_linh, **pp_coluna, **pp_reso_arr;
{
unsigned char elem_img_chr;
int cont_lin, cont_col, num_iden_qua_ent, num_read;

num_iden_qua_ent = open( nome_arq, O_RDONLY );

/* verificar abertura arquivo */
if( num_iden_qua_ent < 0 )
erro(6);

/* ler cabecalho arquivo, intuito pular dados do cabecalho */
if( read( num_iden_qua_ent, CABE_IMG, B_CABE ) != B_CABE )
erro(4); /* erro leitura cabecalho */

/* ler e registrar elementos imagem */
for( cont_lin = 0; cont_lin < **pp_linh; cont_lin++){
for( cont_col = 0; cont_col < **pp_coluna; cont_col++){

num_read = read( num_iden_qua_ent, &elem_img_chr, sizeof(elem_img_chr) );

if( num_read != 1 ){
erro(3); /* erro leitura */
}
else{
arrj_img[ **pp_reso_arr * cont_lin + cont_col ] = (float)elem_img_chr;
}
}
}

close( num_iden_qua_ent );
}

/*****
NOME:
ProcssDOG( p_registroD_img1, p_registroD_img2 )

PARAMETROS:
p_registroD_img1: ponteiro - aponta registro dados imagem 1
p_registroD_img2: ponteiro - aponta registro dados imagem 2

```



```

int cont_lin, cont_col, tam_yy, tam_xx, cont_masc, calculo=0, res_bas, mscr_max;

float aux_a, aux_b, aux_c, aux_d, acumulador;

res_bas = (**pp_base_conv)->resol;
tam_xx = (**pp_base_conv)->tam_x;
tam_yy = (**pp_base_conv)->tam_y;

mscr_max = (*p_mascr).intr_mscr;
calcu_int = (float)mscr_max/2;

regis_ender_reser = gera_end_base_reg_infr( &res_bas, &tam_xx, &tam_yy );

reserv_gaussn( regis_ender_reser );

for( cont_lin = calculo_int; cont_lin < tam_yy - calculo_int; cont_lin++){
  for( cont_col = calculo_int; cont_col < tam_xx - calculo_int; cont_col++){

    acumulador = 0.0;

    for( cont_masc = 0; cont_masc < mscr_max; cont_masc++){

      calculo = cont_col + ( cont_masc - calculo_int );

      aux_a = (*p_mascr).mscr[cont_masc];
      aux_b = (**pp_base_conv)->p_gaussn[ res_bas * cont_lin + calculo ];

      acumulador += aux_a * aux_b;
    }

    (*regis_ender_reser->p_gaussn)[res_bas * cont_lin + cont_col] = acumulador/(*p_mascr).flut_mscr;
  }
}

for( cont_lin = calculo_int; cont_lin < tam_yy - calculo_int; cont_lin++){
  for( cont_col = calculo_int; cont_col < tam_xx - calculo_int; cont_col++){

    acumulador = 0.0;

    for( cont_masc = 0; cont_masc < mscr_max; cont_masc++){

      calculo = cont_lin + ( cont_masc - calculo_int );

      aux_c = (*p_mascr).mscr[cont_masc];
      aux_d = (*regis_ender_reser->p_gaussn)[ res_bas * calculo + cont_col ];

      acumulador += aux_c*aux_d;
    }

    (**pp_base_conv)->p_laplcn[res_bas * cont_lin + cont_col] = acumulador/(*p_mascr).flut_mscr;
  }
}

libr_memo_gaussn( regis_ender_reser );

free( regis_ender_reser );
}

/*****
NOME:
  calcula_GB( p_registroD )

PARAMETRO:
  p_registroD: ponteiro - aponta dados a convolucionar

PROCESSO:
  calcular gaussiano B
*****/
calcula_GB( p_registroD )
p_base_dado *p_registroD;
{
  extern mascara_dado mascr_l;
  p_base_dado regis_ender_reser, *p_regis_ender_reser;

```

```

int cont_lin, cont_col, res_bas, tam_xx, tam_yy;

res_bas = (*p_registroD)->resol;
tam_xx = (*p_registroD)->tam_x;
tam_yy = (*p_registroD)->tam_y;

regis_ender_reser = gera_end_base_reg_infr( &res_bas, &tam_xx, &tam_yy );

reserv_laplcn( regis_ender_reser );

regis_ender_reser->p_gaussn = (*p_registroD)->p_laplcn;

for( cont_lin = 0; cont_lin < tam_yy; cont_lin++){
    for( cont_col = 0; cont_col < tam_xx; cont_col++){

        if( ( cont_lin%2 == 0 ) || ( cont_col%2 == 0 ) ){
            ( *regis_ender_reser->p_gaussn )[ res_bas * cont_lin + cont_col ] = 0.0;
        }
    }
}

p_regis_ender_reser = &regis_ender_reser; /* artifício */

exect_convlc( &p_regis_ender_reser, &mascr_1 );

(*p_registroD)->p_laplcn = regis_ender_reser->p_laplcn;

libr_memo_gaussn( regis_ender_reser );

free( regis_ender_reser );
}

/*****
NOME:
    calcula_DOG( p_registroD )

PARAMETRO:
    p_registroD: ponteiro - aponta dados convolucionados

PROCESSO:
    calcular diferenca de gaussianos - DOG.
*****/
calcula_DOG( p_registroD )
p_base_dado *p_registroD;
{
    int cont_lin, cont_col;
    float aux_a, aux_b;

    for( cont_lin = 0; cont_lin < (*p_registroD)->tam_y; cont_lin++){
        for( cont_col = 0; cont_col < (*p_registroD)->tam_x; cont_col++){

            aux_a = ((*p_registroD)->p_gaussn)[ (*p_registroD)->resol * cont_lin + cont_col ];
            aux_b = ((*p_registroD)->p_laplcn)[ (*p_registroD)->resol * cont_lin + cont_col ];

            ( (*p_registroD)->p_laplcn )[ (*p_registroD)->resol * cont_lin + cont_col ] = aux_a - aux_b;
        }
    }
}

/*****
NOME:
    reserv_gaussn( p_base_dd )

PARAMETRO:
    p_base_dd: ponteiro - aponta reserva memoria

PROCESSO:
    reservar memoria para armazenar dados dos calculos
    o acesso do tamanho da reserva e funcao do valor da resolucao
    fixar a reserva no < p_base_dd -> gauss_ptr >
*****/
reserv_gaussn( p_base_dd )

```

```

p_base_dado p_base_dd;
{
switch( p_base_dd->resol ){
case N3 : p_base_dd -> p_gaussn = (img_1024 *)malloc( sizeof(img_8) );
break;
case N4 : p_base_dd -> p_gaussn = (img_1024 *)malloc( sizeof(img_16) );
break;
case N5 : p_base_dd -> p_gaussn = (img_1024 *)malloc( sizeof(img_32) );
break;
case N6 : p_base_dd -> p_gaussn = (img_1024 *)malloc( sizeof(img_64) );
break;
case N7 : p_base_dd -> p_gaussn = (img_1024 *)malloc( sizeof(img_128) );
break;
case N8 : p_base_dd -> p_gaussn = (img_1024 *)malloc( sizeof(img_256) );
break;
case N9 : p_base_dd -> p_gaussn = (img_1024 *)malloc( sizeof(img_512) );
break;
case N10 : p_base_dd -> p_gaussn = (img_1024 *)malloc( sizeof(img_1024) );
break;

default : erro(6);
break;
}
}

```

NOME:

reserv_laplcn(p_base_dd)

PARAMETRO:

p_base_dd: ponteiro - aponta reserva memoria

PROCESSO:

reservar memoria para armazenar dados dos calculos

o acesso do tamanho da reserva e funcao do valor da resolucao

*****/

reserv_laplcn(p_base_dd)

p_base_dado p_base_dd;

```

{
switch( p_base_dd->resol ){
case N3 : p_base_dd -> p_laplcn = (img_1024 *)malloc( sizeof(img_8) );
break;
case N4 : p_base_dd -> p_laplcn = (img_1024 *)malloc( sizeof(img_16) );
break;
case N5 : p_base_dd -> p_laplcn = (img_1024 *)malloc( sizeof(img_32) );
break;
case N6 : p_base_dd -> p_laplcn = (img_1024 *)malloc( sizeof(img_64) );
break;
case N7 : p_base_dd -> p_laplcn = (img_1024 *)malloc( sizeof(img_128) );
break;
case N8 : p_base_dd -> p_laplcn = (img_1024 *)malloc( sizeof(img_256) );
break;
case N9 : p_base_dd -> p_laplcn = (img_1024 *)malloc( sizeof(img_512) );
break;
case N10 : p_base_dd -> p_laplcn = (img_1024 *)malloc( sizeof(img_1024) );
break;

default : erro(6);
break;
}
}

```

NAME:

libr_memo_gaussn(p_base_dd)

PARAMETRO:

p_base_dd : ponteiro - libera memoria

PROCESSO:

liberar reserva memoria do arranjo-imagem gaussiana enderecada pelo ponteiro <p_base_dd>

*****/

libr_memo_gaussn(p_base_dd)

p_base_dado p_base_dd;

[illegible]

```

free( regis_ender_reser );

printf( "\tFASE 10: Processo produz Fluxo terminado\n" );

fflush( stdout );
}

/*****
NOME:
    reserv_camp_vetrl( p_base_dd )

PARAMETROS:
    p_base_dd: ponteiro - reserva memoria

PROCESSO:
    reservar memoria para registrar o campo vetorial - fluxo
    tamanho da reserva e indicada pelo tipo de resolucao
*****/
reserv_camp_vetrl( p_base_dd )
p_base_dado p_base_dd;
{
    switch( p_base_dd -> resol ){
        case N3 : p_base_dd -> p_vetrl = (camp_desl_1024 *)malloc( sizeof(camp_desl_8) );
            break;
        case N4 : p_base_dd -> p_vetrl = (camp_desl_1024 *)malloc( sizeof(camp_desl_16) );
            break;
        case N5 : p_base_dd -> p_vetrl = (camp_desl_1024 *)malloc( sizeof(camp_desl_32) );
            break;
        case N6 : p_base_dd -> p_vetrl = (camp_desl_1024 *)malloc( sizeof(camp_desl_64) );
            break;
        case N7 : p_base_dd -> p_vetrl = (camp_desl_1024 *)malloc( sizeof(camp_desl_128) );
            break;
        case N8 : p_base_dd -> p_vetrl = (camp_desl_1024 *)malloc( sizeof(camp_desl_256) );
            break;
        case N9 : p_base_dd -> p_vetrl = (camp_desl_1024 *)malloc( sizeof(camp_desl_512) );
            break;
        case N10 : p_base_dd -> p_vetrl = (camp_desl_1024 *)malloc( sizeof(camp_desl_1024) );
            break;

        default : erro(6);
            break;
    }
}

/*****
NOME:
    calcul_camp_fluxo( pp_registroC_img1, pp_registroC_img2, pp_registroD_img1, pp_registroD_img2,
        pp_nom_arq_reg1, pp_nom_arq_reg2, pp_num_iter_atenr )

PARAMETROS:
    pp_registroC_img1: ponteiro-ponteiro para calculos imagem 1
    pp_registroC_img2: ponteiro-ponteiro para calculos imagem 2
    pp_registroD_img1: ponteiro-ponteiro para dados imagem 1
    pp_registroD_img2: ponteiro-ponteiro para dados imagem 2
    pp_nom_arq_reg1: ponteiro-ponteiro aponta lista do campo
    pp_nom_arq_reg2: ponteiro-ponteiro aponta lista do campo
    pp_num_iter_atenr: ponteiro-ponteiro aponta numero de interacoes atenuacao

PROCESSO:
    calcular o campo d0 fluxo
    apagar ponteiros
*****/
calcul_camp_fluxo( pp_registroC_img1, pp_registroC_img2, pp_registroD_img1, pp_registroD_img2,
    pp_nom_arq_reg1, pp_nom_arq_reg2, pp_num_iter_atenr )
p_base_dado **pp_registroC_img1, **pp_registroC_img2, **pp_registroD_img1, **pp_registroD_img2,
    **pp_nom_arq_reg1, **pp_nom_arq_reg2;
int **pp_num_iter_atenr;
{
    vtr_deslocmt_dado calcul_mel_acop();
    int cont_lin, cont_col;
    float min_rmq=0.0;

    reserv_camp_param( **pp_nom_arq_reg2 );

```

```

for( cont_lin = 0; cont_lin < (**pp_nom_arq_reg2)->tam_y; cont_lin++){
  for( cont_col = 0; cont_col < (**pp_nom_arq_reg2)->tam_x; cont_col++){

    ( (**pp_nom_arq_reg2)->p_vetrl ) [ (**pp_nom_arq_reg2)->resol * cont_lin + cont_col ] =
      calcul_mel_acop( &pp_registroC_img1, &pp_registroC_img2, &min_rm, &cont_col, &cont_lin,
                      &cont_col, &cont_lin );

    calcul_curv_ang( &pp_registroC_img1, &pp_registroC_img2,
                    &pp_nom_arq_reg2, &pp_nom_arq_reg2, &cont_lin, &cont_col );
  }
}

libr_memo_laplcn( **pp_registroC_img1 ); /* libera memoria 1 */
libr_memo_laplcn( **pp_registroC_img2 ); /* libera memoria 2 */

apaga_base( **pp_registroC_img1, &pp_registroC_img1, &pp_registroD_img1 );
apaga_base( **pp_registroC_img2, &pp_registroC_img2, &pp_registroD_img2 );

atenuar( &pp_nom_arq_reg2, &pp_nom_arq_reg2, &pp_num_iter_atenr);
}

/*****
NOME:
  reserv_camp_param( p_base_dd )

PARAMETROS:
  p_base_dd: ponteiro - reserva memoria

PROCESSO:
  registrar reserva de memoria para o campo parametrico
  tamanho da reserva e indicada pelo tipo de resolucao
*****/
reserv_camp_param( p_base_dd )
p_base_dado p_base_dd;
{
  switch( p_base_dd->resol ){
    case N3 : p_base_dd->p_param = (param_1024 *)malloc( sizeof(param_8) );
              break;
    case N4 : p_base_dd->p_param = (param_1024 *)malloc( sizeof(param_16) );
              break;
    case N5 : p_base_dd->p_param = (param_1024 *)malloc( sizeof(param_32) );
              break;
    case N6 : p_base_dd->p_param = (param_1024 *)malloc( sizeof(param_64) );
              break;
    case N7 : p_base_dd->p_param = (param_1024 *)malloc( sizeof(param_128) );
              break;
    case N8 : p_base_dd->p_param = (param_1024 *)malloc( sizeof(param_256) );
              break;
    case N9 : p_base_dd->p_param = (param_1024 *)malloc( sizeof(param_512) );
              break;
    case N10 : p_base_dd->p_param = (param_1024 *)malloc( sizeof(param_1024) );
              break;

    default : erro(6);
              break;
  }
}

/*****
NOME:
  calcul_mel_acop( ppp_registroC_img1, ppp_registroC_img2, p_val_rm,
                  p_coXpix_img1, p_coYpix_img1, p_coXpix_img2, p_coYpix_img2 )

PARAMETROS:
  ppp_registroC_img1: ppp - aponta registro dados convolucionados
  ppp_registroC_img2: ppp - aponta registro dados convolucionados
  p_val_rm: p - aponta valor sqd onde sera centrado o melhor igualamento
               calculado a partir duma area 3x3 em redor das coordenadas
               (p_coXpix_img2,p_coYpix_img1)
  coXpix_img1: coordenada x img1 ( pixel de referencia )
  coYpix_img1: coordenada y img1 ( pixel de referencia )
  coXpix_img2: coordenadas x img2, centrado na area nxn, para o calculo rmq
               referente ao pixel (x,y) da img1
  coYpix_img2: coordenadas y img2, centrado na area nxn, para o calculo rmq

```

referente ao pixel (x, y) img1

PROCESSO:

```

encontrar melhor casamento entre o pixel ( coXpix_img1, coYpix_img1 ) e o pixel
(coXpix_img2, coYpix_img2) focalizados na area nxn da img2
*****/
vtr_deslocmt_dado calcula_mel_acop( ppp_registroC_img1, ppp_registroC_img2, p_val_rm,
    p_coXpix_img1, p_coYpix_img1, p_coXpix_img2, p_coYpix_img2 )
p_base_dado ***ppp_registroC_img1, ***ppp_registroC_img2;
float *p_val_rm;
int *p_coXpix_img1, *p_coYpix_img1, *p_coXpix_img2, *p_coYpix_img2;
{
    vtr_deslocmt_dado PTR_DSCL[VTR_ACU_RMQ], DD_RTRN;

    int var_cont_x=0, var_cont_y=0, var_cont=0, area_nxn=0, contador_1=0, contador_2=0,
        vlr_fnl_cont=0, var_aux_xx, var_aux_yy, tam_xx, tam_yy;

    float calc_raiz_min_qua(),
        REG_RMQ[VTR_ACU_RMQ]; /* VTR_ACU_RMQ = AREA*AREA */

    tam_xx = (**ppp_registroC_img1)->tam_x;
    tam_yy = (**ppp_registroC_img2)->tam_y;

    area_nxn = var_oper_BDT_x.intr_oprd_BDT/2;

    *p_val_rm = LIM_MIN;

    /* procura na area nxn, em redor das coordenadas ( p_coXpix_img2, p_coYpix_img2 ) */
    for( var_cont_x = -area_nxn; var_cont_x <= area_nxn; var_cont_x++){
        for( var_cont_y = -area_nxn; var_cont_y <= area_nxn; var_cont_y++){

            var_aux_xx = *p_coXpix_img2 + var_cont_x;
            var_aux_yy = *p_coYpix_img2 + var_cont_y;

            /* verifica-se valores pixels estao no range [ 0 <-> #lc ] */
            if( retrnr_VRD_FLS( &var_aux_xx, &var_aux_yy, &tam_xx, &tam_yy ) ){

                /* registra os valores rmq */
                REG_RMQ[contador_1] = calcula_raiz_min_qua( &ppp_registroC_img1, &ppp_registroC_img2,
                    &p_coXpix_img1, &p_coYpix_img1, &var_aux_xx, &var_aux_yy );

                PTR_DSCL[contador_1].vtr_desl_x = var_aux_xx - *p_coXpix_img1;
                PTR_DSCL[contador_1].vtr_desl_y = var_aux_yy - *p_coYpix_img1;

                if( REG_RMQ[contador_1] < *p_val_rm ){
                    *p_val_rm = REG_RMQ[contador_1];
                    vlr_fnl_cont = contador_1;
                }

                contador_1++;
            }
        }
    }

    contador_2 = 1;

    DD_RTRN = PTR_DSCL[vlr_fnl_cont];

    for( var_cont = 0; var_cont < contador_1; var_cont++){ /* calcula variacao dos valores rmq */

        if( (var_cont != vlr_fnl_cont) && (fabs( REG_RMQ[vlr_fnl_cont] - REG_RMQ[var_cont] ) == 0.0 ) ){
            DD_RTRN.vtr_desl_x += PTR_DSCL[var_cont].vtr_desl_x;
            DD_RTRN.vtr_desl_y += PTR_DSCL[var_cont].vtr_desl_y;
            contador_2++;
        }
    }

    DD_RTRN.vtr_desl_x = DD_RTRN.vtr_desl_x/(float)contador_2;
    DD_RTRN.vtr_desl_y = DD_RTRN.vtr_desl_y/(float)contador_2;

    return( DD_RTRN );
}

/*****

```

NOME:

```
retrnr_VRD_FLS( p_x, p_y, p_tam_xx, p_tam_yy )
```

PARAMETROS:

p_x : ponteiro - aponta coordenada x imagem
 p_y : ponteiro - aponta coordenada y imagem
 p_tam_xx : ponteiro - aponta valor numero colunas
 p_tam_yy : ponteiro - aponta valor numero linhas

PROCESSO:

retornar o valor verdadeiro ou falso de (x, y) da imagem de tamanho (tam_xx * tam_yy)

```
*****/
int retrnr_VRD_FLS( p_x, p_y, p_tam_xx, p_tam_yy )
int *p_x, *p_y, *p_tam_xx, *p_tam_yy;
{
  return( ( (*p_x >= 0) && (*p_x < *p_tam_xx) && (*p_y >= 0) && (*p_y < *p_tam_yy) ) );
}
```

NOME :

```
calc_raiz_min_qua( pppp_registroC_img1, pppp_registroC_img2,
  pp_coXpix_img1, pp_coXpix_img1, p_coXpix_img2, p_coYpix_img2 )
```

PARAMETROS:

pppp_registroC_img1: ppp - para registro img1
 contine o pixel (coXpix_img1, coYpix_img1)
 pppp_registroC_img2: ppp - para registro img2
 contine o pixel (coXpix_img2, coYpix_img2)
 pp_coXpix_img1: pp - aponta coordenada x do pixel na imagem-1
 pp_coYpix_img1: pp - aponta coordenada y do pixel na imagem-1
 p_coXpix_img2: p - aponta coordenada x do pixel na imagem-2
 p_coYpix_img2: p - aponta coordenada y do pixel na imagem-2

PROCESSO:

calcular raiz minima quadrada nas diferencias entre os valores dos pontos apontados
 pelos ponteiros (pp_coXpix_img1, pp_coXpix_img1) e (p_coXpix_img2, p_coYpix_img2)

```
float calc_raiz_min_qua( pppp_registroC_img1, pppp_registroC_img2,
  pp_coXpix_img1, pp_coYpix_img1, p_coXpix_img2, p_coYpix_img2 )
p_base_dado ****pppp_registroC_img1, ****pppp_registroC_img2;
int **pp_coXpix_img1, **pp_coYpix_img1, *p_coXpix_img2, *p_coYpix_img2;
{
  extern mascara_dado mascr_2;

  int var_cont_1=0, var_cont_2=0, ll_1=0, cc_1=0, ll_2=0, cc_2=0, reslc;

  float acumulador_sm=0.0, mul_1=0.0, mul_2=0.0;

  double base, retornar;

  reslc = (****pppp_registroC_img1)->resol;

  for( var_cont_1 = -mascr_2.intr_mscr/2; var_cont_1 <= mascr_2.intr_mscr/2; var_cont_1++){

    ll_1 = **pp_coYpix_img1 + var_cont_1;
    ll_2 = *p_coYpix_img2 + var_cont_1;

    for( var_cont_2 = -mascr_2.intr_mscr/2; var_cont_2 <= mascr_2.intr_mscr/2; var_cont_2++){

      cc_1 = **pp_coXpix_img1 + var_cont_2;
      cc_2 = *p_coXpix_img2 + var_cont_2;

      mul_1 = (float)mascr_2.mscr[ var_cont_1 + mascr_2.intr_mscr/2 ];
      mul_2 = (float)mascr_2.mscr[ var_cont_2 + mascr_2.intr_mscr/2 ];

      base = (double)( ( ****pppp_registroC_img1)->p_laplcn )[ reslc * ll_1 + cc_1 ] -
        ( ****pppp_registroC_img2)->p_laplcn )[ reslc * ll_2 + cc_2 ] );

      acumulador_sm += mul_1 * mul_2 * pow( base, EXPONNT );
    }
  }

  retornar = sqrt( (double)(acumulador_sm) ) / (mascr_1.flut_mscr * mascr_1.flut_mscr);
  return( retornar );
```

```

}

/*****
NOME:
    calcul_curv_ang( ppp_registroC_img1, ppp_registroC_img2,
                    ppp_base_dd_camp_desl, ppp_camp_param_desl,
                    p_l, p_c )

PRAMETROS:
    ppp_registroC_img1: ppp - aponta registro C imagem_1
    ppp_registroC_img2: ppp - aponta registro C imagem_2
    ppp_base_dd_camp_desl: ppp - aponta base do campo de deslocamento
    ppp_camp_param_desl: ppp - aponta campo parametrico do campo de deslocamento
    p_l: p - aponta coordenadas y do campo de deslocamento
           ( para o qual o calculo e executado )
    p_c: p - aponta coordenadas x do campo de deslocamento
           ( para o qual o calculo e executado )

PROCESSO:
    calcular
        curvatura principal da superficie
        angulo rotacional dos eixos valores fundamentais do espaco (x,y)
*****/
calcul_curv_ang( ppp_registroC_img1, ppp_registroC_img2,
                ppp_base_dd_camp_desl, ppp_camp_param_desl,
                p_l, p_c )
p_base_dado ***ppp_registroC_img1, ***ppp_registroC_img2,
***ppp_base_dd_camp_desl, ***ppp_camp_param_desl;
int *p_l, *p_c;
{
    vtr_deslocmt_dado rotar_vtr(), acss_vtr_a, acss_vtr_b;
    param_dado acss_angl;
    area_dado acss_area;

    extern operador_BDT_dado var_oper_BDT_x;

    int calcular, lim_lado_area, i, j, ll, cc, reslc, tam_xx, tam_yy;

    float calcul_dervd_x(), calcul_dervd_y(), calcul_dervd_xx(), calcul_dervd_yy(), calcul_dervd_xy(),
        val_dervd_x=0.0, val_dervd_y=0.0, val_dervd_xx=0.0, val_dervd_yy=0.0, val_dervd_xy=0.0,
        curvtr_max=0.0, curvtr_min=0.0;

    reslc = (**ppp_base_dd_camp_desl)->resol;
    tam_xx = (**ppp_base_dd_camp_desl)->tam_x;
    tam_yy = (**ppp_base_dd_camp_desl)->tam_y;

    calcular = FALSO;

    lim_lado_area = var_oper_BDT_x.intr_oprd_BDT/2;

    acss_vtr_a = (**ppp_base_dd_camp_desl)->p_vetr1[ reslc * *p_l + *p_c ];

    for( i = -lim_lado_area; i <= lim_lado_area; i++ ){ /* calculo area 3x3 */
        for( j = -lim_lado_area; j <= lim_lado_area; j++ ){

            ll = *p_l + (int)acss_vtr_a.vtr_desl_y + i;
            cc = *p_c + (int)acss_vtr_a.vtr_desl_x + j;

            if( retnr_VRD_FLS( &cc, &ll, &tam_xx, &tam_yy ) ){

                acss_area.AREA_DD[ i + lim_lado_area ][ j + lim_lado_area ] =
                    calc_raiz_min_qua( &ppp_registroC_img1, &ppp_registroC_img2, &p_c, &p_l, &cc, &ll );
            }
            else{
                acss_area.AREA_DD[ i + lim_lado_area ][ j + lim_lado_area ] = -1.0;
                calcular = VERDDR;
            }
        }
    }
}

if( !calcular ){
    /* executa-se derivadas usando os operadores-imagem de Beaudet */
    val_dervd_x = calcul_dervd_x(&acss_area);

```

```

val_dervd_y = calcul_dervd_y(&acss_area);
val_dervd_xx = calcul_dervd_xx(&acss_area);
val_dervd_yy = calcul_dervd_yy(&acss_area);
val_dervd_xy = calcul_dervd_xy(&acss_area);

/* calcula curvatura principal ao longo de ( Emin, Emax ) */
if( (val_dervd_xx - val_dervd_yy)*(val_dervd_xx - val_dervd_yy) + 4.0*val_dervd_xy*val_dervd_xy >= 0.0 ){
    curvtr_max =
        ((val_dervd_yy + val_dervd_xx) +
         sqrt((val_dervd_xx - val_dervd_yy)*(val_dervd_xx - val_dervd_yy) + 4.0*val_dervd_xy*val_dervd_xy))/2;

    curvtr_min =
        ((val_dervd_yy + val_dervd_xx) -
         sqrt((val_dervd_xx - val_dervd_yy)*(val_dervd_xx - val_dervd_yy) + 4.0*val_dervd_xy*val_dervd_xy))/2;
}
else {
    curvtr_max = 0.0;
    curvtr_min = 0.0;
}

acss_angl.parm_cur_max = curvtr_max/( CONST_DIV + acss_area.AREA_DD[ lim_lado_area ][ lim_lado_area ] );
acss_angl.parm_cur_min = curvtr_min/( CONST_DIV + acss_area.AREA_DD[ lim_lado_area ][ lim_lado_area ] );

/* calcula angulo */
acss_angl.parm_angl = atan( curvtr_min - val_dervd_xx, val_dervd_xy );

/* angulo de Emin para o eixo x */
rotar_vtr_xy( &val_dervd_x, &val_dervd_y, acss_angl.parm_angl ); /* derivado ao longo de ( Emin, Emax ) */

if( curvtr_min != 0.0 ){
    acss_vtr_b.vtr_desl_x = val_dervd_x/curvtr_min; /* determina numero extremo ao longo do lado A */
}
else{
    acss_vtr_b.vtr_desl_x = 0.0;
}

if( curvtr_max != 0.0 ){
    acss_vtr_b.vtr_desl_y = val_dervd_y/curvtr_max; /* determina numero extremo ao longo do lado B */
}
else {
    acss_vtr_b.vtr_desl_y = 0.0;
}

if( (fabs(acss_vtr_b.vtr_desl_x) > 1.0) || (curvtr_min <= 0.0) ){
    acss_vtr_b.vtr_desl_x = 0.0;
    acss_angl.parm_cur_min = 0.0;
}

if( (fabs(acss_vtr_b.vtr_desl_y) > 1.0) || (curvtr_max <= 0.0) ){
    acss_vtr_b.vtr_desl_y = 0.0;
    acss_angl.parm_cur_max = 0.0;
}

( (**ppp_base_dd_camp_desl)->p_vetri )[ reslc * *p_l + *p_c ] = acss_vtr_a;
( (**ppp_camp_param_desl)->p_param )[ reslc * *p_l + *p_c ] = acss_angl;
}
else{
    acss_vtr_a.vtr_desl_x = 0.0;
    acss_vtr_a.vtr_desl_y = 0.0;

    ( (**ppp_base_dd_camp_desl)->p_vetri )[ reslc * *p_l + *p_c ] = acss_vtr_a;

    acss_angl.parm_cur_min = 0.0;
    acss_angl.parm_cur_max = 0.0;

    ( (**ppp_camp_param_desl)->p_param )[ reslc * *p_l + *p_c ] = acss_angl;
}
}

/*****
NOME:
calcul_dervd_x( val_cal_der )

```

PARAMETROS:

val_cal_der: valor calculo da derivada

PROCESSO:

calcular primeira derivada parcial respecto a x, localizado no centro da pequena area

*****/

float calcul_dervd_x(val_cal_der)

area_dado *val_cal_der;

{

extern operador_BDT_dado var_oper_BDT_x;

int i, j;

float acumulador_soma;

acumulador_soma = 0.0;

for(i = 0; i < var_oper_BDT_x.intr_oprd_BDT; i++){

for(j = 0; j < var_oper_BDT_x.intr_oprd_BDT; j++){

acumulador_soma += var_oper_BDT_x.OPRD_BDT[i][j] * (*val_cal_der).AREA_DD[i][j];

}

}

return(acumulador_soma / var_oper_BDT_x.flut_oprd_BDT);

}

*****/

NOME:

calcul_dervd_y(val_cal_der)

PARAMETROS:

val_cal_der: valor calculo da derivada

PROCESSO:

calcular primeira derivada parcial respecto a y, localizada no centro da pequena area

(usa-se a transposta de var_oper_BDT_x)

*****/

float calcul_dervd_y(val_cal_der)

area_dado *val_cal_der;

{

extern operador_BDT_dado var_oper_BDT_x;

int i, j;

float acumulador_soma;

acumulador_soma = 0.0;

for(i = 0; i < var_oper_BDT_x.intr_oprd_BDT; i++){

for(j = 0; j < var_oper_BDT_x.intr_oprd_BDT; j++){

acumulador_soma += var_oper_BDT_x.OPRD_BDT[j][i] * (*val_cal_der).AREA_DD[i][j];

}

}

return(acumulador_soma/var_oper_BDT_x.flut_oprd_BDT);

}

*****/

NOME :

calcul_dervd_xx(val_cal_der)

PARAMETROS:

val_cal_der: valor calculo da derivada

PROCESSO:

calcular segunda derivada parcial respecto a x, localmente colocada no centro da pequena area

*****/

float calcul_dervd_xx(val_cal_der)

area_dado *val_cal_der;

{

extern operador_BDT_dado var_oper_BDT_xx;

int i, j;

float acumulador_soma;

acumulador_soma = 0.0;


```

for( i = 0; i < var_oper_BDT_xx.intr_oprd_BDT; i++){
  for( j = 0; j < var_oper_BDT_xx.intr_oprd_BDT; j++){
    acumulador_soma += var_oper_BDT_xx.OPRD_BDT[i][j]*(*val_cal_der).AREA_DD[i][j];
  }
}

return( acumulador_soma/var_oper_BDT_xx.flut_oprd_BDT );
}

/*****
NOME:
  calcul_dervd_yy( val_cal_der )

PARAMETROS:
  val_cal_der: valor calculo da derivada

PROCESSO:
  calcular segunda derivada parcial respecto a y, localmente colocada no centro da pequena area
  ( usa-se transposta de var_oper_BDT_xx )
*****/
float calcul_dervd_yy( val_cal_der )
area_dado *val_cal_der;
{
  extern operador_BDT_dado var_oper_BDT_xx;
  int i, j;
  float acumulador_soma;

  acumulador_soma = 0.0;

  for( i = 0; i < var_oper_BDT_xx.intr_oprd_BDT; i++){
    for( j = 0; j < var_oper_BDT_xx.intr_oprd_BDT; j++){
      acumulador_soma += var_oper_BDT_xx.OPRD_BDT[j][i]*(*val_cal_der).AREA_DD[i][j];
    }
  }

  return( acumulador_soma/var_oper_BDT_xx.flut_oprd_BDT );
}

/*****
NOME:
  calcul_dervd_xy( val_cal_der )

PARAMETROS:
  val_cal_der: valor calculo da derivada

PROCESSO:
  calcular segunda derivada parcial respecto x e y, localmente colocada no centro da pequena area
*****/
float calcul_dervd_xy( val_cal_der )
area_dado *val_cal_der;
{
  extern operador_BDT_dado var_oper_BDT_xy;
  int i, j;
  float acumulador_soma;

  acumulador_soma = 0.0;

  for( i = 0; i < var_oper_BDT_xy.intr_oprd_BDT; i++){
    for( j = 0; j < var_oper_BDT_xy.intr_oprd_BDT; j++){
      acumulador_soma += var_oper_BDT_xy.OPRD_BDT[i][j]*(*val_cal_der).AREA_DD[i][j];
    }
  }

  return( acumulador_soma/var_oper_BDT_xy.flut_oprd_BDT );
}

/*****
NOME:
  rotar_vtr_xy( x, y, ang )

PARAMETROS:
  x: coordenada x do vetor
  y: coordenada y do vetor
  ang: angulo ( radianos )

```

PROCESSO:

```

rotar o vetor ( x, y ) o valor do angulo "ang" ( radianos )
*****/
rotar_vtr_xy( x, y, ang )
float *x, *y, ang;
{
    float var_aux_x, var_aux_y;

    var_aux_x = *x;
    var_aux_y = *y;

    *x = var_aux_x * cos( ang ) + var_aux_y * sin( ang );
    *y = -var_aux_x * sin( ang ) + var_aux_y * cos( ang );
}

```

```

/*****

```

NOME:

```

    libr_memo_laplcn( p_base_dd )

```

PARAMETRO:

```

    p_base_dd: ponteiro base_dd

```

FINALIDADE:

```

    liberar reserva memoria para imagem-laplaciana enderecada pelo ponteiro <p_base_dd>

```

```

*****/

```

```

libr_memo_laplcn( p_base_dd )
p_base_dado p_base_dd;
{
    switch( p_base_dd -> resol ){
        case N3 : free( (img_8 *)p_base_dd -> p_laplcn );
            break;
        case N4 : free( (img_16 *)p_base_dd -> p_laplcn );
            break;
        case N5 : free( (img_32 *)p_base_dd -> p_laplcn );
            break;
        case N6 : free( (img_64 *)p_base_dd -> p_laplcn );
            break;
        case N7 : free( (img_128 *)p_base_dd -> p_laplcn );
            break;
        case N8 : free( (img_256 *)p_base_dd -> p_laplcn );
            break;
        case N9 : free( (img_512 *)p_base_dd -> p_laplcn );
            break;
        case N10 : free( (img_1024 *)p_base_dd -> p_laplcn );
            break;

        default : erro(6);
            break;
    }
}

```

```

/*****

```

NAME:

```

    apaga_base( p_base_apa, p_registro_A_img, ppp_registro_B_img )

```

PARAMETROS:

```

    p_base_apa: p - aponta dado base a ser apagada
    p_registroC_img: p - aponta registro C
    ppp_registroD_img: ppp - aponta registro D

```

PROCESSO:

```

    apagar base apontada pelo ponteiro < p_base_apa >

```

```

*****/

```

```

apaga_base( p_base_apa, ppp_registroC_img, ppp_registroD_img )
p_base_dado p_base_apa, ***ppp_registroC_img, ***ppp_registroD_img;
{
    ***ppp_registroC_img = (p_base_dado)NULL;
    ***ppp_registroD_img = (p_base_dado)NULL;

    free( p_base_apa );
}

```

```

/*****

```

NOME:

atenuar(ppp_base_camp_vetrl, ppp_base_camp_param, ppp_num_iter_atenr)

PARAMETROS:

ppp_base_camp_vetrl: ppp - aponta base campo vetorial

ppp_base_camp_param: ppp - aponta base campo parametrico

ppp_num_iter_aten: ppp - aponta numero de iteracoes de atenuacao

FINALIDADE:

atenuar, de forma restringida, o campo vetorial

usa-se o algoritmo de atenuacao de Horn - Schunck

*****/

atenuar(ppp_base_camp_vetrl, ppp_base_camp_param, ppp_num_iter_atenr)

p_base_dado ***ppp_base_camp_vetrl, ***ppp_base_camp_param;

int ***ppp_num_iter_atenr;

{

p_base_dado regis_ender_reser;

vtr_deslocmt_dado rotar_vtr(), aux_var_cont_iter, var_aux_acs, Emax, Emin;

param_dado var_acs_parm;

int num_lin, num_col, cont_lin, cont_col, var_cont_iter, contdr_divsr, resl_base;

float u_val(), v_val(), val_u, val_v;

resl_base = (**ppp_base_camp_vetrl)->resol;

num_lin = (**ppp_base_camp_vetrl)->tam_y;

num_col = (**ppp_base_camp_vetrl)->tam_x;

regis_ender_reser = gera_end_base_reg_infr(&resl_base, &num_col, &num_lin);

reserv_camp_vetrl(regis_ender_reser);

/* inicializar */

for(cont_lin = 0; cont_lin < num_lin; cont_lin++){

for(cont_col = 0; cont_col < num_col; cont_col++){

aux_var_cont_iter.vtr_desl_x = 0.0;

aux_var_cont_iter.vtr_desl_y = 0.0;

}

}

/* atribuir */

for(cont_lin = 0; cont_lin < num_lin; cont_lin++){

for(cont_col = 0; cont_col < num_col; cont_col++){

(*regis_ender_reser->p_vetrl)[resl_base * cont_lin + cont_col] =

((**ppp_base_camp_vetrl)->p_vetrl)[resl_base * cont_lin + cont_col];

}

}

for(var_cont_iter = 0; var_cont_iter < ***ppp_num_iter_atenr; var_cont_iter++){

for(cont_lin = 0; cont_lin < num_lin; cont_lin++){

for(cont_col = 0; cont_col < num_col; cont_col++){

val_u = u_val(®is_ender_reser, &num_lin, &num_col);

val_v = v_val(®is_ender_reser, &num_lin, &num_col);

((**ppp_base_camp_vetrl)->p_vetrl)[resl_base * cont_lin + cont_col].vtr_desl_x =

val_u - var_acs_parm.parm_cur_min *

(var_acs_parm.parm_cur_min * val_u + var_acs_parm.parm_cur_max * val_v) /

(pow(var_acs_parm.parm_cur_min, 2.0) + pow(var_acs_parm.parm_cur_max, 2.0) + pow(ALFA, 2.0));

((**ppp_base_camp_vetrl)->p_vetrl)[resl_base * cont_lin + cont_col].vtr_desl_y =

val_v - var_acs_parm.parm_cur_max *

(var_acs_parm.parm_cur_min * val_u + var_acs_parm.parm_cur_max * val_v) /

(pow(var_acs_parm.parm_cur_min, 2.0) + pow(var_acs_parm.parm_cur_max, 2.0) + pow(ALFA, 2.0));

}

}

}

for(cont_lin=0; cont_lin < num_lin; cont_lin++){

for(cont_col=0; cont_col < num_col; cont_col++){

((**ppp_base_camp_vetrl)->p_vetrl)[resl_base * cont_lin + cont_col] =

```

        (*regis_ender_reser->p_vetrl)[ resl_base * cont_lin + cont_col ];
    }
}

libr_reserv_camp_vetrl( regis_ender_reser );

free( regis_ender_reser );
}

/*****
NOME:
    u_val( p_regis_ender_reser, p_num_lin, p_num_col )

PARAMETROS:
    p_regis_ender_reser_B: ponteiro - aponta valores vetoriais
    p_num_lin: ponteiro - aponta valor y para calculo
    p_num_col: ponteiro - aponta valor x para calculo

PROCESSO:
    calcula os valores de uxx + uyy segundo o metodo de atenuacao de Horn-Schunck
*****/
float u_val( p_regis_ender_reser, p_num_lin, p_num_col )
p_base_dado *p_regis_ender_reser;
int *p_num_lin, *p_num_col;
{
    int reslc;
    float valor;

    reslc = (*p_regis_ender_reser)->resol;

    valor = (1.0/6.0) * (
        (*p_regis_ender_reser->p_vetrl)[ reslc * ( *p_num_lin - 1 ) + *p_num_col ].vtr_desl_x +
        (*p_regis_ender_reser->p_vetrl)[ reslc * ( *p_num_lin + 1 ) + *p_num_col ].vtr_desl_x +
        (*p_regis_ender_reser->p_vetrl)[ reslc * *p_num_lin + ( *p_num_col - 1 ) ].vtr_desl_x +
        (*p_regis_ender_reser->p_vetrl)[ reslc * *p_num_lin + ( *p_num_col + 1 ) ].vtr_desl_x ) +

        (1.0/12.0) * (
            (*p_regis_ender_reser->p_vetrl)[ reslc * ( *p_num_lin - 1 ) + *p_num_col - 1 ].vtr_desl_x +
            (*p_regis_ender_reser->p_vetrl)[ reslc * ( *p_num_lin - 1 ) + *p_num_col + 1 ].vtr_desl_x +
            (*p_regis_ender_reser->p_vetrl)[ reslc * ( *p_num_lin + 1 ) + *p_num_col - 1 ].vtr_desl_x +
            (*p_regis_ender_reser->p_vetrl)[ reslc * ( *p_num_lin + 1 ) + *p_num_col + 1 ].vtr_desl_x );

    return( valor );
}

/*****
NAME :
    v_val( p_regis_ender_reser, p_num_lin, p_num_col )

PARAMETROS:
    p_regis_ender_reser: aponta valores vetoriais
    p_num_lin: ponteiro - aponta valor y para calculo
    p_num_col: ponteiro - aponta valor x para calculo

FINALIDADE:
    calcula os valores de uxx + uyy segundo o metodo de atenuacao de Horn-Schunck
*****/
float v_val( p_regis_ender_reser, p_num_lin, p_num_col )
p_base_dado *p_regis_ender_reser;
int *p_num_lin, *p_num_col;
{
    int reslc;
    float valor;

    reslc = (*p_regis_ender_reser)->resol;

    valor = (1.0/6.0) * (
        (*p_regis_ender_reser->p_vetrl)[ reslc * ( *p_num_lin - 1 ) + *p_num_col ].vtr_desl_y +
        (*p_regis_ender_reser->p_vetrl)[ reslc * ( *p_num_lin + 1 ) + *p_num_col ].vtr_desl_y +
        (*p_regis_ender_reser->p_vetrl)[ reslc * *p_num_lin + ( *p_num_col - 1 ) ].vtr_desl_y +
        (*p_regis_ender_reser->p_vetrl)[ reslc * *p_num_lin + ( *p_num_col + 1 ) ].vtr_desl_y ) +

        (1.0/12.0) * (
            (*p_regis_ender_reser->p_vetrl)[ reslc * ( *p_num_lin - 1 ) + *p_num_col - 1 ].vtr_desl_y +

```

```

        (*p_regis_ender_reser)->p_vetrl)[ reslc * ( *p_num_lin - 1 ) + *p_num_col + 1 ].vtr_desl_y +
        (*p_regis_ender_reser)->p_vetrl)[ reslc * ( *p_num_lin + 1 ) + *p_num_col - 1 ].vtr_desl_y +
        (*p_regis_ender_reser)->p_vetrl)[ reslc * ( *p_num_lin + 1 ) + *p_num_col + 1 ].vtr_desl_y);

return( valor );
}

/*****
NOME:
    libr_reserv_camp_vetrl( p_base_dd )

PARAMETROS:
    p_base_dd: ponteiro - aponta base

PROCESSO:
    liberar memoria reservada para o campo vetorial
    o endereco e apontado pelo ponteiro p_base_dd
*****/
libr_reserv_camp_vetrl( p_base_dd )
p_base_dado p_base_dd;
{
    switch( p_base_dd -> resol ){
        case N3 : free( (camp_desl_8 *)p_base_dd -> p_vetrl );
            break;
        case N4 : free( (camp_desl_16 *)p_base_dd -> p_vetrl );
            break;
        case N5 : free( (camp_desl_32 *)p_base_dd -> p_vetrl );
            break;
        case N6 : free( (camp_desl_64 *)p_base_dd -> p_vetrl );
            break;
        case N7 : free( (camp_desl_128 *)p_base_dd -> p_vetrl );
            break;
        case N8 : free( (camp_desl_256 *)p_base_dd -> p_vetrl );
            break;
        case N9 : free( (camp_desl_512 *)p_base_dd -> p_vetrl );
            break;
        case N10 : free( (camp_desl_1024 *)p_base_dd -> p_vetrl );
            break;

        default : erro(6);
            break;
    }
}

/*****
NOME:
    libr_reserv_camp_param( p_base_dd )

PARAMETROS:
    p_base_dd: ponteiro - aponta base

PROCESSO:
    liberar memoria reservada para o campo parametrico
*****/
libr_reserv_camp_param( p_base_dd )
p_base_dado p_base_dd;
{
    switch( p_base_dd -> resol ){
        case N3 : free( (param_8 *)p_base_dd -> p_param );
            break;
        case N4 : free( (param_16 *)p_base_dd -> p_param );
            break;
        case N5 : free( (param_32 *)p_base_dd -> p_param );
            break;
        case N6 : free( (param_64 *)p_base_dd -> p_param );
            break;
        case N7 : free( (param_128 *)p_base_dd -> p_param );
            break;
        case N8 : free( (param_256 *)p_base_dd -> p_param );
            break;
        case N9 : free( (param_512 *)p_base_dd -> p_vetrl );
            break;
        case N10 : free( (param_1024 *)p_base_dd -> p_param );
            break;
    }
}

```



```
fflush( stdout );  
}  
  
/******  
NAME:  
    apaga_ptr( p_apaga_ptr, pp_nom_arq_reg )  
  
PARAMETROS:  
    p_apaga_ptr: ponteiro aponta ponteiro a ser apagado  
    pp_nom_arq_reg: ponteiro-ponteiro aponta ponteiro a liberar  
  
PROCESSO:  
    apagar ponteiro apontada pelo < p_apaga_ptr >  
*****/  
apaga_ptr( p_apaga_ptr, pp_nom_arq_reg )  
p_base_dado p_apaga_ptr, **pp_nom_arq_reg;  
{  
    **pp_nom_arq_reg = (p_base_dado)NULL;  
  
    free( p_apaga_ptr );  
}  
  
/*12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 */  
NOME:  
    fim()   
  
PROCESSO:  
    Imprimir mensagem de finalizacao da execucao do programa  
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 */  
fim()  
{  
    printf( "\n\tFASE 12: Programa Executado\n" );  
    printf( "\n%s\n", mensa_final );  
}
```