

TRABALHO DE GRADUAÇÃO

**CAPTURA DE IMAGENS E
ESTIMAÇÃO DE MOVIMENTO
EM AMBIENTE LINUX**

Daniel Emídio de Rezende

Brasília, Dezembro de 2006

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

**CAPTURE DE IMAGENS E
ESTIMAÇÃO DE MOVIMENTO
EM AMBIENTE LINUX**

Daniel Emídio de Rezende

Relatório submetido como requisito parcial para obtenção
do grau de Engenheiro Eletricista

Banca Examinadora

Prof. Ricardo L. de Queiroz, Ph.D ENE/UnB _____
(Orientador)

Prof^a. Juliana Fernandes Camapum, Ph.D _____
ENE/UnB

Prof. Docteur Ricardo Pezzuol Jacobi, CIC/UnB _____

Dedicatória

O autor dedica este projeto final ao Grupo de Processamento Digital de Sinais.

Daniel Emídio de Rezende

Agradecimentos

Daniel Emídio agradece a Deus pela vocação recebida. Ao pai, José Mauro de Rezende, meu maior incentivador e exemplo. E também a Monica Hickson por todo o apoio durante todo esse tempo.

Agradece também ao professor Ricardo de Queiroz pela orientação ao projeto. Ao Tiago Alves que dedicou um grande tempo a esclarecer as dúvidas do autor. Ao Rafael Galvão que forneceu uma valiosa contribuição bibliográfica sem a qual o projeto não seria possível.

Daniel Emídio de Rezende

RESUMO

O trabalho a seguir tem o objetivo apresentar um novo modo para os usuários do Ambiente Linux controle a posição do ponteiro do *mouse*, utilizando uma *webcam* para captura de imagens e extranindo informações de movimento destas imagens para alterar a posição do ponteiro. Para isso, o autor portou uma interface semelhante desenvolvida para o Sistema Operacional Microsoft Windows para o Ambiente Linux.

ABSTRACT

The following work presents a new way to the Linux's users to control the mouse position with the motion estimation from the images captaded from a webcam. To reach this purpose, the author started with a similar interface developed to the Microsoft Windows Operational System to the Linux enviroment.

SUMÁRIO

1	INTRODUÇÃO	1
1.1	OBJETIVOS DO PROJETO.....	1
1.2	CONTEXTUALIZAÇÃO	1
1.3	SISTEMA OPERACIONAL LINUX	2
1.3.1	DISTRIBUIÇÃO DEBIAN	4
1.4	DEFINIÇÃO DO PROBLEMA	5
2	DISPOSITIVOS DE IMAGEM E MOUSE.....	7
2.1	CAPTURE DE IMAGEM	7
2.1.1	VIDEO FOR LINUX	8
2.2	PONTEIRO DO <i>mouse</i>	9
2.3	IMPLEMENTAÇÃO.....	9
2.3.1	CAPTURE DAS IMAGENS	9
2.3.2	POSIÇÃO DO MOUSE	10
3	ESTIMAÇÃO DE MOVIMENTO E VETOR DE MOVIMENTO	14
3.1	INTRODUÇÃO	14
3.2	TÉCNICAS DE ESTIMAÇÃO DE MOVIMENTO	14
3.2.1	COMPARAÇÃO POR CORRELAÇÃO.....	14
3.2.2	COMPARAÇÃO POR SAD - SOMA DAS DIFERENÇAS ABSOLUTAS.....	15
3.3	TÉCNICAS DE CRIAÇÃO DE VETOR DE MOVIMENTO.....	16
3.3.1	MODELOS DE MOVIMENTO	16
3.3.2	BUSCA DIAMANTE.....	18
3.3.3	BUSCA HEXAGONAL	19
3.3.4	BUSCA CIRCULAR.....	20
3.4	IMPLEMENTAÇÃO.....	22
3.4.1	DETECÇÃO DE MOVIMENTO	22
3.4.2	MAKEFILE.....	22
4	CONCLUSÕES	24

REFERÊNCIAS BIBLIOGRÁFICAS.....	27
ANEXOS.....	28

LISTA DE FIGURAS

1.1	Arquitetura do projeto	6
3.1	Campo vetorial tridimensional representado por duas dimensões.	16
3.2	Exemplos de campo de vetores de movimento e a compensação seguindo alguns modelos. (a) translacional, (b) <i>affine</i> , (c) projeção linear e (d) quadrática	17
3.3	Modelo de busca diamante.	18
3.4	Modelo de busca hexagonal.	19
3.5	Modelo de busca hexagonal melhorado.	20
3.6	Modelo de busca circular.	21
4.1	Imagem apresentada com o conflito do OPENGL.....	25
4.2	Interface do projeto	26

LISTA DE TABELAS

3.1	Modelos de movimento	17
-----	----------------------------	----

1 INTRODUÇÃO

Este capítulo apresenta os objetivos do projeto de graduação e a descrição simplificada do projeto, assim como as etapas de seu desenvolvimento, além de apresentar o sistema operacional Linux.

1.1 OBJETIVOS DO PROJETO

Neste projeto de graduação, o principal objetivo do autor é ganhar experiência com o sistema operacional LINUX e aprender a programar neste sistema operacional e também como manipular arquivos dos dispositivos como o *mouse* e *webcam*. O segundo objetivo do projeto é adquirir noções de processamento de imagens, como estimação de movimento. Para isso será desenvolvido uma interface, em ambiente Linux, que permitirá o usuário controlar a posição do *mouse* com movimentos livres em frente a uma *webcam*.

1.2 CONTEXTUALIZAÇÃO

Com as tentativas de popularização do Sistema Operacional Linux, novas ferramentas precisam ser desenvolvidas para que sua interface com o usuário fique cada dia mais simples e similar à interface do sistema operacional *Microsoft Windows*, o sistema operacional presente na grande maioria dos computadores atualmente. Exemplos destas novas ferramentas são as interfaces do usuário e computador é a captura de imagens de uma *webcam*, que são principalmente utilizadas para comunicação entre pessoas pela internet utilizando programas como *MSN Messenger* e *Skype*. Aplicações com esta ferramenta ainda não são bem desenvolvidas para o sistema operacional em estudo. Estes aplicativos popularizaram as *webcams*, que estão cada vez mais simples de instalar, melhor qualidade de imagem, mais funcionalidades e preços mais acessíveis.

O presente projeto busca o desenvolvimento de uma nova funcionalidade ao usuário do Linux: Controlar a posição do ponteiro do *mouse* utilizando a mão livre em frente a uma *webcam*. Alguns filmes de ficção científica, como *Blade Runner* e *Minority Report* já ilustraram esta aplicação para o dispositivo.

1.3 SISTEMA OPERACIONAL LINUX

O Linux é um sistema operacional para computadores baseado em *UNIX*, e é um dos melhores exemplos de *software* de código aberto e gratuito - seus códigos fontes estão disponíveis para todos usarem, modificarem e redistribuir livremente.

Inicialmente, o sistema operacional foi desenvolvido para ser utilizado em computadores pessoais, mas recebeu atenção de grandes corporações como *IBM*, *Sun Microsystems*, *Hewlett-Packard*, *Novel Inc.* e suas aplicações aumentaram e o Linux passou a ser um sistema operacional para grandes servidores, chegando a marca de oitenta por cento dos grandes servidores de empresas utilizam este sistema em seus servidores *web*.

O Linux está ampliando suas fronteiras, deixando de ser sistema operacional de computadores passando a ser o núcleo de processamento de super computadores, telefones celulares, video games, aumentando sua participação no mercado.

O Kernel do Linux foi, originalmente, escrito por Linus Torvalds do Departamento de Ciência da Computação da Universidade de Helsinki, Finlândia, com a ajuda de vários programadores voluntários através da Internet.

Linus Torvalds começou o desenvolvimento do kernel (ou núcleo) como um projeto particular, inspirado pelo seu interesse no Minix, um pequeno sistema UNIX desenvolvido por Andrew S. Tanenbaum. Ele limitou-se a criar, nas suas próprias palavras: Um Minix melhor que o Minix. No dia 5 de outubro de 1991 Linus Torvalds anunciou a primeira versão oficial do Linux, versão 0.02, chamado de FREAX. Desde então muitos programadores têm ajudado-o a fazer do Linux o sistema operacional que é hoje.

O Linux possui um kernel monolítico. Isto significa que as funções do tipo agendamento de processos, gerenciamento de memória, operações de entrada e saída, acesso ao sistema de arquivos ... são executadas no espaço do kernel. Uma característica do Linux é que algumas das funções (drivers de dispositivos, suporte à rede, sistemas de arquivo, por exemplo) podem ser compiladas e executadas como módulos LKM - *Loadable Kernel Modules*, que são bibliotecas compiladas separadamente da parte principal do kernel e podem ser carregadas e descarregadas após o kernel estar em execução.

Inicialmente, Torvalds lançou o Linux sob uma licença que proibia qualquer uso comercial. Isso foi mudado de imediato para a Licença Pública Geral - GNU (*General Public License*). Essa licença permite a distribuição e mesmo a venda de versões possivelmente modificadas do Linux

mas requer que todas as cópias sejam lançadas dentro da mesma licença e acompanhadas do código fonte. Apesar de alguns dos programadores que contribuem para o kernel permitirem que o seu código seja licenciado com GPL versão 2 ou posterior, grande parte do código (incluído as contribuições de Torvalds) menciona apenas a GPL versão 2.

Logo que Linus Torvalds passou a disponibilizar o Linux, ele apenas disponibilizava o núcleo com alguns comandos básicos. O próprio usuário devia encontrar os outros programas, compilá-los e configurá-los e, talvez por isso, o Linux tenha carregado consigo a etiqueta de sistema operativo apenas para técnicos. Foi neste ambiente que surgiu a MCC (Manchester Computer Centre), a primeira distribuição Linux, feita pela Universidade de Manchester, na tentativa de poupar algum esforço na instalação do Linux.

Desde o começo, o núcleo Linux era inútil sem os utilitários GNU. De fato, o núcleo é apenas uma parte de um sistema operacional utilizável: são necessários também vários outros componentes como bibliotecas de funções, interpretadores de comandos, utilitários e mesmo, em última instância, aplicativos como compiladores e editores de texto.

Todos esses já vinham sendo reunidos pelo Projeto GNU da Fundação Software Livre - *Free Software Foundation*, que embarcara num subprojeto que ainda continua para obter um núcleo, o Hurd. Dada a demora no subprojeto do núcleo GNU, o Linux veio a constituir um sistema operacional completo híbrido, o GNU/Linux.

Atualmente, um Sistema Operacional GNU/Linux completo é uma coleção de software livre (e não-livres algumas vezes) criados por programadores solitários ou em grupos e organizações de todo o mundo, tendo o Linux como seu núcleo. Companhias como a *Red Hat*, a *SuSE*, a *Mandriva* (união da Mandrake com a Conectiva) e *Patrick Volkerding* bem como projetos de comunidades como o **Debian** ou o *Gentoo*, compilam o *software* e fornecem um sistema completo, pronto para instalação e uso.

As distribuições de GNU/Linux começaram a receber popularidade desde a segunda metade dos anos 90, como uma alternativa livre para os sistemas operacionais *Microsoft Windows* e *Mac OS*, principalmente por parte de pessoas acostumadas com o Unix na escola e no trabalho. O sistema tornou-se popular no mercado de Desktops e servidores, principalmente para a Web e servidores de bancos de dados.

No decorrer do tempo, várias distribuições surgiram e desapareceram, cada qual com sua característica. Algumas distribuições são maiores outras menores, dependendo do número de aplicações e sua finalidade. Algumas distribuições de tamanhos menores cabem numa disquete

com 1,44 MB, outras precisam de vários CDs, existindo até algumas versões em DVD. Todas elas tem o seu público e sua finalidade, as pequenas (que ocupam poucas disquetes) são usadas para recuperação de sistemas danificados ou em monitorizações de redes de computadores.

1.3.1 Distribuição Debian

O Debian foi fundado em 1993 por Ian Murdock, ao tempo estudante universitário, que escreveu o Manifesto Debian que apelava à criação de uma distribuição Linux a ser mantida de uma maneira aberta, segundo o espírito do Linux e do GNU. O nome Debian vem dos nomes dos seus fundadores, Ian Murdock e de sua mulher, Debra.

O Projeto Debian cresceu vagarosamente e lançou suas versões 0.9x em 1994 e 1995, quando dpkg ganhou notoriedade. Os primeiros ports para outras arquiteturas iniciaram em 1995, e a primeira versão 1.x do Debian aconteceu em 1996.

Bruce Perens substituiu Ian Murdock como líder do projeto. Ele iniciou a criação de vários documentos importantes (o contrato social e o free software guidelines) e a legítima umbrella organization (SPI), bem como liderou o projeto através dos lançamentos das versões da ELF/libc5 (1.1, 1.2, 1.3).

Bruce Perens deixou o projeto em 1998 antes do lançamento da primeira versão Debian baseada em glibc, a 2.0. O Projeto continuou elegendo novos líderes e fazendo mais duas versões 2.x, cada qual incluindo mais ports e mais pacotes. APT foi lançada durante este tempo e o Debian GNU/Hurd também iniciou-se.

O ano de 1999 trouxe as primeiras distribuições Linux baseadas em Debian, Corel Linux e Stormix's Storm Linux, hoje descontinuadas mas que iniciaram o que é hoje uma notável tendência às distribuições baseadas em Debian.

Perto do ano 2000, o projeto se direcionou ao uso de repositórios de pacotes e à distribuição teste, alcançando um marco maior no que se refere aos arquivos e o gerenciamento de lançamentos. Em 2001, os desenvolvedores iniciaram conferências anuais, Debconf, com conversas, workshops, e a recepção aos usuários técnicos. A versão 3.0 de 2002 incluiu mais do que o dobro do número de pacotes da versão anterior e estava disponível para cinco novas arquiteturas.

O ciclo de desenvolvimento das versões do Debian passa por três fases:

- *Unstable* - instável;
- *Testing* - teste;
- *Stable* - estável.

Quando as versões estão na fase de teste são identificadas por codinomes tirados dos personagens do filme Toy Story. Ao se tornarem estáveis, as versões recebem um número de versão (ex: 3.1).

1.4 DEFINIÇÃO DO PROBLEMA

Para elaborar o projeto, o autor dividiu-o em três etapas:

- i CAPTURA DE IMAGEM DA *WEBCAM*: Utilizando seus *drivers* genéricos do dispositivo para linux e a biblioteca **V4L - Video for Linux**. Os quadros serão capturados a uma taxa constante e armazenados em *buffer* para que seja retirada a informação de movimento;
- ii ESTIMAÇÃO DE MOVIMENTO: Uma vez a imagem capturada e armazenada, rotinas de estimação de movimento detectarão os deslocamentos na imagem, criando o vetor de movimento;
- iii MOVIMENTO DO PONTEIRO DO *MOUSE*: O vetor de movimento deverá ser passado para o arquivo do *mouse*, fazendo que o seu ponteiro movimente-se na tela de acordo com os gestos;

A arquitetura do projeto é ilustrada na figura 1.1:

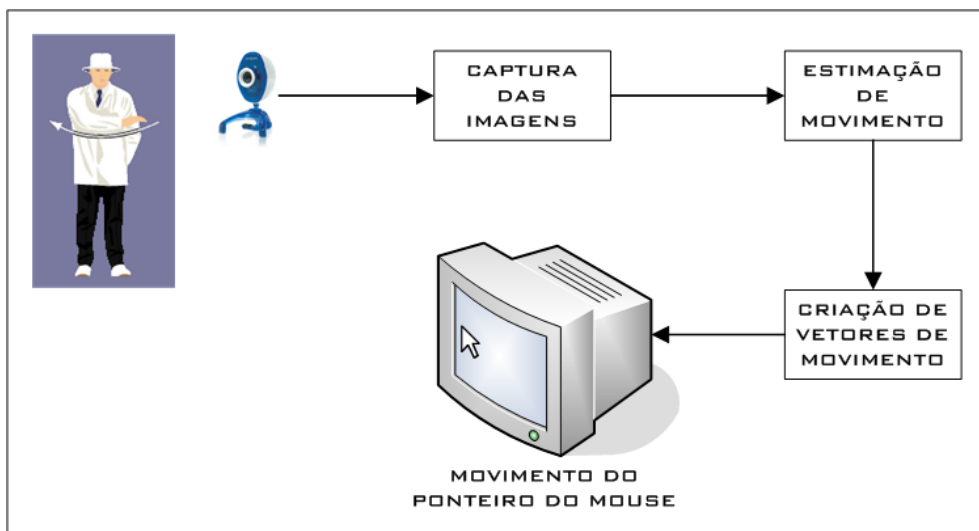


Figura 1.1: Arquitetura do projeto

2 DISPOSITIVOS DE IMAGEM E MOUSE

Este capítulo apresenta como o primeiro item foi solucionado, a captura de imagem da webcam e como o linux interpreta o mouse.

2.1 CAPTURA DE IMAGEM

O dispositivo escolhido para capturar as imagens do usuário foi a *webcam* modelo **WebCam NX** fabricada pela **CREATIVE**. A escolha foi feita devido ao empréstimo de um exemplar para o autor pelo Grupo de Processamento Digital de Sinais - GPDS.

O primeiro passo para capturar as imagens foi instalar o *driver* da *webcam*. Infelizmente, o fabricante do dispositivo, não havia desenvolvido seu *driver* para o sistema operacional Linux até o início do projeto. Então foi feita uma pesquisa de *drivers* genéricos para que o dispositivo fosse reconhecido pelo sistema.

O *driver* utilizado foi o *spca5xx - Softwares Package for Camera Adaptator*, versão 0.57, desenvolvido pelo professor Michel Xhaard, Docteur en Electronique Université Pierre et Marie Curie. Este driver utiliza a biblioteca **V4L - Video for Linux**. Ele pode ser obtido no sítio do seu desenvolvedor: <http://mxhaard.free.fr/download.html>.

Sua instalação não foi simples, mas seguindo os passos descritos nos arquivos README e INSTALL, assim o *driver* foi corretamente configurado:

Antes de conectar a *webcam* ao computador, o arquivo `spca5xx-20060501.tar.gz` foi descompactado utilizando o comando:

```
$ tar -xzvf spca5xx-20060501.tar.gz
```

onde `tar` é o programa que descompacta o arquivo e `-xzvf` são os parâmetros:

x - Extrai um arquivo;

z - Arquivo no formato gzip (extensão tar.gz);

v - Imprime na tela os detalhes da operação;

f - Especifica o arquivo tar a ser operado.

O próximo passo foi limpar os objetos pré-compilados presentes no arquivo compactado com

o comando:

```
$ make clean;
```

A seguir compila-se o *driver* com o comando:

```
$ make ;
```

E finalmente, o instala, alterando o usuário para **root** (o usuário com poderes ilimitados, que é capaz de instalar programas no Linux:

```
$ su ;
```

(senha do usuário root) ;

```
# make install;
```

A câmera foi conectada ao computador e a instalação do *driver* foi testada com sucesso utilizando o programa `xawtv`, que abre o dispositivo de imagem e disponibiliza na tela do computador os quadros capturados pela câmera.

2.1.1 Video for Linux

Video for Linux - V4L foi a primeira API - *Application Programming Interface* (Interface de Programação de Aplicativos) desenvolvida para fornecer uma interface comum para os dispositivos para captura de vídeo. A grande maioria destes dispositivos placas PCIs - *Peripheral Component Interconnect* (Interconector de componentes Periféricos) e *webcams* USB *Universal Serial Bus* (Barramento Universal Serial) são compatíveis com esta interface.

Esta API é simples e todo o seu processamento acontece no kernel. O V4L fornece rotinas como `open()`, `close()` e `ioctl()` que são utilizadas para iniciar, fechar e controlar o dispositivo, respectivamente. A rotina `ioctl()` é responsável pela configuração do dispositivo e requisição de novos quadros de imagem.

Ultimamente, uma nova API de vídeo, o **V4L2** - *Video for Linux 2*, está em desenvolvimento para complementar o suporte sistemas mais complexos e consertar algumas limitações do V4L original. Esta API ainda está em desenvolvimento e aplicações criadas com o V4L são compatíveis a segunda versão devido a nova rotina `ioctl()` possuir uma camada de compatibilidade com a anterior. Mas aplicações desenvolvidas no V4L2 não funcionarão corretamente na versão anterior.

2.2 PONTEIRO DO MOUSE

Assim como todos os dispositivos, o *mouse* é interpretado como um arquivo do Linux. Assim é possível ler e escrever neste arquivo. O arquivo que o controla nas versões 2.6 do *kernel* era `/dev/psaux` para dispositivo ps2 e `/dev/ttySn` para *mouse serial*.

Nas versões posteriores o arquivo que controla o *mouse* passou a ser o `/dev/input/mice`, um dispositivo virtual que substituiu com uma grande vantagem o antigo `/dev/psaux` - a possibilidade de permitir múltiplos *mice*. Cada *mouse* tem um arquivo próprio de controle `/dev/input/mouse0`, `/dev/input/mouse1`, `/dev/input/mouse2` ... conforme a necessidade. Os eventos destes arquivos são repetidos no arquivo `/dev/input/mice`. Uma excelente função se considerarmos os *USB mice plug&play*.

Porém existem algumas desvantagens. Infelizmente, elas são notáveis tais como alteração na velocidade do ponteiro. Hora seu movimento é mais rápido, hora mais lento. Isto acontece porquê agora o *kernel* processa as informações dos *mice* e as recalcula na maneira que ele decide ser conveniente, o que pode ser contornado modificando alguns parâmetros.

2.3 IMPLEMENTAÇÃO

2.3.1 Captura das Imagens

Para implementar o código de captura de vídeo foi utilizado como base o arquivo `CapturaImagens.cpp`, desenvolvido pelos autores do projeto final: *Implementação de Interface Visual Baseada em Estimação de Movimentos e Posição para Controle de Mouse e Teclado*, JESSÉ BERTO ANDRADE e RAFAEL GALVÃO DE OLIVEIRA, de junho de 2005. Os autores cederam gentilmente seus códigos fontes desenvolvidos para o *Sistema Operacional Windows*. Todo o processo de portar o arquivo para o ambiente Linux foi desenvolvido com a ajuda de fóruns, lista de discussão na internet, *e-mails*, trocando experiências com outros desenvolvedores de interface de captura de imagem para o sistema operacional idealizado por Linus Torvalds.

Primeiramente, foi criado o arquivo `main.cpp`. Nele encontramos a função `aplicacao::aplicacao()`, responsável por verificar se o se existem algum erro para alocar memória para a captura de imagem, e se a *webcam* está corretamente conectada ao computador. O arquivo também é responsável por chamar o arquivo `CapturaImagens.cpp`, que por sua vez chamará as demais funções do projeto. Maiores detalhes dos arquivos podem ser encontrados

em seus códigos fontes, encontrados no CD em anexo ao projeto.

No arquivo `CapturaImagens.cpp` é definido os ponteiros e variáveis utilizadas para capturar a imagem, tais como o ponteiro para as coordenadas horizontais e verticais *pixel* atual, tamanho do quadro a ser capturado, que característica de modelo da câmera utilizada, e o formato de vídeo. Normalmente o tamanho formato do vídeo é o CIF - *Common Intermediate Format*, Formato Comum Intermediário, com dimensões de 352x288.

A função `int CAPTURA::IniciarCaptura()` é responsável por abrir o arquivo de vídeo, definido em seu arquivo cabeçalho (`CapturaImagens.h`, como `(#define VFILE "/dev/video0")`). Após verificar se não ocorreu nenhum erro iniciar a captura de imagem, passamos para o laço para a captura dos próximos quadros. Para esta finalidade, foi criado a função `void CAPTURA::capture_next (void)`. Ela é responsável para capturar o quadro atual, armazenando o quadro anterior. As imagens e armazena-las já em escala de cinza, para que as funções de detecção de movimento possam ser utilizadas. O programa fica neste laço até o comando de terminar o programa, então a função `bool CAPTURA::FinalizarCaptura()` fecha o arquivo do dispositivo de vídeo.

Outro atributo deste arquivo é alocar memória suficiente para armazenar as imagens capturadas, a função `bool CAPTURA::Aloca()` é a responsável por tal atividade. Ao encerrar o programa, a função `CAPTURA::CAPTURA()` libera a memória para o sistema operacional.

A principal mudança para portar o programa foi alterar a biblioteca de vídeo utilizada, `vfw` - Video for Linux originalmente para `V4L` - *Video for Linux*.

A biblioteca `V4L` é capaz de chamar funções que recebem informações do dispositivo de vídeo, como modelo, fabricante, dimensões da imagem, formato, cor, brilho, contraste e todas as informações da imagem. Todas estas funções são solicitadas em `void CAPTURA::open_v4l (char *device)`.

2.3.2 Posição do mouse

Para controlar a posição do ponteiro do *mouse*, foi utilizado o arquivo `regiao.cpp`, desenvolvido para o ambiente *Microsoft Windows*, pelos autores já citados. O programa original, possui mais funcionalidades, como simular alguns botões do teclado, diferentes áreas de detecção com funcionalidades distintas.

O primeiro passo para iniciar a portabilidade para o Sistema Operacional Linux, foi identificar

os trechos do código que controlam o *mouse* e altera-lo, conforme a necessidade.

Para simular movimentos do ponteiro, foi utilizado o programa gratuito `swinput`, versão 0.7 que altera a posição do *mouse* com comandos no terminal do Linux. O programa está disponível no enlace: <http://savannah.nongnu.org/projects/swinput>.

Com o programa salvo no computador, o arquivo foi descompactado, com o programa `tar`, e instalando-o, com procedimentos semelhantes aos descrito anteriormente. Com o programa corretamente descompactado e instalado, o arquivo `README` explica como operar o *mouse* por comandos:

Primeiramente, foi preciso criar o dispositivo virtual de *mouse*, com o comando:

```
# mknod /dev/swmouse c 10
```

onde:

`mknod` é o comando para se criar arquivos do tipo nó;

`/dev/swmouse` é o arquivo do dispositivo;

`c 10` são as opções para criar o arquivo de dispositivo, e 10 é o número menor do arquivo, conforme as instruções do arquivo.

E para movimentar o ponteiro, com este programa, é necessário escrever os comandos neste arquivo `/dev/swmouse`, conforme o exemplo:

```
# echo "u 50" > /dev/swmouse
```

Este comando move o ponteiro para cima (`u`, do inglês *up*), 50 pontos na tela. Para obter movimentos para outras direções, e com valores diferentes, basta alterar o comando acima para `d` (*down*), `l` (*left*) e `r` (*right*). e a quantidade de pontos a ser deslocado.

Desvendado como operar com o *mouse* virtual, o autor dedicou-se a alterar o programa `regiao.cpp` para funcionar no ambiente desejado. As principais alterações neste arquivo foram as seguintes:

No arquivo original, os comandos para movimentar o ponteiro são os seguintes:

```

    input[0].type = INPUT_MOUSE;
input[0].mi.time = 0;
input[0].mi.dx = SomaX[i]*Tolerancia[i]/(Width*Height);
input[0].mi.dy = (-1)*SomaY[i]*Tolerancia[i]/(Width*Height);
input[0].mi.mouseData = 0;
input[0].mi.dwExtraInfo = 0;
input[0].mi.dwFlags = MOUSEEVENTF_MOVE;
SendInput(1, input, sizeof(INPUT));

```

Já na primeira versão novo arquivo `regiao.cpp` para Linux, o vetor deslocamento, criado pela estimação de movimento das imagens, é passado para o dispositivo com os seguintes comandos:

```

for(int i=0;i<1;i++)

{ int dx = (int) SomaX[i]*Tolerancia[i]/(Width*Height);
int dy = (int) (-1)*SomaY[i]*Tolerancia[i]/(Width*Height);

    if(dx >= 0)
{
// Se o deslocamento horizontal for maior do que 0, o mouse se deslocará para a direita

    sprintf(string,"r %d",i); // cria a string para escrever no arquivo /dev/swmouse, r é o comando
para o movimento para a direita (right)
    write(fdM,string,10); // Escreve o comando no dispositivo
}
...

```

Alterando-se os comandos `u`, `d`, `l`, `r`, e as variáveis `dx`, `dy` de acordo com o sentido do movimento.

onde `fdM` foi definido para abrir o arquivo do dispositivo:

```
fdM = open("/dev/swmouse", O_RDWR)
```

A primeira versão do programa, apresentou movimentos muito rápidos e instáveis, então foram propostos os seguintes ajustes no código para suavizar os deslocamentos:

- **DESCARTE DA METADE DOS VETORES DE MOVIMENTOS ESTIMADOS:** Foi introduzido uma nova variável que habilita ou não a manipulação do arquivo de dispositivo. Quando a variável era positiva, o comando era passado para o dispositivo, e então a variável era alterada para um valor negativo, e no próximo laço, não se escreve no arquivo `/dev/swmouse` e seu valor volta a ser positivo;
- **FATOR DE CORREÇÃO DE DX E DY:** Além de diminuir a quantidade de vetores a serem passados para o dispositivo, foi aplicado um fator de correção, para reduzi-los. Após alguns testes realizados, os vetores foram divididos por 1,3. Assim os movimentos ficaram satisfatoriamente suaves;
- **VETOR DE MOVIMENTO NULO:** Percebeu-se que quando não existia movimento na imagem ($dx = dy = 0$), o ponteiro continuava-se deslocando. Testou-se o comando `# echo "r 0" > /dev/swmouse` no terminal e foi constatado que ao solicitar um deslocamento de zero pontos para a direita, o *mouse* deslocava 1 ponto para a direita. Então alterou-se o código para escrever no dispositivo quando o deslocamento for um número não nulo;
- **CRIAÇÃO DE ZONA MORTA:** Para que o *mouse* pudesse ficar estável numa posição, uma zona morta para que movimentos menores de 3 pontos, não fosse passado para dispositivo. E para obter transições mais leves, dx e dy são escritos no dispositivos subtraídos de 3 unidades.

Implementando as alterações propostas, o programa ficou muito mais suave e estável. O novo código do programa `regiao.cpp` pode ser visto em anexo.

3 ESTIMAÇÃO DE MOVIMENTO E VETOR DE MOVIMENTO

Este capítulo apresenta a solução do segundo desafio, a estimação de movimento e a criação do vetor movimento.

3.1 INTRODUÇÃO

Uma das técnicas para estimar o movimento de uma imagem é primeiramente determinar o objeto de referência que terá o seu movimento estimado. Se o objeto tiver pequenas dimensões se comparado com a dimensão total da imagem, ele pode ser tratado como uma sub imagem.

O algoritmo de detecção da posição da imagem necessita de uma máscara que contém a forma do objeto. Este algoritmo compara a sub imagem armazenado na máscara com toda a da imagem original, e o lugar da imagem que mais se assemelha com a máscara é escolhido como a posição que o objeto está.

A cada novo quadro capturado, o algoritmo de detecção encontra o objeto especificado. Se o objeto estiver uma posição diferente do quadro anterior, a diferença entre as duas coordenadas dos pontos nos dois instantes é definido como **vetor de movimento**.

Mas também é possível determinar o movimento da imagem como um todo, ou áreas pré-definidas. Onde cada macrobloco da imagem anterior é procurada na imagem atual. Assim, o vetor de movimento é criado analisando-se toda a imagem. Existem diferentes maneiras de se determinar qual é o ponto que mais se aproxima da máscara. Veremos aqui dois métodos que foram propostos para tal finalidade.

3.2 TÉCNICAS DE ESTIMAÇÃO DE MOVIMENTO

3.2.1 Comparação por correlação

A correlação é definida pelos estatísticos como a interdependência de duas ou mais variáveis. Para a detecção de posição pode ser utilizada como a interdependência entre a sub imagem e a imagem, ou um macrobloco anterior com o atual. E é calculada segundo a expressão 3.1:

$$C(x, y) = \sum_s \sum_t f(s, t) \times w(x + s, y + t) \quad (3.1)$$

onde $x = 0, 1, 2, \dots, M - 1$, $y = 0, 1, 2, \dots, N - 1$, $s = 0, 1, 2, \dots, J - 1$ e $t = 0, 1, 2, \dots, K - 1$ e M é a largura e N é a altura da imagem em *pixels* e J e K são, respectivamente, as dimensões do macrobloco.

Para obter valores de correlação mais amplos, utiliza-se níveis de cinza da imagem variando de -127 até +128, ao invés dos tradicionais 0 a 255. Assim quando a correlação for calculada em regiões onde o macrobloco não está contido aparecem valores positivos e negativos que tentem a se anularem quando somados. Mas quando o macrobloco é comparado com o melhor candidato, $f(a, b) = w(a, b)$ o valor retornado pela equação 3.1 é máximo. Assim o melhor candidato é determinado com mais exatidão.

3.2.2 Comparação por SAD - Soma das Diferenças Absolutas

A SAD - *Sum of Absolute Differences* é determinada pela equação 3.2:

$$SAD(x, y, s, t) = \sum_i \sum_j \|A_{(x+i, y+j)} - B_{((x+s)+i, (y+t)+j)}\| \quad (3.2)$$

onde $x = 0, 1, 2, \dots, M - 1$, $y = 0, 1, 2, \dots, N - 1$, $s = 0, 1, 2, \dots, J - 1$ e $t = 0, 1, 2, \dots, K - 1$ e M é a largura e N é a altura da imagem em *pixels* e J e K são, respectivamente, as dimensões do macrobloco.

Como pode ser visto, o cálculo da SAD é mais simples e rápido do que o cálculo da correlação, pois a única operação necessária é a adição. Porém a diferença entre o melhor candidato e os outros prováveis é pequena, o que pode afetar na determinação verdadeiro.

Se esta técnica for utilizada para localizar objetos em imagens ela será ineficiente se a sub imagem for definida sob certas condições de iluminação e posição da *webcam* e com o passar do tempo essas condições se alteram, logo a probabilidade de se encontrar a sub imagem na imagem é muito pequena.

Mas neste projeto, não foi utilizado a técnica de localizar um objeto, e sim estimar o mo-

vimento de toda a imagem, comparando o quadro anterior com o atual. Assim, o método de comparação com o métodos da soma das diferenças absolutas é vantajoso, pois as condições do ambiente não se alteram bruscamente entre dois quadros consecutivos.

3.3 TÉCNICAS DE CRIAÇÃO DE VETOR DE MOVIMENTO

Para criar o vetor de movimento, precisamos lembrar que no mundo real, estamos no espaço tridimensional e a *webcam* captura as imagens apenas com duas dimensões. Então todos os movimentos no mundo real devem ser representados no espaço bidimensional.

Abaixo, a figura 3.1 que exemplifica o movimento em três dimensões representado em apenas duas:

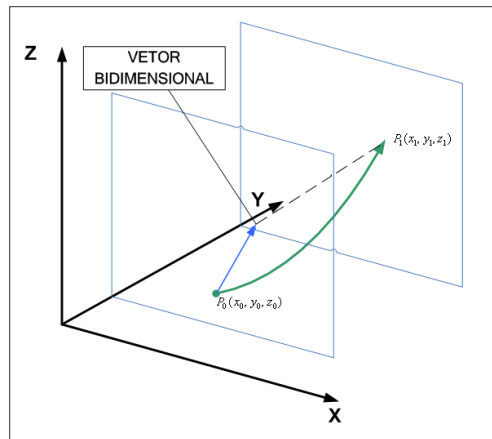


Figura 3.1: Campo vetorial tridimensional representado por duas dimensões.

Ao capturar a imagem, perdemos uma dimensão, vários movimentos no mundo real, podem gerar o mesmo vetor de movimento bidimensional, logo, não existe unicidade entre os movimentos reais e os vetores de movimento.

3.3.1 Modelos de movimento

Existem alguns modelos utilizados na compensação de movimento, tais como translacional, *affine*, projeção linear e quadrática, como podem ser visualizados na figura 3.2, e a tabela 3.1 apresenta tais modelos:

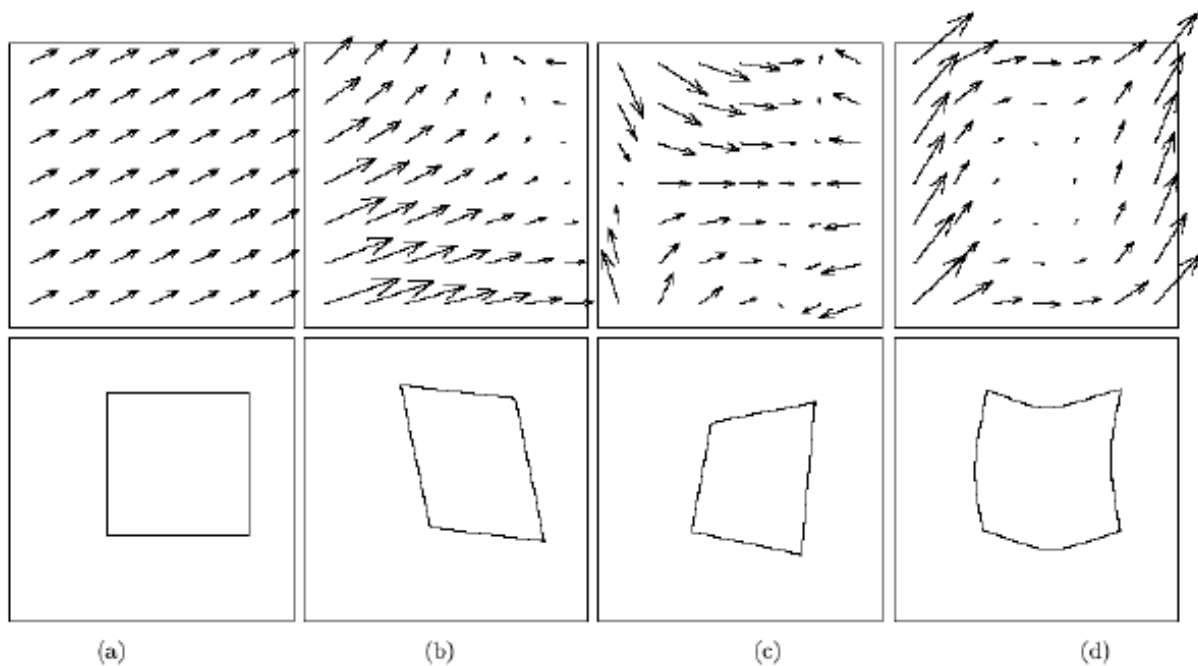


Figura 3.2: Exemplos de campo de vetores de movimento e a compensação seguindo alguns modelos. (a) translacional, (b) *affine*, (c) projeção linear e (d) quadrática

Tabela 3.1: Modelos de movimento

TIPO	PARÂMETROS	CAMPO DE MOVIMENTO
TRANSLACIONAL	2	$d(x) = (a_1, b_1)^T$
AFFINE	6	$d(x) = \begin{pmatrix} a_1 & a_2 \\ b_1 & b_2 \end{pmatrix} x + \begin{pmatrix} a_3 \\ b_3 \end{pmatrix}$
PROJEÇÃO LINEAR	8	$d(x) = \begin{pmatrix} \frac{a_1 + a_2x + a_3y}{1 + a_4x + b_4y} \\ \frac{b_1 + b_2x + b_3y}{1 + a_4x + b_4y} \end{pmatrix} - x$
QUADRÁTICA	12	$d(x) = \begin{pmatrix} a_1 + a_2x + a_3y + a_6x^2 + a_5xy + a_4y^2 \\ b_1 + b_2x + b_3y + b_6x^2 + b_5xy + b_4y^2 \end{pmatrix}$

No projeto desenvolvido, os vetores de movimento serão criados seguindo o modelo translacional, que representa o movimento rígido sobre uma projeção ortográfica. Esta projeção resulta em movimentos de duas dimensões espacialmente constante.

3.3.2 Busca diamante

Como já foi explicado como se compara a imagem anterior com a atual para estimar o movimento, uma maneira eficiente para realizar a comparação precisa ser escolhida. Podemos comparar cada macrobloco da imagem anterior com todos da imagem atual. Assim obtemos, com o menor erro possível, o movimento da imagem. Mas esta solução demanda um enorme esforço computacional, o que pode inviabilizar que o programa trabalhe em tempo real, essencial para aplicações de interfaces visuais.

Outra estratégia mais interessante, pela sua simplicidade e eficiência para realizar a busca é a losango ou mais conhecida como diamante. A busca começa pelo centro de um losango, a varredura segue pelos seus vértices. O vértice onde que possui o melhor casamento das duas imagens acontece, torna-se o centro do próximo losango. Esta busca pode continuar até um valor pré-definido de passos, sendo que o mínimo de comparação é cinco, pois a cada passo são realizadas três comparações. Outra forma do algoritmo encerrar a comparação é quando o centro de um losango possuir menor SAD do que os seus vértices, assim novos losangos diferentes não podem ser formados.

A figura 3.3 ilustra a busca diamante:

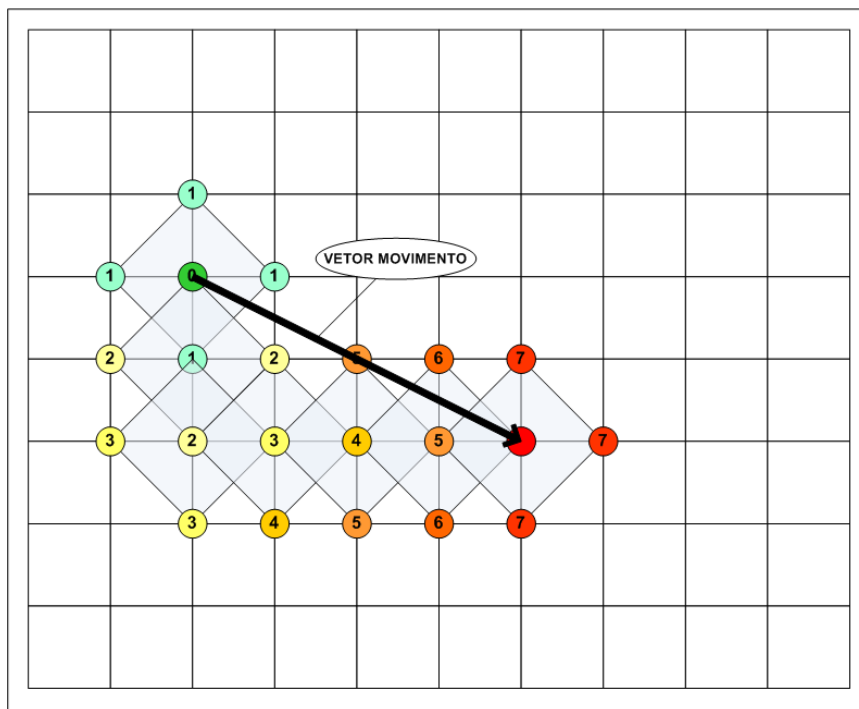


Figura 3.3: Modelo de busca diamante.

3.3.3 Busca hexagonal

Um segundo método de busca utilizado é o hexagonal. Nesta busca, o centro do hexágono é comparado com os seus seis vértices. O vértice que possuir a menor SAD, será o novo centro do próximo hexágono. A busca continua até o macrobloco seja o mais parecido com o original. Pela figura 3.4 concluímos que apenas são necessárias três novas comparações para cada novo hexágono.

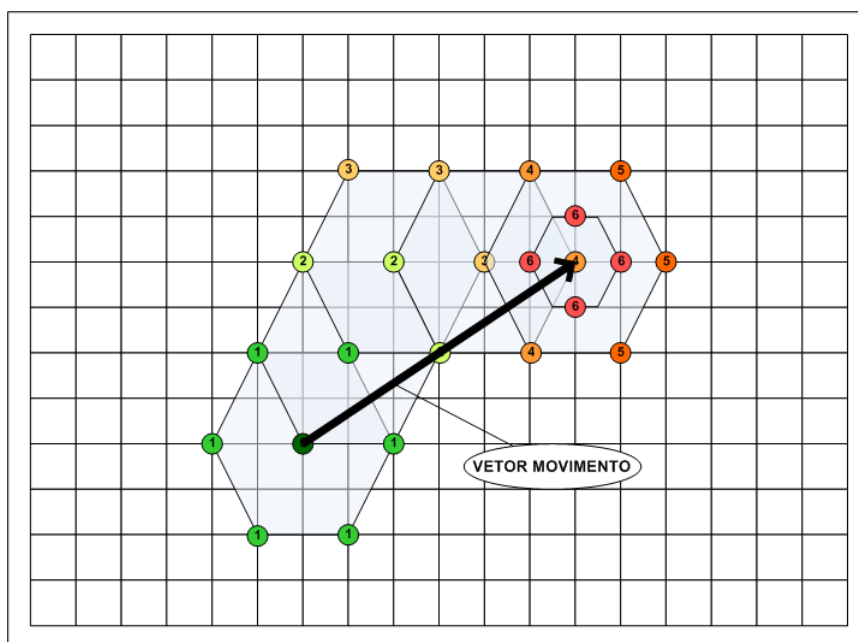


Figura 3.4: Modelo de busca hexagonal.

Existem variações do padrão hexagonal, que se diferenciam principalmente no ajuste fino das últimas comparações. Dentre estas, está a busca **hexagonal melhorado**, que é ilustrado na figura 3.5:

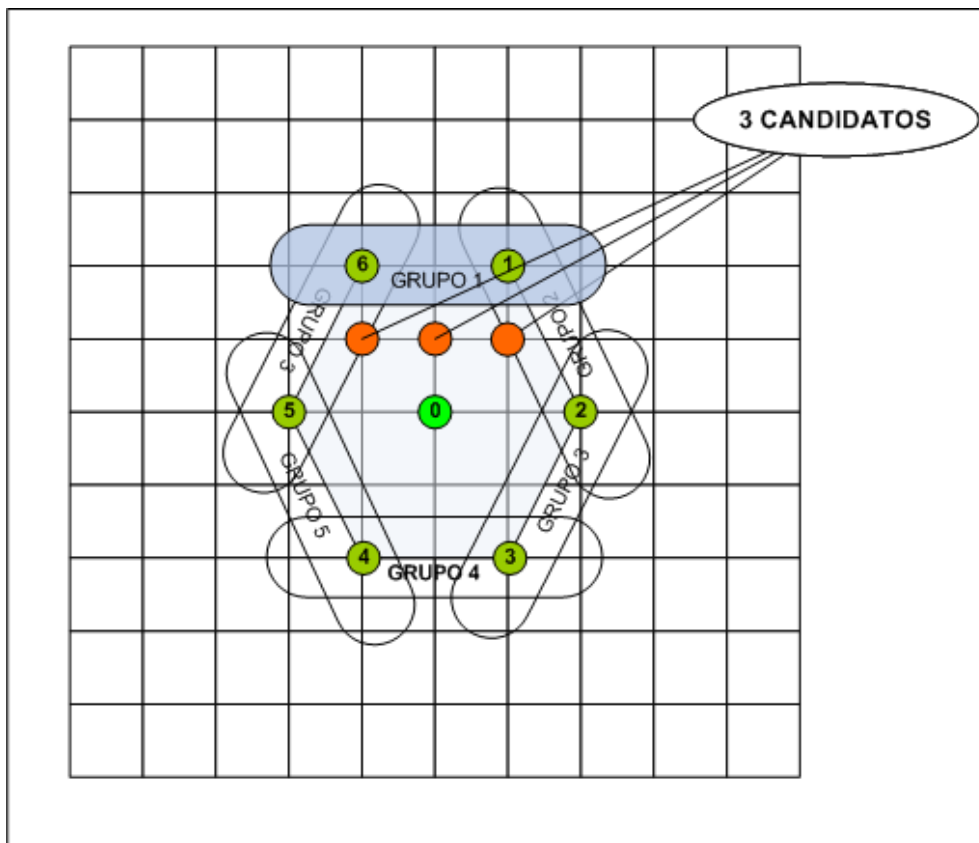


Figura 3.5: Modelo de busca hexagonal melhorado.

Sua melhoria está no ajuste fino, que apenas os macroblocos mais próximos da aresta do último hexágono que possui os vértices de menor soma das diferenças absolutas são comparados, diminuindo as quantidade de comparações, aumentando a eficiência do algoritmo.

3.3.4 Busca circular

O modelo que aparentemente faz o menor número de buscas, é o circular. Este padrão de busca faz uma varredura em zonas circulares com o centro na posição original do macrobloco em questão, como o ilustrado na figura 3.6:

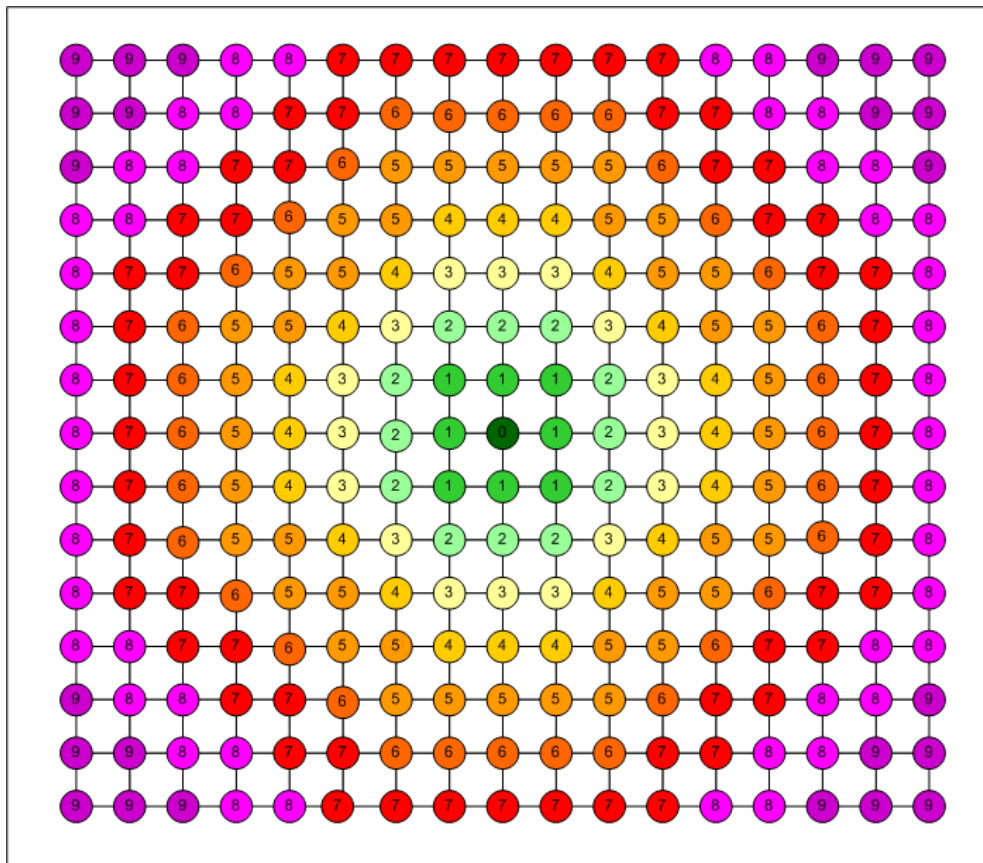


Figura 3.6: Modelo de busca circular.

A busca começa pelo macrobloco central e continua pelas circunferências adjacentes até que um macrobloco comparado tenha uma SAD menor do que as SADs dos macroblocos da circunferência imediatamente externa. Assim o vetor movimento é definido. Exemplificando, se um macrobloco da região 4 tem uma SAD menor do que os localizados na circunferência 5, a busca é encerrada e o vetor está definido. Caso a SAD da região 5 for menor do que os candidatos da região 4, a busca continua na circunferência 6.

Este padrão é muito bom para imagens que tenham pouco movimento e movimentos lentos, pois a busca começa a ficar extensiva quando os candidatos se encontram em camadas mais externas. Por exemplo: se o macrobloco deslocado encontra-se na circunferência oito, são necessárias mais de trezentas comparações, pois além de comparar os candidatos das oito circunferências menores, o algoritmo fará a comparação dos macroblocos da região nove para concluir que o melhor candidato está na circunferência anterior. Então o número de comparações cresce muito rapidamente com o aumento do raio da circunferência.

Portanto, quando não se tem certeza do tipo de imagem a se estimar o movimento, utilizar este padrão de busca não é conveniente para determinar o vetor.

3.4 IMPLEMENTAÇÃO

3.4.1 Detecção de Movimento

Os algoritmos de estimação de movimento utilizados são inspirados nos desenvolvidos pelos autores do mesmo projeto desenvolvido para o *Microsoft Windows*, JESSÉ BERTO ANDRADE e RAFAEL GALVÃO DE OLIVEIRA, que cederam gentilmente seus códigos para serem implementados no ambiente Linux.

Do arquivo `DetectarMovimento.cpp` foram retirados os códigos para controle de *mouse* e o programa foi compilado e funcionou sem problemas.

3.4.2 Makefile

Para facilitar a compilação e a instalação foi criado o arquivo `Makefile` que compila os arquivos com o programa `g++`.

Ao digitar o comando `$ make` no terminal do Linux, na pasta onde estão os arquivos fontes: `main.cpp`; `DetectarMovimento.cpp`; `CapturaImagens.cpp`; `regiao.cpp`, são criados todos os objetos (arquivos `.o`) dos arquivos são ligados entre si. E para excluir os arquivos executáveis e os objetos de compilações anteriores, basta digitar o comando `$ make clean`.

A estrutura deste arquivo pode ser vista abaixo:

```
SOURCE = main.cpp DetectarMovimento.cpp CapturaImagens.cpp regiao.cpp
OBJECTS = ${SOURCE:.cpp=.o}
```

```
.PREFIXES = .cpp .o
CC = g++
ARCH = "X86"
CFLAGS = -I. -O4
```

```
.cpp.o:
$CC -c ${CFLAGS} $<
```

```
all: main
```

```
echo "Ready!"
```

```
main: main.o DetectarMovimento.o CapturaImagens.o regio.o  
${CC} -o $ $.o DetectarMovimento.o CapturaImagens.o regio.o
```

```
clean:  
-${RM} main *.o
```


4 CONCLUSÕES

Ao final da elaboração deste projeto, o autor considera que obteve sucesso no principal objetivo do inicialmente traçado: Adquirir experiência com o sistema operacional LINUX e aprender a programar neste ambiente.

Antes de iniciar o projeto, o autor já tinha utilizado o sistema operacional Linux, mas conhecia poucos recursos dele e não conseguia instalar novos programas. Na metade do projeto, o autor participou do CURSO DE EXTENSÃO EM PROGRAMAÇÃO LINUX E DESENVOLVIMENTO DE SISTEMAS EMBARCADOS, oferecido pelo Grupo de Processamento Digital de Sinais - GPDS da Universidade de Brasília. O que ajudou muito a realização do projeto. Ao final do projeto, o autor já não possui grandes dificuldades ao operar o sistema operacional, sendo capaz de executar seus principais comandos, instalar e atualizar seus programas e principalmente o autor consegue manipular os arquivos de dispositivos do Linux, como a *webcam* e o *mouse*.

Outro recurso que o autor conheceu foi o editor de texto \LaTeX e sua interface gráfica do Linux KILE. Apesar de sua estrutura ser bem diferente do Microsoft Word, o autor logo se familiarizou com o novo editor de texto, obtendo sucesso escrevendo a monografia final no formato `.tex`.

Em relação ao segundo objetivo do projeto, adquirir noções de processamento de imagens, o autor conseguiu resultados satisfatórios ao compreender e portar um programa escrito para o sistema operacional da *Microsoft*. Anteriormente ao projeto, não tinha conhecimentos sobre o assunto e agora já sabe como comparar os macroblocos e decidir qual é o melhor candidato do próximo quadro para criar o vetor movimento. Foram vistos a comparação por correlação e soma das diferenças absolutas. Técnicas eficientes foram implementadas para obter o melhor casamento de macroblocos rapidamente. Foi estudado métodos como o hexagonal, hexagonal melhorado, circular e diamante. Concluiu que o melhor custo de processamento era a busca diamante e implementando-a.

O programa desenvolvido manipula os arquivos de dispositivos, seleciona a área na qual deve-se fazer a estimação de movimento, possui um parâmetro de tolerância para movimento do ponteiro, para que pequenos movimentos não altere sua posição. Mas apesar disso, o ponteiro do *mouse* sempre se movimenta um pouco nos testes feitos. Estes pequenos movimentos podem ser atribuídos aos ruídos presentes nas imagens provenientes da *webcam*.

Durante todo o processo de desenvolvimento o autor teve problemas de conflito com a placa de vídeo de seu *notebook* e o OPENGL, uma ferramenta para apresentar a imagem da *webcam*. Apesar do programa de captura de imagem funcionar corretamente, e o movimento era estimado satisfatoriamente, imagem não era formada corretamente por seu programa no computador portátil, onde o projeto era desenvolvido. A figura 4.1 apresenta a imagem mostrada na tela de seu *notebook*. Mas o mesmo programa não apresentava erro no outro computador do GPDS. O autor dedicou bastante tempo do seu projeto para tentar resolver o problema, mas não obteve sucesso.



Figura 4.1: Imagem apresentada com o conflito do OPENGL

Devido a este contratempo, o autor não concluiu a *interface* gráfica do projeto. A imagem era capturada corretamente pelo dispositivo, mas não era visualizada corretamente, e também não conseguiu apresentar os vetores de movimento. Apenas o esqueleto da interface foi elaborado, conforme a figura 4.2:

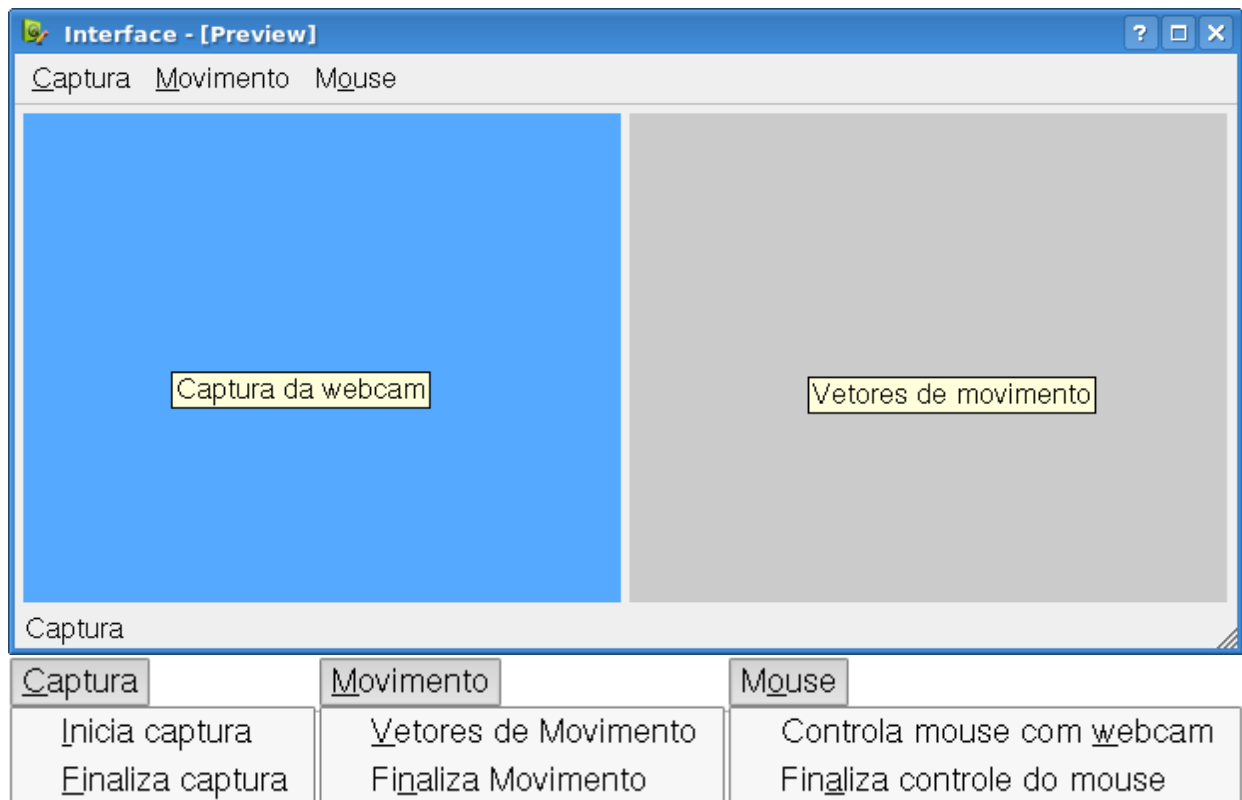


Figura 4.2: Interface do projeto

O projeto pode ganhar mais funcionalidades futuramente. Como por exemplo uma *interface* completa, onde o usuário poderá definir a área da imagem que deseja fazer a estimação de movimento, e escolher parâmetros como a tolerância para movimentar o *mouse*. Outra funcionalidade interessante para a *interface* é alterar os parâmetros a imagem, como o brilho, contraste, correção gamma, saturação entre outros.

Em relação ao controle do *mouse* o programa pode receber melhorias, para possibilitar que o usuário consiga simular o *click* de seus botões esquerdos e direitos, utilizando o reconhecimento de objetos como a mão fazendo o movimento de pressionar os botões. O programa pode também ser utilizado para escrever no arquivo do teclado, permitindo que o usuário simule algumas teclas, movimentando sua mão em determinada região e num certo sentido.

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] ANDRADE, JESSÉ BERTO; OLIVEIRA, RAFAEL GALVÃO DE. *Implementação de Interface Visual Baseada em Estimação de Movimentos e Posição para Controle de Mouse e Teclado*. (Projeto de graduação), Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 2005.
- [2] XHAARD, MICHEL *SpcaView & SpcaTools Howto* - <http://mxhaard.free.fr/sview.html> Ecole Centrale d'Electronique ECE, Université Pierre et Marie Curie, Paris, France, 2004.
- [3] PIEKARSKI, WAYNE. *Tutorial Notes - Hacking Your Own Virtual and Augmented Reality Apps for Fun and Profit*. School of Computer and Information Science, The University of South Australia, Adelaide, Australia, 2004.
- [4] LEE, SAU DAN. <http://www.informatik.uni-freiburg.de/danlee/fun/psaux/> Institut für Informatik, Alber-Ludwig-Universität Freiburg, Deutschland, 2004.
- [5] VASSILIADIS, S.; HAKKENNES, E.A.; WONG, J.S.S.M.; PECHANNEK, G.G. *The Sum-Absolute-Difference Motion Estimation Accelerator*. Electrical Engineering Department, Delft University of Technology, The Netherlands, 2004.
- [6] WINISCHHOFER, THOMAS. *SiS/XGI graphics chipsets and X.org/XFree86/Linux*, Australia, 2004.
- [7] *Enciclopédia livre Wikipedia* - <http://pt.wikipedia.org>

