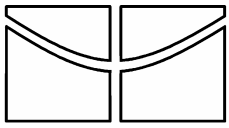
UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Implementação de Interface Visual Baseada em Estimação de Movimentos e Posição para Controle de Mouse e Teclado

Jessé Berto de Andrade
Rafael Galvão de Oliveira

Projeto Final de Graduação em Engenharia Elétrica

Brasília, Julho de 2005



UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Implementação de Interface Visual Baseada em Estimação de Movimentos e Posição para Controle de Mouse e Teclado

Jessé Berto Andrade
Rafael Galvão de Oliveira

PROJETO FINAL DE GRADUAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO.

APROVADA POR:

_____

RICARDO LOPES DE QUEIROZ , Ph.D, ENE/UnB

(Orientador)

_____

FRANCISCO DE ASSIS O. NASCIMENTO, Doutor, ENE/UnB

(Examinador)

_____

RICARDO PEZZUOL JACOBI, Docteur, CIC/UnB

(Examinador)

Brasília, 05 de Julho de 2005.

FICHA CATALOGRÁFICA

ANDRADE, JESSÉ BERTO DE;
OLIVEIRA, RAFAEL GALVÃO DE;

Implementação de Interface Visual Baseada em Estimação de Movimentos e Posição para Controle de Mouse e Teclado [Distrito Federal] 2005

Projeto Final de Graduação – Universidade de Brasília, Faculdade de Tecnologia. Departamento de Engenharia Elétrica.

REFERÊNCIA BIBLIOGRÁFICA

ANDRADE, JESSÉ BERTO DE ANDRADE; OLIVEIRA, RAFAEL GALVÃO DE OLIVEIRA (2005). Implementação de Interface Visual Baseada em Estimação de Movimentos e Posição para Controle de Mouse e Teclado. (Projeto Final de Graduação), Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF.

CESSÃO DE DIREITOS

NOME DO AUTORES: Jessé Berto de Andrade; Rafael Galvão de Oliveira
TÍTULO DA DISSERTAÇÃO: Implementação de Interface Visual Baseada em Estimação de Movimentos e Posição para Controle de Mouse e Teclado
GRAU/ANO: Engenheiro Eletricista /2005.

_____
Jessé Berto de Andrade
QS 05, Rua 121, Casa 10
CEP 71963-360 – Águas Claras – DF – Brasil

_____
Rafael Galvão de Oliveira
SQS 310 Bloco F apt. 302
CEP 70363-060 – Brasília – DF - Brasil

AGRADECIMENTOS

Jessé agradece em primeiro lugar à Deus por lhe proporcionar as vitórias em sua vida. À sua tia Selita, por ter sido fundamental no início de sua vida acadêmica. À sua família Maria das Graças, Geniane Raquel, Genaíne e Neusair, por lhe proporcionar meios para se dedicar à faculdade. À Luana Carolina pela companhia e por tantas alegrias que são fundamentais para o bom desempenho de sua vida profissional e acadêmica.

Agradece ao Professor Ricardo de Queiroz por ter orientado o desenvolvimento deste projeto. Aos amigos João Gabriel, Gabriel Sartori e Marco Rafael Evangelista, por terem iniciado juntos o Grupo de Interface Visual. Ao amigo Rafael Galvão por ter trabalhado junto no desenvolvimento deste projeto.

Agradece aos colegas de curso: Gabriel Sartori, João Gabriel, Otávio Franco, Marco Rafael Evangelista, Maurício Iryoda, Aguiar da Costa, Gabriel Costa, Pascoal Guido, Rafael Neves, Rodrigo Coelho, Eduardo Peixoto, Flávio Alencar, Maísa Leidemer, Alberto Friedman, Gabriel Castro, Andrey Leonardo, Antônio Alex, Aniela, Carlos, Demian, Henrique Louzada, Jhonathan, Nagao, Léa, Milene, Lucas Lira, Luciana, Saulo, Popov, Joaz, Gustavo, Jill, Virginia, Darfe Diogo, Tiago Alves, Edson entre outros não mencionados; pela receptividade à universidade, pelas brincadeiras que descontraiam e permitiam eliminar um pouco do estresse, por participarem no dia a dia da faculdade e sendo sempre úteis nos momentos de aperto e desespero antes das provas e por serem sempre amigos.

Agradece aos amigos do CEFET-MG, onde cursou os três primeiros semestres do curso: Renato Ramalho, Ramom, Vinícius, Marcos Antônio Camargos, Tiago, Lucas, Maria Cristina, Simone, Erick Daniel, Luiz Fernando, Leandro Coelho, Tiago Abjaldi, Leonardo e a outros que o tempo atrapalha de lembrar os nomes; por terem sido fundamentais no início de uma nova etapa de sua vida.

Agradece também aos colegas de trabalho Paulo Renato Medrado Roque, Ednilson Bruno Silva do Nascimento e Juliana Batista Costa por lhe ajudarem na fase final do desenvolvimento deste projeto.

Rafael agradece aos seus pais, pois sem eles nada disso seria possível, irmãos Gabriel e Larissa, pela amizade.

Agradece à Érica companheira em todos esses anos.

Agradece ao seu orientador, Professor Ricardo de Queiroz, por ter acreditado em meu potencial e por toda a orientação. Aos amigos Gabriel Sartori e Marco Rafael Evangelista, pelo apoio e incentivo nesse projeto. Ao amigo João Gabriel por ter iniciado o projeto. E a Jessé, que esta comigo nesse momento.

Agradece ao Rafael Ortiz sem o qual o trabalho no GPDS seria muito mais difícil, senão impossível.

Aos colegas de curso, pelos momentos de lazer e descontração.

Aos amigos Edson e Tiago com quem convivi muito nesses últimos meses. Especialmente ao Edson que me ajudo a revisar este trabalho.

A todos do GPDS, que tornaram deste laboratório não somente um local de trabalho, mas um excelente local de convívio.

Aos professores, que contribuíram com minha formação acadêmica.

DEDICATÓRIA

Jessé dedica este trabalho à sua família e à Luana Carolina

Rafael dedica este trabalho à sua família, especialmente a sua mãe, e a Érica.

RESUMO

O presente trabalho visa apresentar uma proposta de interface visual genérica baseada em estimação de movimentos e estimação de posição capaz de controlar o mouse e o teclado. A interface proposta possui uma arquitetura versátil podendo ser adaptada para atender diversas necessidades. Além disso, acessível, pois foi desenvolvida para o Sistema Operacional Windows, a plataforma mais popular entre os usuários de microcomputadores.

ABSTRACT

The present work presents a general purpose visual interface based on motion and position estimation. The interface is able to control the mouse and the keyboard and has a versatile architecture. Besides, it is very accessible since it was developed for the popular Microsoft Windows operational system.

Índice

Índice de Figuras

Índice de Tabelas

LISTA DE ABREVIAÇÕES

Abreviação	Significado
fps	Frames per second
VFW	Video for Windows
MFC	Microsoft Foundation Class
DFD	Distorção do Frame Deslocado
DBD	Distorção do Bloco Deslocado
DRD	Distorção da Região Deslocada
SAD	Sum of Absolute Difference
OWT	Overlap-Windowed Transform

CAPÍTULO 1

INTRODUÇÃO

Nos últimos anos, tem havido uma popularização muito grande do acesso aos computadores e sistemas informatizados. Elas permeiam nossa vida cotidiana nos mais diversos aspectos. Exemplos disso são a digitação de trabalhos como esse, o uso de um caixa eletrônico ou terminais de acesso em diversos locais públicos, acesso a Internet para diversos fins ou mesmo o uso para o lazer com jogos mais variados.

Também essa tecnologia tem chegado a mais pessoas com o barateamento dos equipamentos nos últimos anos. Essa inclusão digital passou a ser, inclusive, uma política de governo.

Junto com esse barateamento e popularização, houve um vertiginoso aumento da capacidade computacional. Apesar de tudo isso, a maneira como interagimos com os computadores não mudou praticamente nada. Temos até hoje como os principais dispositivos de entrada, o mouse e o teclado. Outras formas de interação mais naturais, como o reconhecimento de voz, já foram propostas, mas ainda tem pouco uso.

Este trabalho trata de uma outra forma de interação com o computador, uma interface visual. A idéia é que o usuário possa ficar na frente de uma câmera e sem tocar em nada possa fazer uso do computador.

Embora dificilmente ela venha no curto prazo substituir formas tradicionais de interface, ela pode ser usada em situações que é inconveniente, impossível ou simplesmente menos divertido o uso de um teclado ou um mouse. Por exemplo, um cirurgião poderia acessar o prontuário do paciente sendo operado sem a necessidade de preocupação com assepsia ou um

operário que tem ambas as mãos ocupadas poderia utilizar seu computador sem preocupações.

Seria possível também possibilitar uma maior autonomia a pessoas debilitadas com dificuldade de movimentos. Um paciente paraplégico poderia controlar a cadeira de rodas, o mouse de seu computador ou os eletrodomésticos.

Ainda podemos utilizar essa interface para jogar.

Esse trabalho implementa uma interface visual simples. Ela tem como objetivo controlar o mouse e o teclado. Para tornar acessível a maioria dos usuários de computador, o software foi desenvolvido em ambiente Windows e utiliza-se uma webcam simples.

A captura, como será explicado em mais detalhes em outro capitulo, é feita pela biblioteca chamada Vídeo for Windows que esta disponível em qualquer distribuição do Windows.

A partir das imagens capturadas, são estimados os movimentos ou posições. Com isso podemos interpretar os dados com reconhecimento de padrões e usar isso para a interface.

O presente texto está dividido em sete capítulos, sendo o primeiro esta introdução. O segundo capítulo da uma visão geral de como deve ser a arquitetura proposta para a interface visual. O terceiro capítulo é sobre a captura de imagens feita no projeto. Descreve duas tentativas, sendo uma bem sucedida. Além disso, mostra todas as funções referentes a captura de imagens. O quarto capítulo mostra como foi feita a detecção de movimento, que tipo de mascara foi utilizada e os critérios empregados. O quinto e o sexto capítulos são referentes a estimação de movimentos. O quinto é um resumo teórico e o sexto apresenta alguns padrões de busca utilizados no projeto. O sétimo capítulo traz as conclusões dos autores. As implementações e informações adicionais da interface estão nos apêndices.

CAPÍTULO 2

ARQUITETURA DA INTERFACE

A arquitetura proposta é mostrada esquematicamente na figura 2.1. Deve-se perceber que as formas tradicionais de interface com o computador são mantidas. Assim, se o usuário quiser, ainda pode utilizar o teclado ou o mouse.

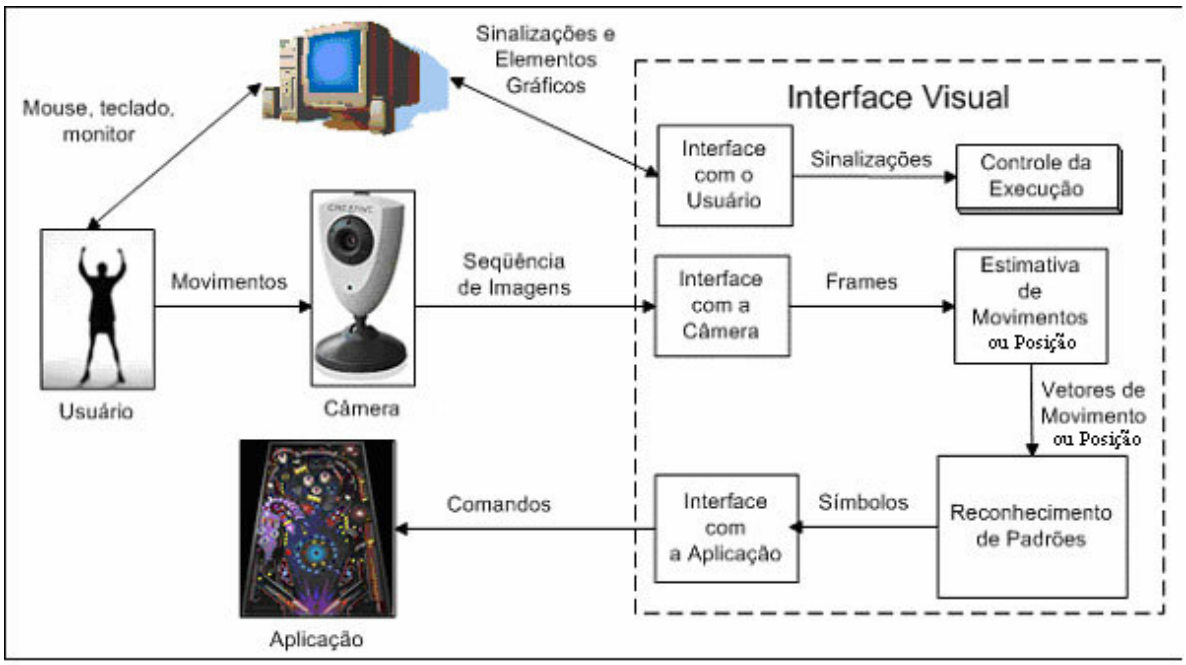


Figura 2.1 Arquitetura da Interface Visual

A interface com a câmera permite que as imagens possam ser acessadas *pixel* a *pixel*. No caso da estimativa de movimento, usa-se o frame anterior e o atual para se criar um campo de vetorial com essas estimativas.

Atualmente, esses vetores de movimento são processados por regiões como será visto a frente. As regiões são definidas em um arquivo de configuração, que ainda contém o significado de cada movimento em cada região. Por exemplo, pode-se definir duas regiões, uma a esquerda e outra a

direita. Um movimento predominantemente para cima na região a direita pode significar a tecla A e na da direita pode ser a tecla L.

Ou pode-se utilizar a media dos movimentos da região para movimentar o mouse de acordo.

No caso da detecção de posição, utiliza-se a posição de um objeto para controlar o mouse.

Contudo, como já se tem o campo vetorial e a posição do objeto, pode-se interpretá-los de forma mais sofisticada. Já existe um trabalho que calcula o centro e a média dos movimentos. Esses valores servem como entrada em um modelo baseado em Hiden-Markov para reconhecer movimentos mais complexos. No trabalho citado, esses movimentos eram usados para controlar um carro de controle remoto. O mesmo poderia ser utilizado para controlar outros dispositivos. Ou seja, a arquitetura proposta é bem flexível e pode se adaptar as mais diversas necessidades.

CAPÍTULO 3

CAPTURA DAS IMAGENS

3.1 DIRECTSHOW

Foram adotadas duas abordagens bem diferentes para a interface com a câmera. Primeiramente utilizou-se o DirectShow que é parte integrante do DirectX da Microsoft.

O desenvolvimento no DirectShow é feito cascateando filtros de forma conveniente. [8] O programador não necessita saber nada sobre o funcionamento interno do filtro para utilizá-lo. Basta saber qual o tipo de entrada e qual o tipo de saída para fazer as conexões correta com os outros filtros.

Há três tipos de filtro. O primeiro tipo é de captura. Ele faz a conexão com o dispositivo e disponibiliza o fluxo de dados, possui apenas saída. Há também o filtro de transformação, que processa os dados recebidos e passam adiante. Finalmente, o ultimo tipo de filtro é o de renderização. Este tipo de filtro fornece uma saída para os dados, que pode ser a escrita em um arquivo, a visualização na tela ou a tocar nas caixas de som, dependendo do tipo de dado e do filtro escolhido.

Outra facilidade do desenvolvimento em DirectShow é que ele fornece uma interface gráfica para o desenvolvimento de grafos de filtros, o GraphEdit. Na figura 3.1, é mostrado um grafo possível para o problema proposto.

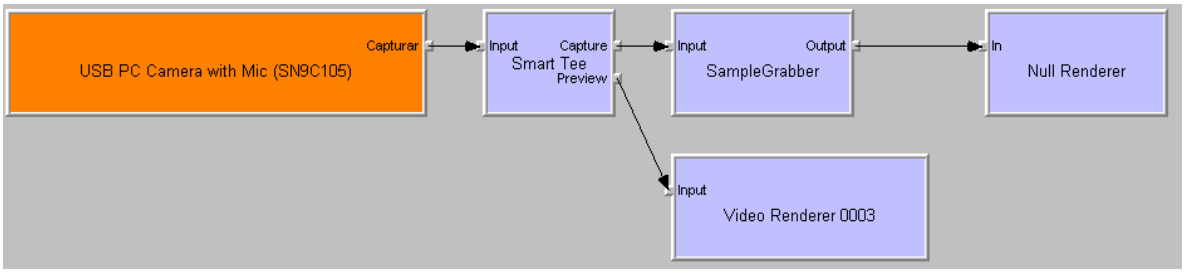


Figura 3.1 Grafo de filtros proposto para a captura.

O filtro Smart Tree separa o fluxo que recebe em dois. Um deles é utilizado para a captura e vai para o filtro SampleGrabber, que permite acesso a cada pixel da imagem capturada. A saída desse filtro vai para o Null Renderer, que não faz nada. Caso se desejasse mostrar o resultado de algum processamento na tela, como a compensação de movimentos que será visto a frente, poderia se utilizar um Vídeo Renderer como o usado depois da saída Preview. Isso foi feito para que a imagem aparecesse na tela.

Infelizmente, embora tanto a Borland quanto a Microsoft digam que há compatibilidade entre o DirectShow e o C++ Builder, não foi encontrada nenhuma documentação acerca disso. Por isso, quando se migrou do Microsoft Visual C++.Net, resolveu-se criar uma nova classe de captura utilizando Vídeo for Windows.

Apesar disso, tudo o que foi desenvolvido foi guardado. Caso queira-se migrar para o novamente para o Visual C++ ou descubra-se como utilizar no Builder e isso seja conveniente, pode-se aproveitar o que já foi feito.

3.2 VIDEO FOR WINDOWS

A utilização do Vídeo for Windows (VFW) é bem mais simples que o DirectShow e possui bem menos recursos. Contudo, ele serviu muito bem as necessidades do projeto.

O primeiro passo ao se utilizar o VFW para se fazer uma captura é criar uma janela de captura. Isso pode ser feito com a função *capCreateCaptureWindow* do próprio VFW. Essa função retorna um valor do tipo HWND, *Window Handling*. A partir daí podemos usar esse valor para controlar os atributos da captura, conectar ao driver disponível, iniciar a captura, finalizar, entre outros.

Há duas opções para utilizar a janela criada. Uma é enviar mensagens para janela utilizando a função *SendMessage* e a constante apropriada. Outra opção mais interessante é utilizar os macros fornecidos nessa biblioteca. Abaixo vemos duas linhas de código que fazem a mesma coisa:

SendMessage(JanelaDeCaptura, WM_CAP_DRIVER_CONNECT, 0, 0);

capDriverConnect(JanelaDeCaptura, 0);

Ambas as linhas tentam conectar com o primeiro dos *drivers* instalados (o Windows guarda uma lista de até 10 *drivers* de captura de vídeo). A primeira opção foi escolhida por ser mais didática.

O VFW usa funções de *Callback* que são chamadas automaticamente na ocorrência de um determinado evento. O programador passa um ponteiro para a função que ele deseja que seja chamada. Os dois principais tipos de eventos utilizados foram o *StreamCalback* e o *FrameCalback*. O primeiro é chamado toda vez que o buffer para o vídeo esta cheio. A função a ser chamada pode ser escolhida pela macro *capSetCallbackOnVideoStream*.

Contudo, a função somente será chamada caso uma captura, de fato, tenha sido iniciada. Ou seja, deve-se indicar um arquivo para receber os dados ou utilizar a macro *capCaptureSequenceNoFile*. Não é preciso dizer que o uso de um arquivo estava fora de questão. Infelizmente, foram encontrados problemas ao se utilizar esta ultima macro. Por isso, resolveu-se utilizar o *FrameCalback*. Este é chamado toda vez que um frame vai ser mostrado na tela e pode ser definido pela macro *capSetCallbackOnFrame*. O problema é que se uma janela estiver na frente da janela de *preview*, não haverá uma chamada. A solução encontrada para esse problema foi pedir a captura de um frame de tempos em tempos ao invés de escolher uma taxa e deixar que o VFW se preocupasse com tudo. Para tanto, utilizou-se um componente Timer do *Builder* para utilizar a macro *capCaptureSingleFrame*, que captura um único frame. Como a taxa foi definida com sendo 30fps, o intervalo entre as chamadas deve ser de 33ms.

É interessante notar que, toda vez que se usa a macro *capCaptureSingleFrame*, a função que é definida para *FrameCalback* é chamada. Mesmo quando não é mostrada na tela. Sendo assim, essa solução foi adotada para a captura.

A implementação da captura encontra-se no Apêndice A

CAPÍTULO 4

DETECÇÃO DE POSIÇÃO

Uma outra forma, de controle do mouse foi usar detecção de posição. Nesta técnica um objeto é utilizado para que sua posição na imagem capturada pela webcam sirva para determinar a posição do mouse na tela.

O objeto a ser encontrado tem suas dimensões inferiores às dimensões da imagem capturada. Este objeto estará, por definição, em uma sub-imagem da imagem original.

Para que seja encontrada a posição desta sub-imagem é utilizada uma mascara, de dimensões variáveis, que a contém. O algoritmo de detecção de posição consiste em comparar a sub-imagem contida na máscara com cada ponto da imagem original e apontar onde elas mais se parecem.

Aqui serão apresentadas duas formas de comparação: Soma das Diferenças Absolutas (SAD, do inglês *Sum of Absolute Difference*) e Correlação.

4.1 SOMA DAS DIFERENÇAS ABSOLUTAS

Como o cálculo da SAD será apresentado no item 5.2.1 vamos nos ater aqui as suas vantagens e desvantagens nesta aplicação.

O cálculo usando SAD é atraente por sua simplicidade, pois utiliza a operação de subtração que é bem mais rápida do que a multiplicação – utilizada na correlação como será mostrado mais adiante.

Porém a desvantagem encontrada no uso da SAD nesta aplicação deve-se aos seguintes fatos: o objeto a ser procurado é definido em um instante muito anterior ao que a imagem é capturada. Esse fato pode trazer grandes diferenças entre a iluminação da imagem e a iluminação da sub-imagem contida na máscara; o objeto está sob controle do usuário, portanto acontecem movimentos de aproximação, afastamento e rotação, causando também diferença com o antigo objeto contido na máscara.

Estes fatos fazem com que, na prática, sempre haja diferença entre o objeto contido na máscara e o objeto contido na imagem. Ou seja, nunca haverá a diferença exatamente igual a zero e em alguns casos essa diferença pode ser relativamente grande. Se for calculada a diferença entre regiões totalmente diferentes, porém que tenham baixos níveis de cores, esse cálculo retornará um valor baixo. Este poderá ser menor que o valor retornado, quando foi testado o ponto em que o objeto realmente se encontra, devido aos problemas enumerados acima.

Uma forma de superar esta dificuldade seria utilizar sempre uns objetos com níveis de cores mais altos, porém esta já seria uma restrição ao sistema e a comparação por correlação se mostrou mais robusta.

4.2 CORRELAÇÃO [7]

A função correlação é dada pela expressão:

$$c(x, y) = \sum_{s} \sum_{t} f(s,t)w(x+s, y+t) \qquad (4.2.1)$$

para os valores de $f \in \Re$ e $x = 0,1,2,...,M-1$; $y = 0,1,2,...,N-1$; $s = 0,1,2,...,K-1$ e $t = 0,1,2,...,J-1$ sendo M e N as dimensões horizontal e vertical da imagem, K e J as dimensões horizontal e vertical da sub-imagem respectivamente.

Para que a função correlação funcione a contento, foi necessário alterar a faixa de níveis de cinza de 0 a 255 para –127 a +128. Desta forma, quando se calcula a correlação em um ponto qualquer da imagem, em que não

se encontra a sub-imagem procurada, a multiplicação retorna valores negativos e positivos que tendem a se anular. Já quando a função é calculada no ponto da imagem que contém a sub-imagem procurada, tem-se $f(a,b) = w(a,b)$ e

$$c(x,y) = \sum_s \sum_t f(s,t) f(x+s, y+t) \qquad (4.2.2)$$

a multiplicação retorna o quadrado do valor de cada pixel e a correlação é máxima. A técnica consiste então em calcular a correlação para todos os pontos da imagem, e no valor máximo é encontrado o ponto em que se encontra o objeto procurado, como ilustrado abaixo.

Figura 4.1 Imagem obtida através da wecam

Figura 4.2 Máscara usada para buscar o objeto

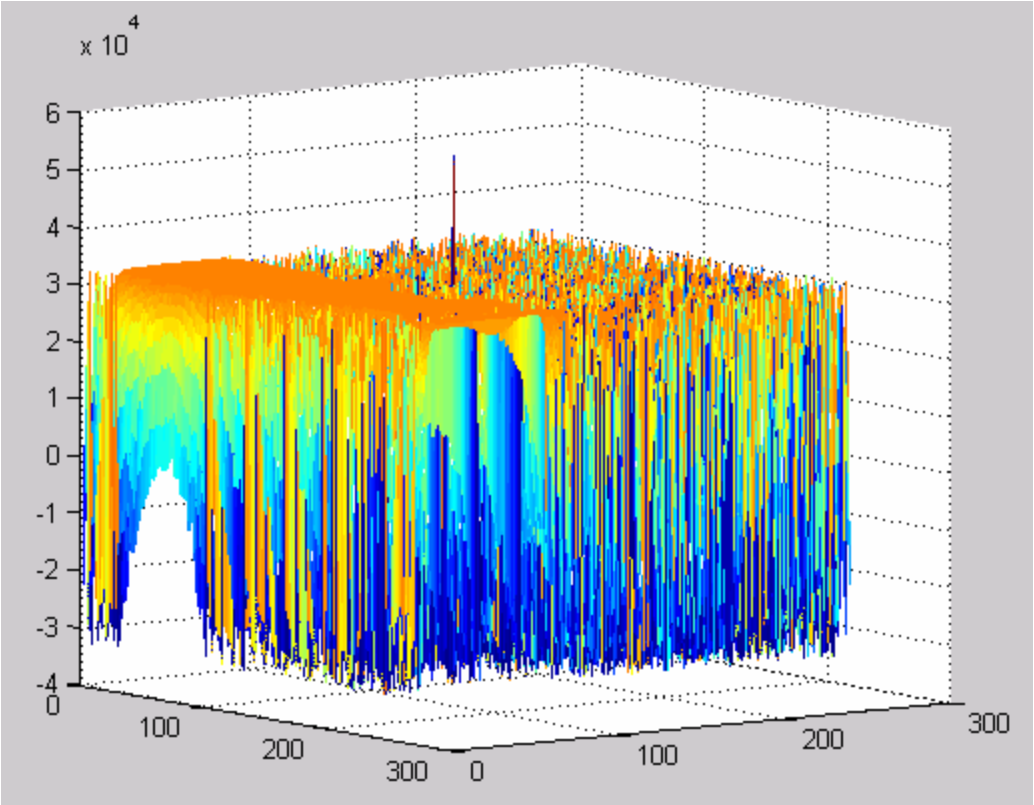


Figura 4.3 Correlação encontrada para cada ponto da imagem.

A implementação da detecção de posição está descrita no Apêndice B.

CAPÍTULO 5

ESTIMATIVA DE MOVIMENTO EM UMA SEQÜÊNCIA DE IMAGENS

Para estimarmos os movimentos em uma seqüência de imagens, primeiro temos que entender como essas imagens são formadas. Para efeitos deste trabalho, consideraremos que vivemos em um mundo tridimensional e tentaremos representá-lo em imagens bidimensionais. Devido a isso, qualquer movimento tridimensional será projetado em duas dimensões, o plano da imagem. Essa projeção é chamada de movimento aparente ou fluxo ótico. São necessárias, ainda, a escolha de três elementos. O primeiro é o modelo. Esse modelo relacionará os movimentos bidimensionais com as trajetórias tridimensionais. A escolha do modelo dependerá dos objetivos e, quanto mais preciso, maior será a complexidade do modelo. Em adição temos o segundo elemento, que se refere ao critério de estimação, que será discutido mais adiante. Finalmente, deve-se escolher o terceiro elemento, que é a estratégia de busca.

5.1 MODELOS

5.1.1 Representação dos Movimentos.

Como já foi visto, as trajetórias tridimensionais dos objetos em frente a uma câmera são projetadas em um plano bidimensional, como toda a imagem.

O modelo deve representar os movimentos tridimensionais a partir dos movimentos bidimensionais percebidos no plano da câmera.

Na figura 5.1, vemos um exemplo dessa projeção da trajetória.

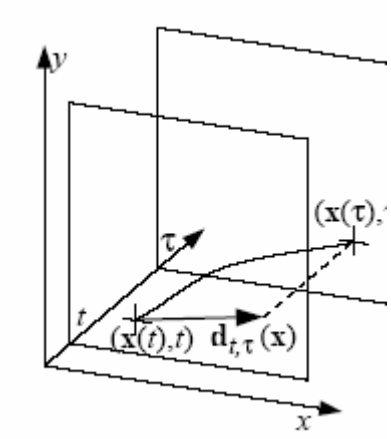


Figura 5.1 Projeção de trajetória tridimensional no plano da imagem.

Vários movimentos tridimensionais podem resultar em um único movimento bidimensional, ou seja, não há unicidade entre os movimentos bidimensionais e os movimentos tridimensionais.

A tabela abaixo mostra alguns modelos que podem ser usados. E na figura 5.2 podemos ver a compensação de movimentos para um quadrado localizado no centro da imagem e os vetores de movimento.

Tabela 5.1Modelos de Movimento.

2-D model			*3-D model*		
	Number of parameters	*Motion field*	*3-D surface function*	*3-D motion*	*Camera model*
Translational	2	$\boldsymbol{d}(\boldsymbol{x}) = (a_1, b_1)^T$	arbitrary	rigid 3-D translation	orthographic
Affine	6	$\boldsymbol{d}(\boldsymbol{x}) = \begin{pmatrix} a_1 & a_2 \\ b_1 & b_2 \end{pmatrix} \boldsymbol{x} + \begin{pmatrix} a_3 \\ b_3 \end{pmatrix}$	planar	3-D affine	orthographic
Projective linear	8	$\boldsymbol{d}(\boldsymbol{x}) = \begin{pmatrix} \frac{a_1+a_2x+a_3y}{1+a_4x+b_4y} \\ \frac{b_1+b_2x+b_3y}{1+a_4x+b_4y} \end{pmatrix} - \boldsymbol{x}$	planar	3-D affine	perspective
Quadratic	12	$\boldsymbol{d}(\boldsymbol{x}) = \begin{pmatrix} a_1+a_2x+a_3y+a_6x^2+a_5xy+a_4y^2 \\ b_1+b_2x+b_3y+b_6x^2+b_5xy+b_4y^2 \end{pmatrix}$	parabolic	3-D affine	orthographic
Sampled	2 per Δ^2 pixels	$\boldsymbol{d}(\boldsymbol{x}) = \sum_{i,j} \begin{pmatrix} a_{ij} \\ b_{ij} \end{pmatrix} H(x - \Delta \cdot i, y - \Delta \cdot j)$	'smooth' as specified by interpolation kernel H		arbitrary
Polynomial	$2\lvert\mathcal{K}\rvert$ motion-adaptive	$\boldsymbol{d}(\boldsymbol{x}) = \sum_{(i,j)\in\mathcal{K}} \begin{pmatrix} a_{ij} \\ b_{ij} \end{pmatrix} x^i y^j$	'smooth' as specified by $\mathcal{K}$		arbitrary

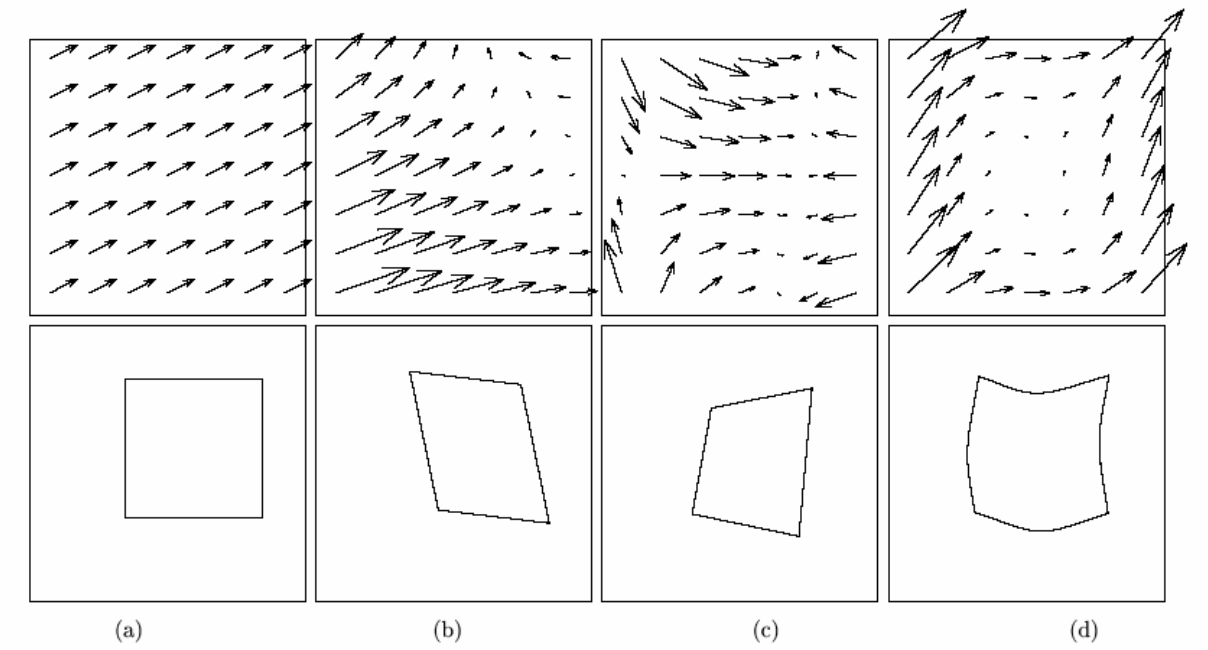


Figura 5.2 Exemplos de campos de vetores de movimento e a compensação seguindo alguns modelos.
(a) translacional, (b) affine, (c)projeção linear e (d) quadrática.

O modelo mais simples é o translacional que representa movimento rígido sobre uma projeção ortográfica resultando em um movimento 2-D constante espacialmente. Assim, a compensação de movimento com um campo desse tipo preserva a forma 2-D do objeto.

O modelo translacional será o modelo utilizado nesse trabalho. Ele também é o mais utilizado nos padrões de compressão de vídeo.

5.1.2 Região para representação de movimentos.

A região de suporte são os pontos da imagem em que o modelo escolhido é aplicado. Quanto menor for a região, maior será a precisão e a complexidade do campo vetorial.

Na figura 5.3, podemos ver alguns tipos de região de suporte.

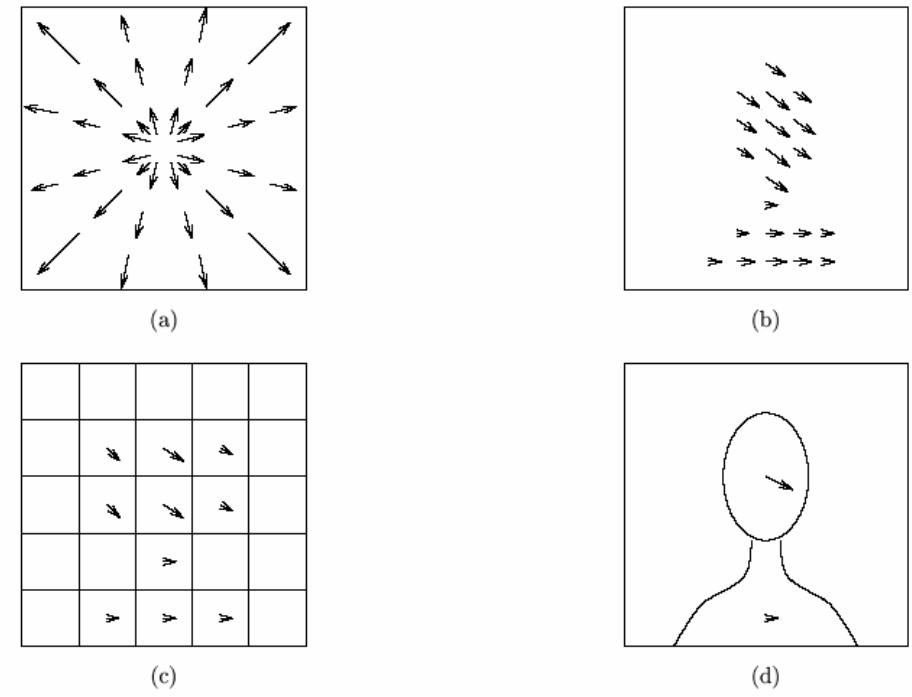


Figura 5.3 Exemplos de regiões de suporte para um modelo de movimento.
(a) global, (b) densa, (c) baseada em bloco e (d) baseada em regiões arbitrárias.

5.1.2.1 Movimento Global

É o caso mais simples. A região de suporte é a imagem inteira, sendo assim, toda ela tem o mesmo movimento. Esse tipo de região é muito limitado, mas funciona muito bem para movimentos introduzidos pela câmera como no caso do zoom.

5.1.2.2 Movimento de todos os pontos da imagem.

Esse caso é o extremo oposto do caso acima. A região de suporte é um único ponto da imagem, um *pixel*. Neste caso, a precisão é muito grande, pois associa a um enorme número de vetores no campo vetorial. Implicando ainda em um maior esforço computacional, uma vez que aumentamos o espaço de soluções.

5.1.2.3 Movimento baseado em regiões.

Podemos ainda considerar um caso híbrido baseado na representação de movimentos descritos em 5.1.2.1 e 5.1.2.2. Assim podemos conseguir uma precisão bem maior que o caso do movimento global tendo uma complexidade menor que a segundo caso. As regiões mais simples e mais utilizadas são retângulos não superpostos, cuja união cobre a imagem toda. O particionamento nessas regiões em blocos é vastamente utilizado na compressão de vídeo.

No caso de regiões com formato arbitrário, há uma possibilidade de casamento muito melhor, mas há um aumento muito grande na complexidade da descrição da fronteira das regiões.

Na figura 5.4, estão exemplos de campos de vetores para as varias regiões de suporte.

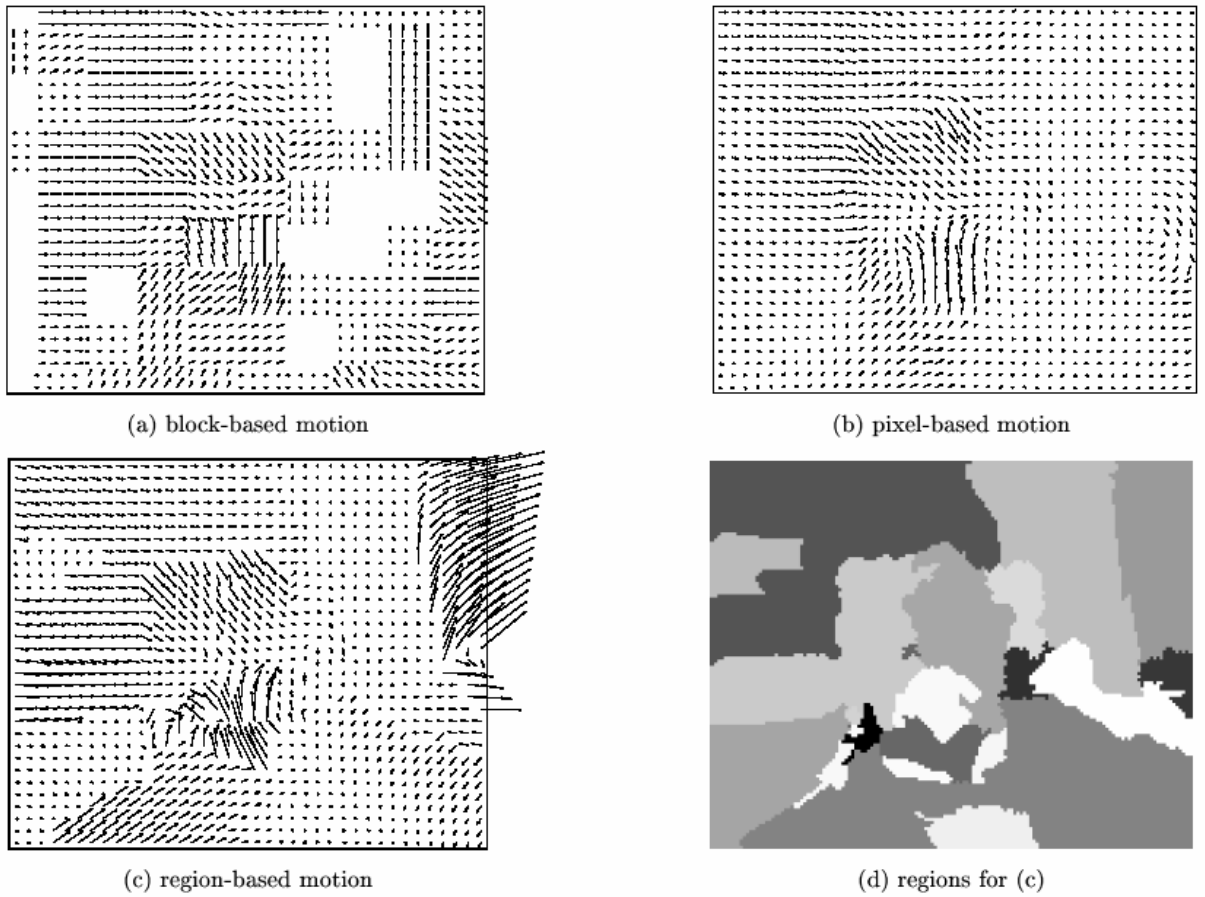


Figura 5.4 Campos de vetores de movimento para três regiões de suporte. (a) baseada em blocos de 16 x 16; (b) baseada em pixels; e (c,d) baseada em regiões.

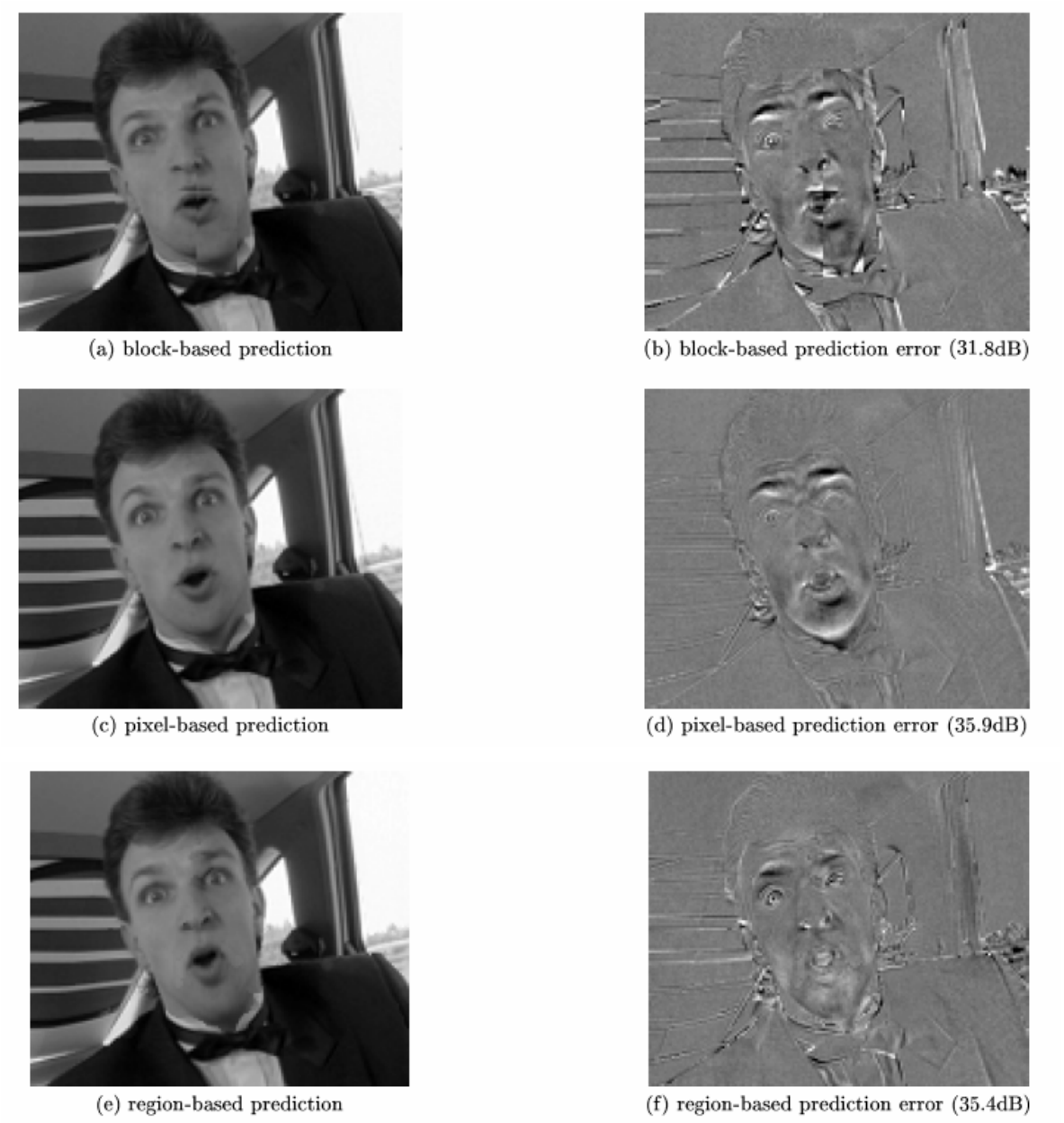


Figura 5.5 Compensação de movimentos para os campos de vetores da figura 5.4.

5.2 CRITÉRIO DE ESTIMAÇÃO.

Há diversos critérios que podem ser usados na estimação de movimento. Já o segundo, mede-se o erro baseados na diferença de duas imagens (entre a imagem anterior e a atual).

5.2.1 Critério baseado no DFD

Talvez o critério mais utilizado se baseie na minimização do erro mostrado na equação (5.2.1-1).

$$\varepsilon_{t,\tau}(\boldsymbol{x}) = g_t(\boldsymbol{x}) - \hat{g}_{t,\tau}(\boldsymbol{x}), \qquad \forall \boldsymbol{x} \in \mathcal{R} \tag{5.2.1-1}$$

onde $\hat{g}_{t,\tau}(\boldsymbol{x})$ = g_τ (**x**+$\mathbf{d}_{t,\tau}$(**x**)) é chamado a estimativa, pela compensação de movimento, de g_t(**x**).

Caso $\mathcal{R}$ seja a imagem inteira, o erro é a DFD (do inglês *displaced frame difference*) , diferença do frame deslocado. Sendo uma região, g_t(x) pode ser chamado de diferença do bloco deslocado (DBD) ou diferença da região deslocada (DRB) conforme o caso.

Para medir o erro, temos diversos critérios que podem ser utilizados. Abaixo são mostrados alguns.

$$J_1(\boldsymbol{d}) = \sum_{\boldsymbol{x} \in \mathcal{R}} \left(g_t(\boldsymbol{x}) - g_\tau(\boldsymbol{x} + \boldsymbol{d}(\boldsymbol{x}))\right)^2 . \tag{5.2.1-2}$$

é o modulo do erro ao quadrado,

$$J_2(\boldsymbol{d}) = \sum_{\boldsymbol{x} \in \mathcal{R}} \left|g_t(\boldsymbol{x}) - g_\tau(\boldsymbol{x} + \boldsymbol{d}(\boldsymbol{x}))\right| \tag{5.2.1-3}$$

Segundo a referencia [1], esse ultimo é um critério mais robusto, pois um erro grande em um único *pixel* pode contribuir excessivamente para a soma toda. Este último critério é conhecido como SAD (*Sum of absolute difference*).

Geralmente, quando se esta trabalhando em ponto fixo, utiliza-se o erro médio, para evitar *overflow*. Contudo, isso não é uma preocupação nesse trabalho já que utilizamos um inteiro de 32 *bits* e a soma dos erros não excede a faixa de valores suportados.

Há diversos outros critérios baseados na mediana dos erros, utilizando logaritmo, fazendo operações de “ou-exclusivo” dentre outros. Contudo, eles

não serão abordados aqui. Pode-se consultar as referências [1] e [2] para maiores informações.

5.3 ESTRATÉGIA DE BUSCA.

Como estratégia de busca utiliza-se o casamento (m*atching*). São testados vários candidatos e escolhe-se o melhor, com menor erro. Assim estaremos minimizando o erro da imagem compensada como um todo. Como são escolhidos esses candidatos depende do padrão de busca escolhido. Isso será tratado no próximo capitulo.

CAPÍTULO 6

PADRÃO DE BUSCA.

Como foi visto no capitulo anterior, devem ser escolhidos alguns candidatos para o casamento e escolher o melhor (que minimize o erro). Há diversas formas de fazer isso. A mais obvia é fazer a comparação de cada bloco com todos os blocos do frame anterior. Essa forma garantirá o menor erro possível. Contudo, isso demanda um imenso esforço computacional. Deve-se lembrar que esse trabalho propõe criar uma interface visual. Por isso, todo o processamento será feito em tempo real e não deve ocupar boa parte dos recursos, que serão reservados para a aplicação.

6.1 PADRÕES DE BUSCA

6.1.1 Busca circular [3]

Um exemplo de padrão de busca que reduz o numero de pontos de buscas é pela varredura em zonas circulares, conforme mostrado na figura 6.1:

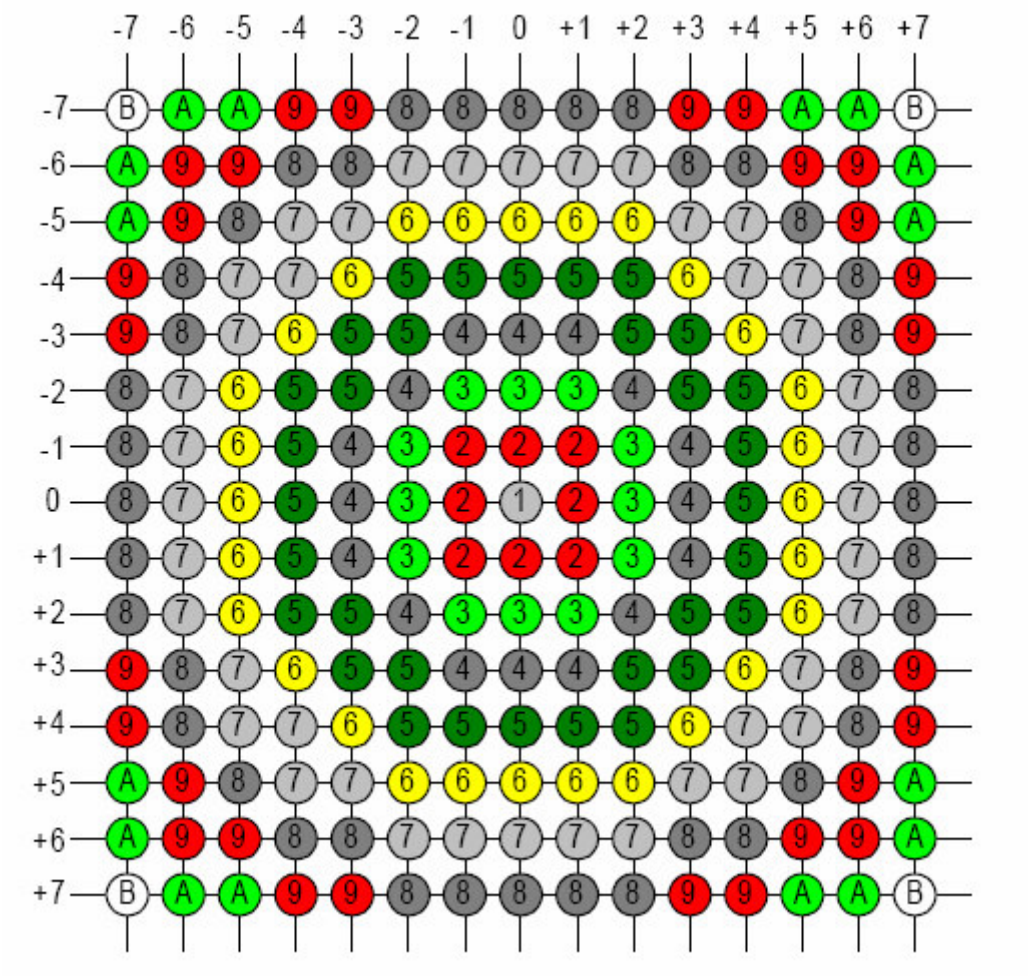


Figura 6.1. Zonas circulares.

Ele compara o casamento do melhor candidato de uma zona interior com a imediatamente exterior. O algoritmo pára quando o casamento do raio interno for melhor que o externo. Caso contrário, o algoritmo continua no raio seguinte. Por exemplo, caso o melhor candidato da zona 6 for melhor que o da zona 7, o algoritmo pára. Caso não seja, ele comparará o melhor da zona 7 com o melhor da zona 8.

Esse padrão de busca já é muito melhor que buscar em toda a imagem. Mas a busca ainda é muito exaustiva. Além do mais, à medida que os candidatos se afastam do centro, é exigido um maior esforço computacional. Sendo assim, algumas implementações limitam o raio máximo para a busca. Entretanto, o número mínimo de pontos de busca é 9.

Esse padrão de busca é apresentado na referencia [3].

6.1.2 Padrão de busca Hexagonal [4] e [5]

Outro padrão de busca interessante é o padrão Hexagonal, mostrado na figura 6.2.

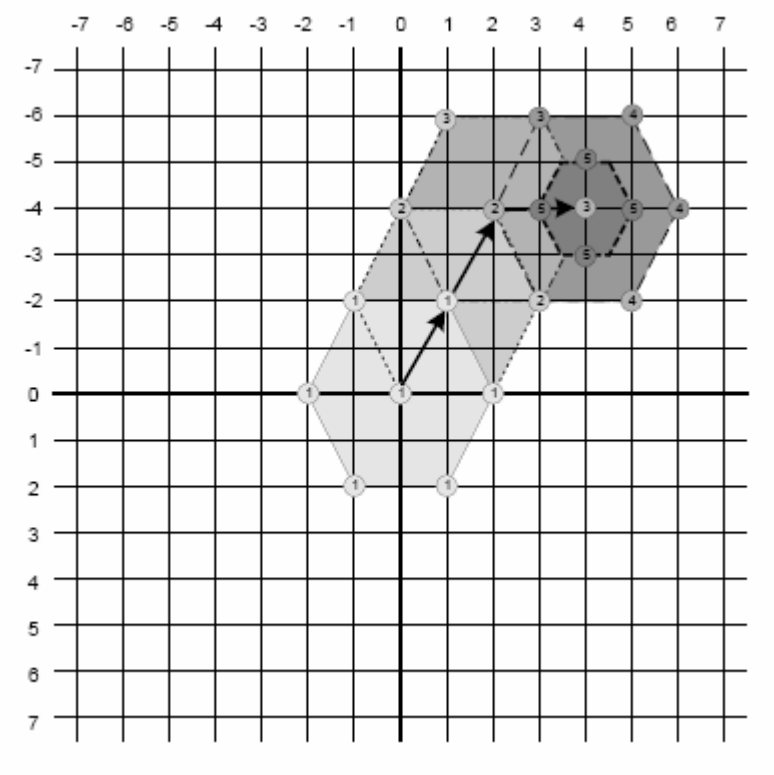


Figura 6.2 Padrão de busca Hexagonal.

Verifique que o movimento é mantido enquanto o candidato central não for o melhor. É interessante notar que, em cada novo hexágono, somente há a necessidade de se fazer três novas comparações, pois as outras já são conhecidas dos passos anteriores. Há diversas variações desse padrão de busca. Principalmente dizendo respeito ao ajuste fino que é feito no último hexágono.

O número mínimo de pontos de busca da variação mostrada na figura 6.2 é 11.

Uma variação interessante desse padrão hexagonal é o Hexagonal Melhorado (*Enhanced Hexagonal*), na figura 6.3.

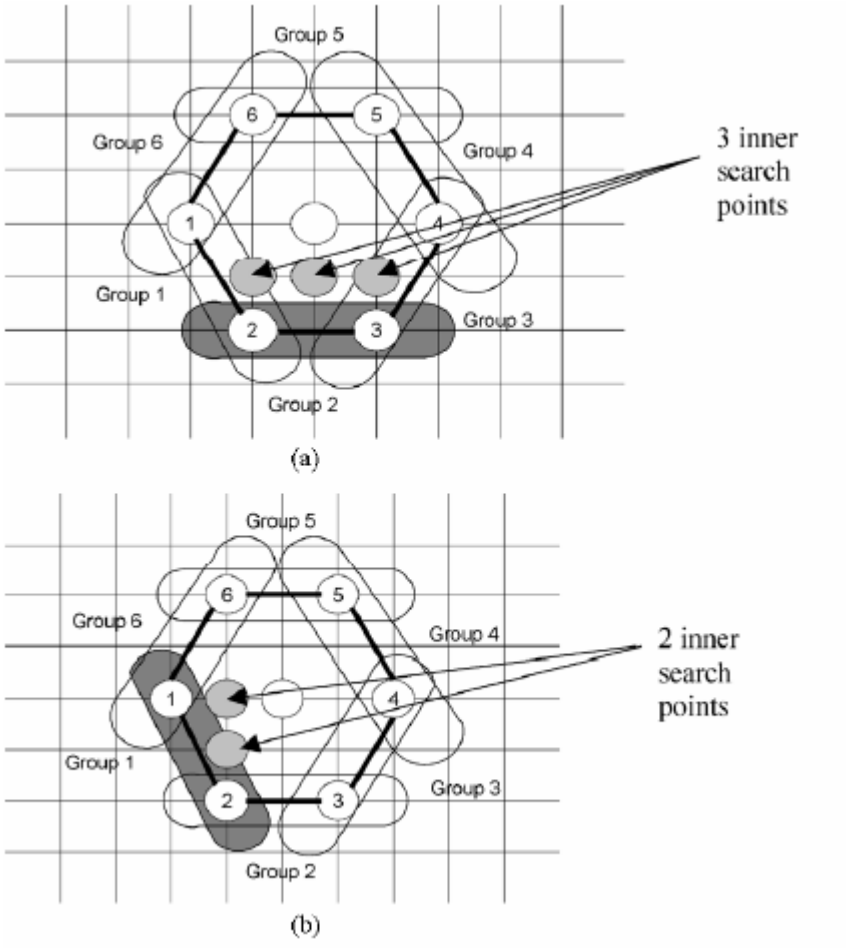


Figura 6.3 Padrão Hexagonal Melhorado.

A diferença deste para o anterior é que, no ajuste fino, somente são testados os pontos próximos ao melhor grupo (que tenha a menor soma dos erros). Por conseguinte, diminui-se o número de pontos de busca.

6.1.3 Padrão de busca em forma de losango [6]

O último padrão a ser apresentado aqui é o padrão em formato de losango (*Diamond*). Ele é o mais simples de todos e pode ser visto na figura 6.4.

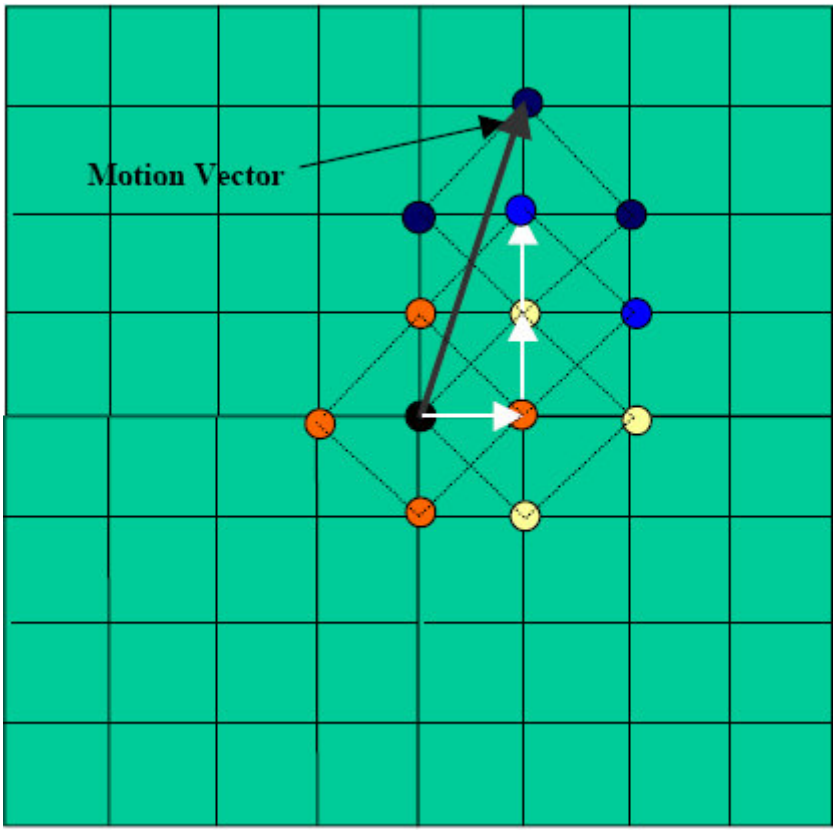


Figura 6.4 Busca em forma de losango (Diamond search).

Veja que a busca ocorre somente no ponto central do losango, e a varredura é feita aos candidatos dentre os vértices. Em seguida o melhor casamento passa a ser o centro do novo losango. Sendo ainda possível definir um número máximo de pontos de busca. O número mínimo de pontos de busca é cinco, o menor dos padrões estudados. A cada passo, são buscados em três pontos novos.

A implementação da busca se encontra no Apêndice C.

Tanto o padrão de busca Hexagonal quanto o *Diamond* foram testados e percebeu-se que o *Diamond* exigiu um esforço computacional muito menor. Por isso, esse foi o escolhido para uso na interface.

CAPÍTULO 7

CONCLUSÃO

Os objetivos propostos neste trabalho foram satisfatoriamente alcançados. Ou seja, implementou-se com sucesso uma interface visual capaz de controlar tanto o mouse como o teclado.

O controle do mouse por posição, apesar de atender satisfatoriamente os objetivos traçados, pode ainda obter grandes melhorias. A busca pelo objeto na imagem ainda é feita em toda a imagem, o que demanda muito trabalho do processador. Métodos como os usados na estimação de movimentos poderão trazer um grande ganho no desempenho da interface. Houve tentativas de se fazer essa busca mais eficiente, porém sem sucesso, provavelmente por não dispor de mais tempo para este trabalho. Outro estudo a ser feito é a implementação da técnica OWT(Overlap-Windowed Transform) [2], assim a diferença entre os pixels pode ser feita com uma operação lógica OU-EXCLUSIVO e usando um processador de 32 bits pode ser comparado até 32 pixels de uma só vez. Contudo a afirmação de que se ganhará desempenho usando essa técnica só pode ser feita após um estudo aprofundado, pois também há um gasto de desempenho apenas para se aplicar a OWT, e aqui fica apenas como sugestão. Várias dificuldades foram encontradas na implementação dessa interface.

A estimativa de movimento já esta numa fase muito madura necessitando apenas algumas mudanças no que diz respeito a implementação. Um estudo que pode ser feito é o uso da OWT pelos mesmos motivos já apresentados.

Obviamente ainda há muito a ser feito. Implementações futuras devem passar pela a utilização de um reconhecimento de padrão mais sofisticado e reconhecer movimentos mais sutis, fazendo com que a interface seja mais natural. Ou seja, ela se adapte ao usuário e não o usuário a interface.

Um outro estudo que pode ser feito é a implementação da estimação de movimento, parte do programa que mais consome recursos computacionais,

em “hardware”. Assim os recursos do computador ficarão disponíveis para a aplicação. Pode-se usar ainda esse tipo de interface para controle de um dispositivo externo.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] C. Stiller, J. Konrad, “Estimating Motion in Image Sequences – A tutorial on modeling and computation of 2D motion”, IEEE Signal Processing Magazine, Julho 1999

[2] Y.-K. Wang, G.-F. Tu, “*Fast Binary Block Matching Motion Estimation using Efficient One-Bit Transform*”.

[3] A. M. Tourapis, O. C. Au, M. L. Liou, “*Fast Motion Estimation using Circular Zonal Search*”.

[4] Ce Zhu*, Senior Member, IEEE*, Xiao Lin, Lappui Chau, and Lai-Man Po, “Enhanced Hexagonal Search for Fast Block Motion Estimation”

[5] Ce Zhu, Xiao Lin, Lap-Pui Chau, Keng-Pang Lim, Hock-Ann Ang, Choo-Yin Ong “A NOVEL HEXAGON-BASED SEARCH ALGORITHM FOR FAST BLOCK MOTION ESTIMATION”

[6] Ying, Lu , Yue Chen, Yu Wang “New Motion Estimation Algorithm Used in H.263+”

[7] GONZALEZ, Rafael C; WOODS, Richard E. “Digital **image processing”**. 2. ed. New jersey: Prentice Hall, 2002. 793 p. ISBN 0-201-18075-8

[8] MSDN Home Page, disponível em http://www.msdn.microsoft.com.

APÊNDICE A

Implementação da captura

A classe de captura é uma das principais do programa desenvolvido. Abaixo serão listadas as funções que compõe a classe e uma breve explicação do que elas fazem.

int IniciarCaptura(HWND JanelaPrincipal);

Recebe o valor do *Windows Handling* da janela principal e retorna um inteiro para indicar o erro. Caso nenhum erro tenha ocorrido, a função retorna zero. Essa função é responsável pela conexão com o driver, definição da função de *Calback*, exibição da janela, criação dos outros objetos que serão vistos em outros capítulos e já chama a função Adapta(). Esta deve ser a primeira função a ser chamada.

bool FinalizarCaptura();

Pára a captura, destrói a janela e desconecta o driver. Deve ser chamada quando não se deseja mais fazer a captura.

int Adapta();

Essa função pega as dimensões da imagem e adapta todo o programa de acordo, inclusive chamando funções semelhantes de outros objetos. Deve ser chamada toda vez que há mudança no tamanho da captura. Retorna um inteiro indicando o erro ocorrido.

bool Aloca();

Reserva espaço na memória para guardar dois frames, o atual e o anterior. Ela já é chamada pela função Adapta().

void CapturaFrame()

Captura um frame.

int RetornaLargura();

Retorna a largura do frame capturado.

int RetornaAltura();

Retorna a altura do frame capturado.

LRESULT CALLBACK FrameCallbackProc(HWND hWnd, LPVIDEOHDR lpVHdr);

É a função que é chamada cada vez que um frame é capturado.

Toda a informação necessária para o desenvolvimento dessa classe foram encontradas na pagina da MSDN da Microsoft [8].

APÊNDICE B

Implementação da Detecção de Posição

Para implementar este método primeiro foi necessário o armazenamento da máscara. Foi criada a classe **Tmascara** e seu construtor **TMascara(int Larg, int Alt),** que tem como parâmetros de entrada a largura e altura, em pixels, da máscara.

Essa classe possui os seguintes métodos:

bool Aloca()

Semelhantemente ao método **bool Aloca()** pertencente à classe Captura, porém, este reserva o espaço na memória para armazenar os dados da máscara.

void LerMascara()

Lê a máscara e salva no espaço alocado por bool Aloca(). O endereço inicial do espaço em que a máscara se encontra está no ponteiro **Pixel. Esse endereço é passado para a classe TMouseVisual.

Classe TMouseVisual

Esta é a classe principal desta aplicação por detecção de posição. O construtor da classe é **TMouseVisual(pixel **Imagem, int Largl, int Altl, pixel **Mascara, int LargM, int AltM).** Tem como parâmetros de entrada o ponteiro para o local em que está armazenado a imagem, a largura e a altura da imagem, o ponteiro para o local em que se encontra a máscara, a largura e a altura da máscara.

Os métodos pertencentes a esta classe são:

int CalculaCorrelacao(int x, int y)

Tem como entrada o ponto (x,y) a ser testado e retorna o valor da correlação entre a imagem e a máscara para este ponto.

void BuscaCorrelacao()

Esta função testa cada ponto válido da imagem. Um fato importante a ressaltar é que deve ser separada uma borda, cujo suas dimensões deve ser a metade das dimensões da máscara, na imagem em que os pontos não serão testados. Pois se fossem testados todos os pontos da imagem, haveria o seguinte problema: ao se testar o ponto (0,0) da imagem por exemplo, como a origem da máscara está em seu centro, seriam comparados valores da máscara com pontos fora da imagem. Esses pontos podem conter qualquer tipo de dado e não se refere à imagem, portanto pode ser considerado como lixo. O valor retornado por essa comparação é imprevisível, e não há relação alguma com a correlação calculada nos outros pontos. Isso trará erros ao sistema. Portanto é definida uma borda vertical e outra horizontal, e o teste vai de Borda+1 até Dimensão-Borda-1, como mostrado abaixo:

```
for(int i=BordaHoriz+1;i<LargImagem-BordaHoriz-1;i++)

    for(int j=BordaVert+1;j<AltImagem-BordaVert-1;j++)

    {

    // teste ...

    }
```

Para cada valor de correlação retornado é feita uma comparação com a máxima correlação encontrada até o momento. Se o valor retornado é maior do que o Máximo até o momento, o Máximo é atualizado com o novo valor da correlação é guardado as posições x e y do ponto em questão. Ao final da busca em toda a imagem, tem-se o máximo valor encontrado e as coordenadas deste ponto. O código é apresentado abaixo:

```
for(int i=BordaHoriz+1;i<LargImagem-BordaHoriz-1;i++)
```

```
for(int j=BordaVert+1;j<AltImagem-BordaVert-1;j++)
{
        Correlacao=CalculaCorrelacao( i , j );
        if(Correlacao>Maximo)
{
        Maximo=Correlacao;
        PosicaoX=i;
        PosicaoY=j;...
}
```

int * Resultado();

Esta função deve ser chamada após a função BuscaCorrelação. Esta função retorna um ponteiro que aponta para um vetor que contém a coordenada x em sua primeira posição e y na segunda posição.

void AtualizaMouse(int x, int y)

Recebe as coordenadas do ponto em que se encontra o objeto e posiciona o Mouse nesse ponto.

void Executa()

Esta é a função a ser chamada no programa principal, através da linha de código:

MouseVisual->Executa();

Ela chama todas as funções necessárias e já controla o Mouse. Seu código é mostrado abaixo:

```
int TMouseVisual::Executa()

{

    BuscaCorrelação();

    PontResultado=Resultado();

    AtualizaMouse(PontResultado[0] , PontResultado[1]);

}
```

A detecção por posição é feita apenas através das classes TMascara e TMouseVisual. A utilização desta classe se torma bem simples, pois a partir do momento em que as classes são instanciadas e inicializadas basta chamar a função de execução da classe TMouseVisual.

APÊNDICE C

Implementação da Busca

Essa parte do programa foi implementada em uma classe chamada DETECTARMOVIMENTO. As funções são mostradas abaixo.

void BuscaD(pixel **FrameAnterior,pixel **FrameAtual);

Implementa a busca em forma de losango descrita acima. Recebe os ponteiros para o frame anterior e o frame atual.

void Hexagonal(pixel **FrameAnterior,

pixel **FrameAtual);

Implementa a busca hexagonal descrita acima. Os parâmetros são os mesmos da função anterior.

void compensa();

Realiza a compensação de movimentos.

bool Adapta(int width,int height);

Adapta a classe às dimensões da imagem.

bool DefineTamanhoDoBloco(int x);

Define o tamanho do bloco.

int RetornaTamanahoDoBloco();

Retorna o tamanho do bloco.

int SAD(pixel **FrameA, pixel **FrameB,int xA,int yA,

int xB, int yB);

Calcula a SAD (*Sum of Absolut Difference*).

APÊNDICE D

Reconhecimento de Padrões de Movimento

O reconhecimento de padrões utilizado até agora é muito simples e se mostrou bem versátil. O campo vetorial é dividido em regiões e cada uma é analisada separadamente.

Foram definidos diversos tipos de região. Sendo que cada uma tem um tipo de ação de acordo com os movimentos. No caso de uma região que controla o teclado, define-se qual foi a direção predominante do movimento, para cima, baixo, direita ou esquerda. E associa-se uma tecla à direção.

Há sete tipos de regiões definidas, que serão apresentadas a seguir.

Tabela D.1. Tipos de região.

	Movimento Predominante			
Tipo de Região	Para cima	Para baixo	Para direita	Para esquerda
0	Sem ação	Sem ação	Sem ação	Sem ação
1	Aperta e Solta a Tecla1	Aperta e Solta a Tecla2	Aperta e Solta a Tecla3	Aperta e Solta a Tecla4
2	Não se baseia em movimentos predominantes			
3	Aperta a Tecla1	Solta a Tecla1	Aperta a Tecla2	Solta a Tecla2
4	Aperta a Tecla1	Solta a Tecla1	Solta a Tecla2	Aperta a Tecla2
5	Solta a Tecla1	Aperta a Tecla1	Aperta a Tecla2	Solta a Tecla2
6	Solta a Tecla1	Aperta a Tecla1	Solta a Tecla2	Aperta a Tecla2

A região do tipo 0 é reservada como uma região sem ação. Muitas vezes é interessante deixar um espaço entre regiões ativas. A região do tipo 1 controla até quatro teclas do teclado. A do tipo 2 não utiliza de um movimento

predominante, o controle do mouse é feito pela média dos movimentos. As regiões dos tipos 3 a 6 apertam e soltam duas teclas conforme o movimento.

Em todos os casos há um valor limite, movimentos inferiores a este valor não provocam nenhuma ação. Isso é feito para que se evite que ruídos da câmera e pequenos movimentos do usuário tenham efeitos na interface.

Tudo isso está implementado numa classe chamada Regiões. Abaixo são listadas todas as funções dessa classe e o que elas fazem.

void Adapta(int w, int h);

Recebe, respectivamente, a largura e a altura do campo de vetores de movimento e faz as adaptações necessárias dentro do objeto. Deve ser chamada toda vez que esses atributos mudam e no início do programa.

bool DefineRegiao(int reg,int tipo,int xi,

int yi,int xf,int yf, int tolerancia);

Os argumentos são, respectivamente, o numero da região, o tipo da região, a primeira posição horizontal, a primeira posição vertical, as posições finais da região e a tolerância, valor abaixo do qual, os movimentos naquela região não terão nenhum efeito. As posições devem ser dadas em porcentagens inteiras, ou seja, deve ser um numero inteiro entre 0 e 100. Como exemplo, vemos a definição de uma região que controla o mouse e usa um quarto da imagem.

DefineRegiao(1,2,0,0,50,50,20);

Como foi feito, caso a soma dos vetores seja um vetor menor, em módulo, que 20, o mouse não moverá.

bool DefineCodigos(int reg,int cod0,

int cod1,int cod2,int cod3);

Essa função termina a configuração das regiões. Ela recebe a região e outros parâmetros que devem ser interpretados de acordo com o tipo de região. A tabela D.2 resume como devem ser interpretados em cada caso.

Tabela D.2 ***Resumo dos tipos de região para controle do teclado.***

Tipo de Região	cod0	cod1	cod2	cod3
1	Unicode da tecla a ser apertada com movimento para cima.	Unicode da tecla a ser apertada com movimento para direita.	Unicode da tecla a ser apertada com movimento para esquerda.	Unicode da tecla a ser apertada com movimento para baixo.
2	Sensibilidade do mouse.	Ignorado.	Ignorado..	Ignorado.
3	Unicode da tecla a ser apertada com movimento para cima.	Ignorado.	Unicode da tecla a ser apertada com movimento para esquerda.	Ignorado.
4	Unicode da tecla a ser apertada com movimento para cima.	Ignorado.	Ignorado.	Unicode da tecla a ser apertada com movimento para baixo.
5	Unicode da tecla a ser apertada com movimento para cima.	Ignorado.	Ignorado.	Unicode da tecla a ser apertada com movimento para baixo.
6	Ignorado.	Unicode da tecla a ser apertada com movimento para direita.	Ignorado.	Unicode da tecla a ser apertada com movimento para baixo.

Deve-se tomar cuidado com os *unicodes* a serem usados, pois algumas teclas mudam em alguns padrões de teclado, por exemplo, o mesmo *unicode* usado para ";" nos teclados em inglês é usado para "ç" no teclado abnt2. Por isso, sugere-se que, sempre que possível, seja utilizadas teclas que não mudam no diversos teclados.

void SomaVetores();

Soma os vetores das regiões. Ela deve ser chamada antes da função Acoes(), vista a seguir.

void Acoes();

Põe as configurações feitas anteriormente em funcionamento.

APÊNDICE E

Interface com o usuário

Na Figura E.1 vemos a janela principal.

Figura E.1. Interface gráfica.

Os menus permitem o controle da interface.

No menu arquivo, mostrado abaixo, é possível abrir o arquivo de configuração e sair da interface.

Figura E.2. Menu Arquivo.

O menu exibir tem a opção de habilitar ou desabilitar a Sempre Visível. Caso esteja habilitada, a janela ficará sempre acima das outras.

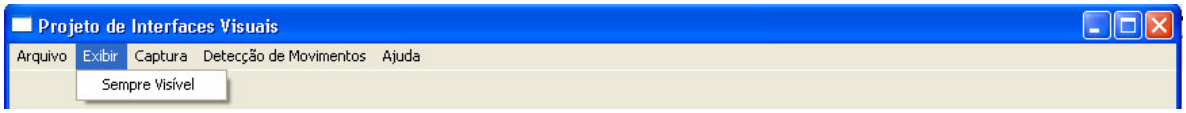


Figura E.3Menu Exibir.

O menu que controla a captura é mostrado abaixo. Ele inicia finaliza e configura a captura. Esta ultima opção não foi ainda implementada.

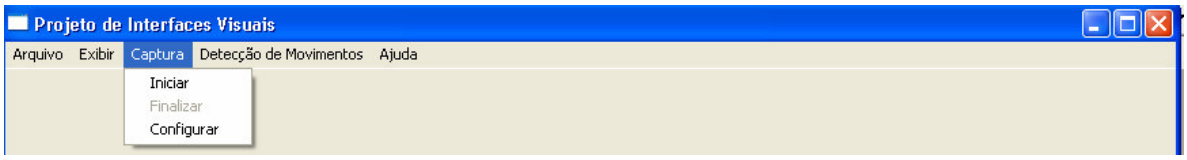


Figura E.4 Menu Captura.

Abaixo, temos o menu de Detecção de Movimentos. A primeira opção é Detectar Movimentos, que inicia a estimação dos movimentos. A opção Ações põe em execução a interface visual. Por ultimo, temos o Mostrar resultados, que mostra a compensação de movimentos.

Figura E.5 Menu Detecção de Movimentos.

Abaixo vemos uma tela de exemplo de captura e compensação de movimentos.

Figura E.6 Exemplo de captura e compensação de movimentos.

APÊNDICE F

Arquivo de Configuração.

A configuração de regiões e das ações a serem tomadas da forma como foi apresentada no capitulo anterior somente faz sentido caso o usuário possa utilizar o mesmo programa para diversas aplicações. Isso foi feito com o uso de um arquivo texto de configuração.

Este arquivo tem um formato bem definido. A estrutura do arquivo é mostrada abaixo.

[nome]

Nome da Aplicação.

[versao]

x.x

[descricao]

Descrição do que o arquivo faz.

[mapeamento]

Mapeamentos das regiões do arquivo.

[acoes]

Códigos de cada região.

[fim]

A versão e a descrição são ignoradas e somente são utilizados para facilitar o uso pelo usuário. E os dados do [mapeamento] e [acoes] seguem a mesma seqüência dos argumentos das funções DefineRegiao e DefineCodigos

vista no capitulo anterior. Abaixo podemos ver um exemplo de um arquivo que tem como objetivo escrever as teclas “a”, “s”, “f” e “g”.

[nome]

Arquivo de exemplo.

[versao]

0.1

[descricao]

Este é um arquivo de exemplo. Ele escreve as teclas as teclas “a”, “s”, “f” e “g”. de acordo com os movimentos na frente da câmera.

[mapeamento]

1 0 1 0 0 100 100 20

[acoes]

1 30 31 32 33

[fim]

Abaixo vemos um exemplo para o controle do mouse.

[nome]

Mouse.

[versao]

1.0

[descricao]

Controla o cursor do mouse.

[mapeamento]

2 2 0 0 100 100 20

[acoes]

2 45 0 0 0

[fim]