ESCOLA POLITÉCNICA - UNIVERSIDADE DE SÃO PAULO

Departamento de Engenharia de Sistemas Eletrônicos

Avenida Professor Luciano Gualberto, travessa 3, nº158 CEP: 05508-900 São Paulo SP
Telefone: (0xx-11) 3091.5728 Fax (0xx-11) 3091.5585

RELATÓRIO PSI-2594

INICIAL ()
INTERMEDIÁRIO ( )
FINAL (X)

DATA DE ENTREGA: ___/___/2007

NOME DO ALUNO: Erich Renato Santos Shiino
No USP: 3726610
LOCAL DO PROJETO: Escola Politécnica da USP
NOME DO ORIENTADOR: Prof. Dr. Hae Yong Kim

TÍTULO DO PLANO DE TRABALHO: Ferramenta de realidade aumentada baseada em conjunto projetor-câmera para auxílio ao ensino

OBJETIVO DO PLANO DE TRABALHO:

Desenvolver uma ferramenta que possibilite a interação de um usuário com seu computador, através da imagem gerada por um projetor. Para isto, utiliza-se uma câmera que identifica as movimentações de um ponteiro (apontador laser ou varinha telescópica), convertendo-as em ações de um mouse num aplicativo em uso no computador.

São Paulo – SP

Erich Renato Santos Shiino

FERRAMENTA DE REALIDADE AUMENTADA BASEADA EM CONJUNTO PROJETOR-CÂMERA PARA AUXÍLIO AO ENSINO

São Paulo
2007

Erich Renato Santos Shiino

FERRAMENTA DE REALIDADE AUMENTADA BASEADA EM CONJUNTO PROJETOR-CÂMERA PARA AUXÍLIO AO ENSINO

Projeto de formatura apresentado à Escola Politécnica da Universidade de São Paulo para a conclusão do curso de Engenharia.

Área de Concentração: Engenharia Elétrica

Orientador: Prof. Dr. Hae Yong Kim

v.1

São Paulo
2007

Erich Renato Santos Shiino

FERRAMENTA DE REALIDADE AUMENTADA BASEADA EM CONJUNTO PROJETOR-CÂMERA PARA AUXÍLIO AO ENSINO

Projeto de formatura apresentado à Escola Politécnica da Universidade de São Paulo para a conclusão do curso de Engenharia.

Área de Concentração: Engenharia Elétrica

Orientador: Prof. Dr. Hae Yong Kim

v.1

São Paulo
2007

RESUMO

Este relatório descreve a ferramenta de realidade aumentada baseada em um conjunto projetor-câmera, que permite que o usuário interaja com uma aplicação qualquer em um computador através de uma imagem projetada. O princípio de funcionamento baseia-se no reconhecimento de ponteiros que atuam sobre a projeção. A implementação será realizada em C++ com o uso da biblioteca OpenCV. Encontram-se neste trabalho, além das especificações técnicas, a análise de projetos relacionados, problemas encontrados e resultados obtidos.

.

Palavras-chave: Realidade aumentada, visão computacional, conjunto projetor-câmera, OpenCV.

SUMÁRIO

1 INTRODUÇÃO

O uso de recursos computacionais difundiu-se por todos os setores da sociedade. As instituições de ensino, além de serem o berço para a pesquisa de novas tecnologias, são as que mais beneficiam com os novos adventos. Entretanto, ainda existe uma subutilização da tecnologia: os computadores não auxiliam na didática de aulas e palestras o tanto quanto poderiam. Neste contexto, a Realidade Aumentada tem um importante papel no enriquecimento da interação homem-máquina.

A realidade aumentada proporciona ao usuário uma interação segura e agradável. Elimina em grande parte a necessidade de treinamento, pois traz para o ambiente real os elementos virtuais, enriquecendo e ampliando a visão do usuário sobre o mundo real. Para que isso se torne possível, é necessário combinar técnicas de visão computacional, computação gráfica e realidade virtual, gerando como resultado a correta sobreposição de objetos virtuais no ambiente real (Azuma 1993).

“A visão computacional lida com o processamento de dados de uma imagem para uso de um computador” (UMBAUGH, 1999).Assim, propõe-se uma ferramenta auxiliadora a processos didáticos, permitindo ao usuário interagir diretamente com a imagem, que esta sendo apresentada a platéia, por meio de um projetor. Seria como se essa projeção fosse uma espécie de tela sensível ao toque.

Dessa forma, caracteriza-se a realidade aumentada como o uso da visão computacional com a captura de uma imagem por uma câmera, processamento por um computador e conversão de um evento de interação comum, como por exemplo, o uso de um mouse. Assim, embora pessoas estejam envolvidas no desenvolvimento do sistema, a aplicação final necessita de um computador para usar a informação visual diretamente sendo, portanto, de acordo com a definição de visão computacional de (UMBAUGH, 1999).

2 OBJETIVOS

Desenvolvimento de uma ferramenta que permite um usuário interagir com uma aplicação qualquer em um computador através da imagem gerada por um projetor. A imagem, proveniente de uma câmera, é capturada e identifica-se a movimentação e eventos gerados por um ponteiro (apontador *laser* ou varinha telescópica). Em seguida, estas ações são convertidas em eventos no sistema operacional do computador em uso. Dessa forma, esta ferramenta poderá ser de grande utilidade em processos educacionais.

Entre os eventos de sistema operacional gerados estão: movimentos e eventos de clique, de clique duplo, e de arrastar-e-soltar, equivalentes aos de um mouse de um microcomputador; eventos de teclado e comandos como executar programa, abrir arquivos, etc.

Os ponteiros reconhecidos são apontador *laser* ou varinha telescópica. A varinha possui LEDs (*Light Emitting Diode*) coloridos para permitir que sejam identificados pela ferramenta. Outra aplicação possível desta ferramenta é possibilidade de interação com um monitores LCD (por exemplo, com notebooks) em substituição a mouses convencionais, bastando para isto a substituição da imagem projetada pelo próprio monitor.

A ferramenta será desenvolvida para o sistema operacional Windows, pois ainda é a plataforma mais difundida entre os usuários-alvo.

Como metas de desempenho estabelece-se uma taxa mínima de amostragem da imagem de cinco quadros por segundo. O erro de mapeamento das coordenadas da imagem projetada em coordenadas do sistema operacional deve ser no máximo 30 pixels. As resoluções da câmera e do projetor influenciam na precisão deste mapeamento. Portanto, recomenda-se a utilização de equipamentos que tenham resolução no mínimo superior à utilizada pelo sistema operacional.

.

3 JUSTIFICATIVA

Nos primórdios do desenvolvimento científico, os grandes avanços tecnológicos ficavam restritos as camadas mais ricas da população mundial. Contudo, o progresso trazido pelas inovações está provocando o seu barateamento, tornando mais acessíveis computadores e outros equipamentos. Diante disso, as instituições de ensino observaram uma oportunidade de desenvolver novas metodologias de educação aliadas a tecnologia.

O Brasil apresenta um número cada vez maior de instituições – desde escolas primárias até universidades – que utilizam computadores e equipamentos eletrônicos durante as aulas. Isso facilita a absorção do conteúdo pelos alunos e torna as aulas mais agradáveis e eficientes.

Há alguns anos, iniciou-se o uso de retroprojetores os quais possuem uma vantagem em relação à tradicional lousa: apresentam uma grande quantidade de dados em gráficos e diagramas que são mais inteligíveis. Recentemente, observa-se o uso de computadores em apresentações por meio de projetores *(data-show*). Porém, estas aplicações limitam-se à passagem de uma série de quadros (*slides*) aos alunos e não aproveitam o verdadeiro potencial da tecnologia, equiparando-se a sua antecessora em relação a interatividade.

Aplicações interativas podem enriquecer o processo de aprendizagem. A utilização da ferramenta de realidade aumentada, permite a interação do usuário diretamente com a projeção proveniente do computador, possibilitando ao educador uma maior dinâmica na apresentação do conteúdo. durantes aulas, palestras, etc. Por exemplo, poderia conter um programa de simulação, exibindo diagramas e gráficos numa ordem aleatória , e diferentes informações poderiam ser visualizadas com muita facilidade. Por outro lado, numa apresentação convencional fica-se preso à linearidade de uma sequência de slides.

Palestras, seminários e aulas podem ser enriquecidas com a ferramenta de realidade aumentada. Um palestrante pode se utilizar de mais recursos computacionais sem ter de ficar preso a uma mesa , ficando mais livre para discursar.

Além do uso direto da ferramenta, a pesquisa realizada para o desenvolvimento, pode servir de base para outros projetos, como jogos e ambientes de realidade aumentada mais sofisticados. Comercialmente ainda, é possível o desenvolvimento de um equipamento integrado, onde um projetor já tenha uma câmera embutida que passa eventos (como os de um *mouse*) automaticamente para o sistema operacional através de uma interface USB ou mesmo Bluetooth.

Um produto com funcionalidades semelhantes, já no mercado, é a lousa interativa.Ela possui uma superfície sensível ao toque e permite a criação de anotações em uma série de aplicativos próprios, além de ter um modo projetado que permite a interação com qualquer aplicativo de um computador (com auxílio da superfície sensível ao toque e a imagem de um projetor)[1].Entretanto, o custo deste tipo de equipamento é extremamente alto (pode chegar a quase R$ 11.000)[2], sendo inviável para escolas públicas ou empresas de pequeno porte.

Por outro lado, a tecnologia envolvida neste projeto visa ao aproveitamento de equipamentos já disponíveis: apenas um computador ligado a um projetor e uma câmera. Além disso, esse conjunto de equipamentos é muito mais móvel que uma lousa interativa, sendo portanto uma alternativa econômica e muito mais versátil.

---

1 Disponível em http://www.scheiner.com.br/produtos_sb.php?id=66&op=caracteristicas. Acesso em 10 de abril de 2007

2 Disponível em http://www.videoacoustic.com/?modulo=produto&acao=flypage&id_produto=879. Acesso em 10 de abril de 2007

4 MATERIAIS E MÉTODOS

Para a especificação desta ferramenta de realidade aumentada, inicialmente, realizou-se uma pesquisa sobre produtos e projetos já existentes que realizassem ao menos parcialmente alguma operação semelhante ao estudo. Em seguida, definou-se o ambiente de desenvolvimento e os equipamentos e materiais necessários. O projeto foi descrito como uma série de módulos inter-dependentes, que são apresentados nas seções 4.4.

4.1 Análise de projetos e ferramentas relacionados

Nesta etapa, uma série de ferramentas, bibliotecas e produtos foram encontrados. Em seguida, os mais relevantes para este projeto são apresentados.

4.1.1 Lousa interativa

Consiste num quadro sensível ao toque, que associado a projetores e ao computador, permite ao usuário interagir com um software usando as mãos ou uma caneta especial [SCHNEIDER]. A funcionalidade deste quadro é muito semelhante à pretendida com este projeto. Entretanto, o custo deste equipamento é muito alto, sendo inviável para escolas e universidades. Outra desvantagem deste sistema é que a lousa é um equipamento muito grande e sensível, o que dificulta sua mobilidade.

4.1.2 Magic Mouse

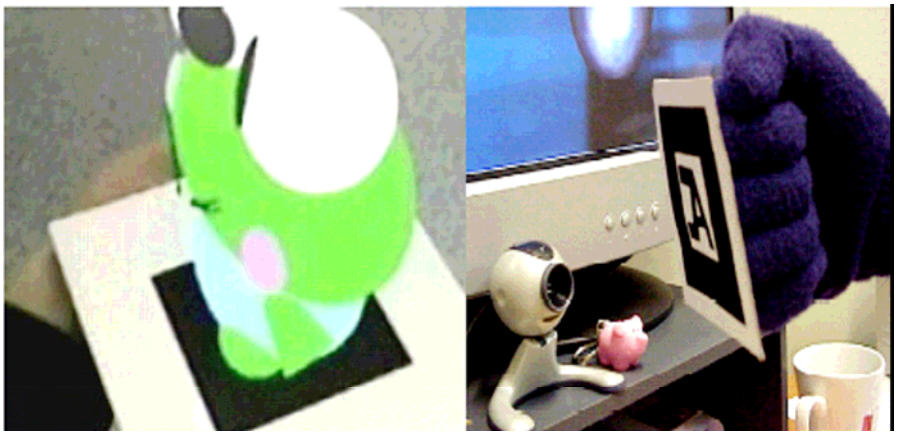

Figura 1 Sobreposição de imagem em marcador e luva do MagicMouse

É a implementação de um dispositivo de entrada para computadores com funcionamento equivalente ao de um mouse comum. Permite a operação em ambiente 2D ou 3D pela movimentação e rotação do pulso. Este dispositivo consiste em uma luva que possui uma placa quadrada com um marcador pré-determinado. Um software utiliza as bibliotecas do ARToolKit para extrair da imagem de uma câmera uma vista deste marcador. A partir da posição deste marcador são extraídos os parâmetros de posição e rotação em relação aos eixos do eixo X, Y e Z. Este dispositivo tem a desvantagem de estar acoplado na luva, o que tira a liberdade do usuário. Além disso, o uso prolongado provoca desconforto devido à posição quase fixa, na qual o pulso deve ser mantido [WOODS].

4.1.3 ARHockey

Trata-se de uma versão do jogo conhecido como *air hockey* em realidade aumentada baseada em projetores [MIRANDA].

Algumas versões deste jogo utilizam um HMD (*head mounted display*) que possui uma lente semi-opaca, a qual serve para a projeção dos elementos sintéticos aumentadores e para visão direta da realidade. Contudo, o uso desta tecnologia limita a visão do jogo apenas aos jogadores.

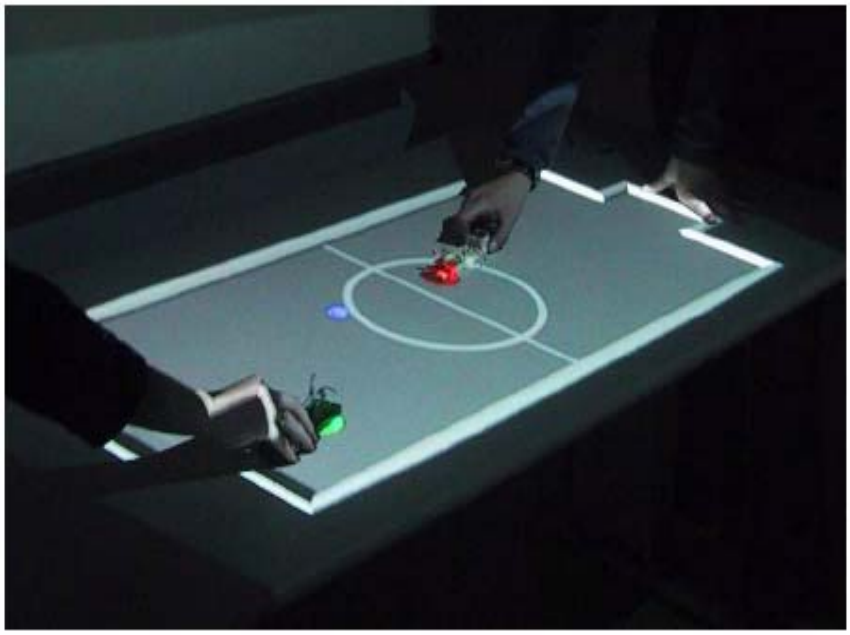

Figura 2 ARHockey em funcionamento [MIRANDA et al]

Ao contrário destas versões, o ARHockey pode ser visualizados por outras pessoas além dos jogadores, pois utiliza projetores ao invés do HMD. Desenvolveu-se utilizando a biblioteca OpenCV. O software acompanha a posição de marcadores com LEDs infravermelhos acoplados e traduz a posição dos mesmos na posição dos batedores do jogo. A diferença fundamental do ARHockey para este projeto é o tratamento dado a estes marcadores. No jogo, utiliza-se um filtro plástico que não permite a passagem de luz no espectro visível, tornando o rastreamento muito mais simples. O artigo do ARHockey não descreve como a transformação do sistema de coordenadas é realizada.

4.1.4 OpenCV

Open Source Computer Vision Library é uma biblioteca desenvolvida pela Intel com funções de programação e principalmente direcionadas à visão computacional em tempo real. Esta é uma das bibliotecas mais utilizadas para reconhecimento de objetos, segmentação de imagens, rastreamento de movimentos, etc. Possui grande quantidade de artigos e documentos disponíveis na

internet. Além disso, um grupo de discussão[3], destinado ao compartilhamento de informações sobre a biblioteca, trocou mais de 500 mensagens mensais nos últimos quatro anos. O OpenCV contribui especificamente para este projeto com rotinas para captura de imagens, calibração de câmera, processamento de imagens e operações em 3D.

4.2 Ambiente de desenvolvimento

- Sistema operacional: Windows XP. Esta plataforma foi escolhida, pois a maior parcela dos usuários em potencial desta ferramenta a utilizarem.
- Interface de Desenvolvimento: Dev-C++ versão 4.9.9.2[4]
- Biblioteca C++ para processamento de imagens Intel OpenCV[5] versão Beta5a
- Compilador C++ GNU Compiler Collection fornecido no Cygwin[6] que é uma coleção de ferramentas livres, permitem ao Windows operar algumas funções como um sistema Linux (no caso deste projeto, o compilador GCC opera na abstração criada pelo Cygwin).

4.3 Equipamentos e materiais

- Notebook ASUS A6R com processador Celeron M 1.7 GHz e placa aceleradora de vídeo Radeon Xpress 200M. 512 Mb de RAM
- Webcam Logitech QuickCam for Notebook Pro com resolução de 640x480.

[3] Disponível em http://tech.groups.yahoo.com/group/OpenCV/
[4] Disponível em www.bloodshed.net
[5] Disponível em http://sourceforge.net/projects/opencvlibrary/
[6] Disponível em www.cygwin.com

Figura 3 Logitech QuickCam for Notebook Pro

- Projetor multimídia
- Monitor LCD com resolução de 1280x800**.** Este monitor pode ser usado em substituição ao projetor num uso particular da ferramenta, no qual o usuário pode substituir o mouse por um apontador que pode ser ou uma varinha ou caneta especial.

Figura 4 Notebook ASUS A6R

- Apontador *laser*
- Varinha telescópica

- LED (*Light Emitting Diode*) coloridos

4.4 Modularização

O projeto é desenvolvido com base em metodologias de engenharia de software de forma a permitir uma clara análise do código desenvolvido e uma fácil identificação de erros, possibilitando futuros estudos, melhorias e correções.

Assim, visando uma codificação mais clara e a facilitação de possíveis desenvolvimentos futuros, a ferramenta é dividida em uma série de módulos de acordo com sua funcionalidade de forma que o acesso de cada um deles seja feito apenas por meio de funções. Além disso, a divisão em módulos permite a fácil substituição de algoritmos sem prejudicar o funcionamento da ferramenta, que pode continuar a utilizar as mesmas chamadas de funções. Os relacionamentos entre os módulos da ferramenta são apresentados no diagrama a seguir:

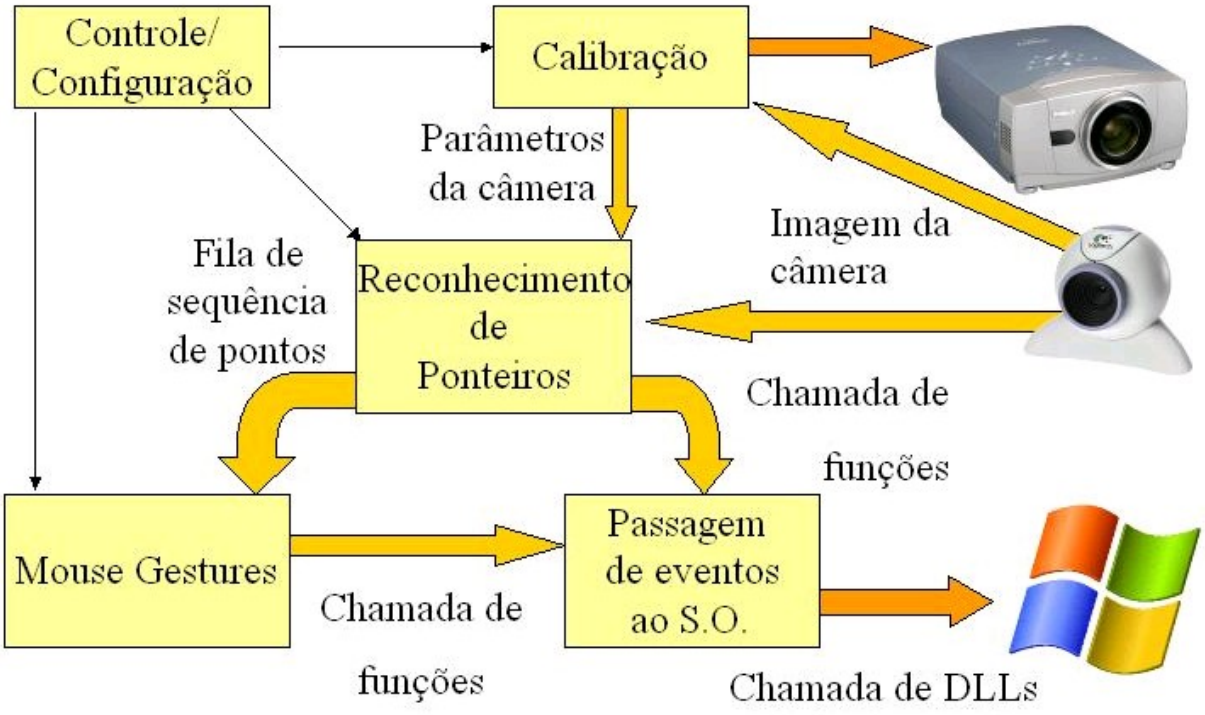

Figura 5 Diagrama de blocos da ferramenta

Para facilitar o desenvolvimento da ferramenta, o protótipo foi implementado e testado em um conjunto câmera-notebook. Nesta montagem, a tela

de projeção e o projetor são substituidos pela tela LCD de um notebook, sendo este um dos modos de uso previsto para a ferramenta. A câmera é posicionada de forma a capturar todo o monitor LCD. Uma das possíveis posições é mostrada a seguir, onde a câmera pode ser vista ao lado direito do notebook:

Figura 6 Montagem da ferramenta com um conjunto câmera-notebook

Esta configuração facilita a implementação e testes por eliminar a necessida de um auditório para o uso da ferramenta e ao mesmo tempo garantir condições de uso semelhantes, desta forma, justificando a simplificação. O conjunto câmera-notebook é semelhante ao câmera-projetor com excessão de uma redução do erro relacionado a distorções da imagem provenientes do projetor e do plano de projeção. Por outro lado, o uso de um monitor LCD apresenta uma dificuldade adicional para o reconhecimento de imagens devido à variação de tonalidade dos pixels de acordo com o ângulo de visão do observador. Como a câmera é posicionada próxima ao monitor, o ângulo com que esta observa os pixels varia muito ao longo de toda a tela. Este efeito faz com que uma única cor seja observada de forma muito diferente ao longo da imagem. Para exemplificar este efeito, apresentou-se uma imagem completamente vermelha (R=255, G=000, B=000) no monitor e capturou-se com a câmera. O resultado é apresentado a seguir:

Figura 7 Projeção de vermelho utilizada para geração de máscara

Para o mesmo vermelho apresentado em toda a tela, a câmera captura a cor R=122, G=57, B=63 no canto superior esquerdo e a cor R=255, G=164, B=30 no canto inferior direito. Este pequeno exemplo mostra como o ângulo de visão pode afetar o reconhecimento de imagens.

4.4.1 Módulo de configuração

Este módulo consiste em uma interface gráfica, na qual o usuário pode chamar as rotinas de inicialização da ferramenta - que consistem na identificação da área da imagem da câmera correspondente à projeção da imagem do computador - e as rotinas de calibração de câmera. Na mesma interface, o usuário deve: definir o tipo de ponteiro utilizado; escolher os eventos reconhecidos; e selecionar a *webcam* que será a entrada de dados do sistema, caso exista mais de uma instalada no computador. Algumas opções para usuários avançados estarão disponível, como por exemplo, o controle sobre a taxa de atualização máxima utilizada pela ferramenta que interfere no uso do processador.

Este módulo é secundário em comparação ao desenvolvimento necessário para os demais módulos. Portanto, este bloco foi implementado apenas com o mínimo para controlar os estados da ferramenta (descritos posteriormente) e opera com configurações padrão pré-estabelecidas.

O fluxograma a seguir apresenta os estados da ferramenta desde sua inicialização até o início das atividades de operação de reconhecimento de ponteiros e eventos. O módulo de configuração controla a passagem entre os estados e aciona os demais módulos da ferramenta.

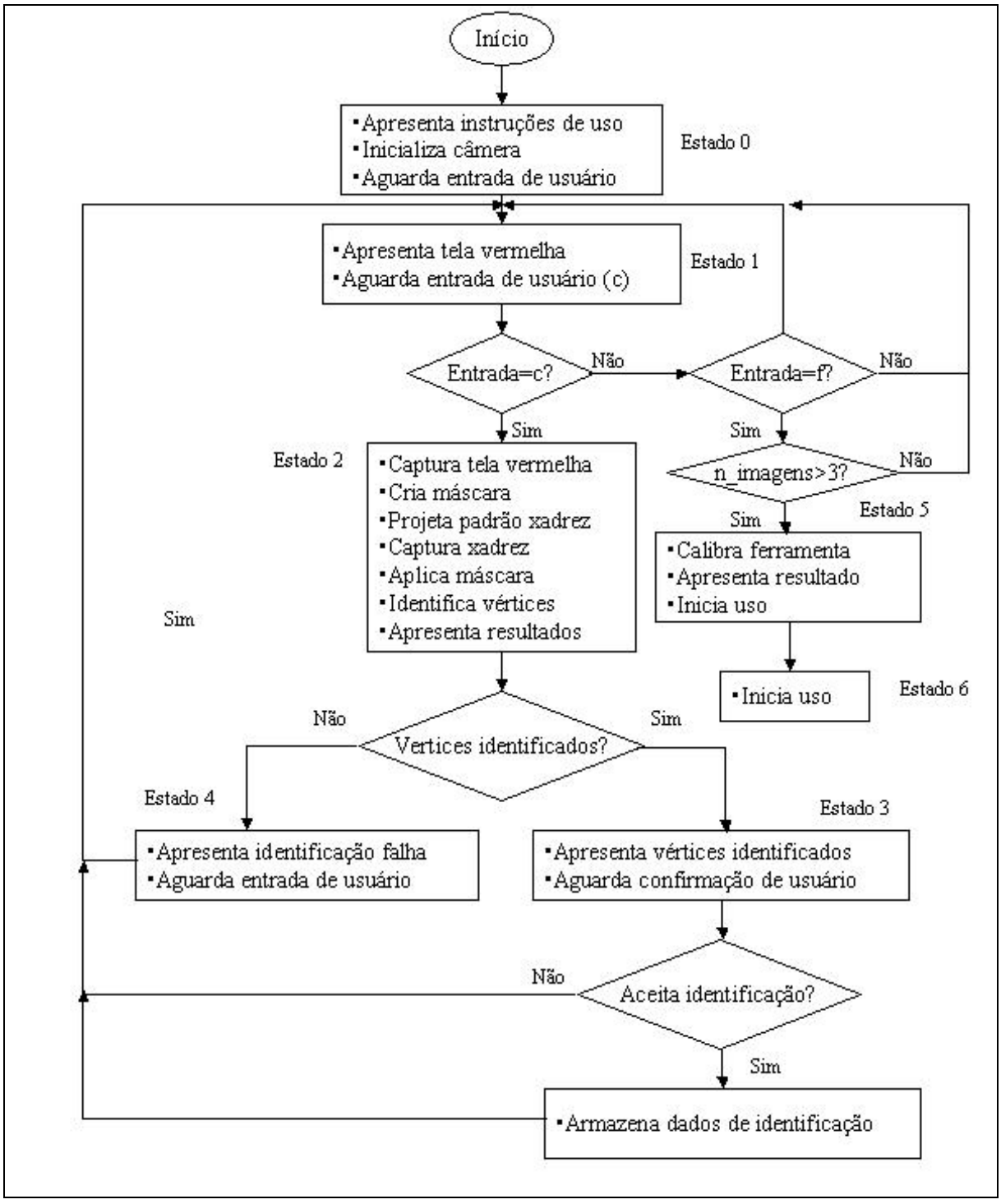

Figura 8 Fluxograma da inicialização e calibração da ferramenta

A ferramenta possui interface com usuário em modo texto, sendo que a passagem de comandos dos usuários é realizada com o simples pressionamento de algumas teclas. A lista de comandos é apresentada na tabela a seguir:

Tabela 1: Lista de comandos da etapa de calibração da ferramenta

Tecla	Estado em que se aplica	Comando realizado
C	Estado 1	Captura imagem para identificação de vértices
A	Estado 3	Aceita identificação de vértices para calibração
D	Estado 3	Descarta identificação dos vértices
F	Estado 1	Finaliza processo de captura e inicia calibração
I	Estado 5	Incia uso da ferramenta
<ESC>	Qualquer	Termina a ferramenta
Qualquer	Estado 4	Retorna a modo de captura após falha de identificação de vértices

4.4.2 Rotinas de calibração/Reconhecimento da área da tela e *Undistort*

Estas rotinas são executas antes que a ferramenta possa ser realmente utilizada, pois é necessário encontrar a relação entre as coordenadas da imagem obtida pela câmera e as coordenadas da saída de vídeo do sistema operacional. Este módulo utiliza diversas funções disponíveis na biblioteca OpenCV (*Open Computer Vision Library*).

A *webcam* é alocada e inicializada com a função `cvCaptureFromCam (int index)`, onde `index` é um valor inteiro que indica a câmera que se deseja utilizar. Entretanto, observou-se que a simples inicialização da câmera para posterior captura de apenas alguns quadros para o processo de calibração não funciona adequadamente. O resultado é apenas uma série de quadros cinzas. Isto ocorre porque o ajuste automático de abertura da câmera não é realizado até que ocorra a captura de uma série de quadros. Para resolver este problema, após a inicialização da câmera, cerca de 20 quadros são capturados e descartados para permitir que a câmera se auto-ajuste. De forma semelhante, a câmera captura alguns quadros

enquanto aguarda os comandos do usuário para garantir a abertura adequada no momento da aquisição.

O processo principal consiste na projeção de figuras pré-definidas de tabuleiros de xadrez com número de linhas e colunas conhecido seguido de uma análise da posição do plano de projeção em relação à câmera. De acordo com [VEZHNEVETS], esta imagem deve ter as bordas brancas e número de quadrados deve ser par em uma das dimensões e impar em outra. A Figura 9 é um exemplo de padrão para calibração.

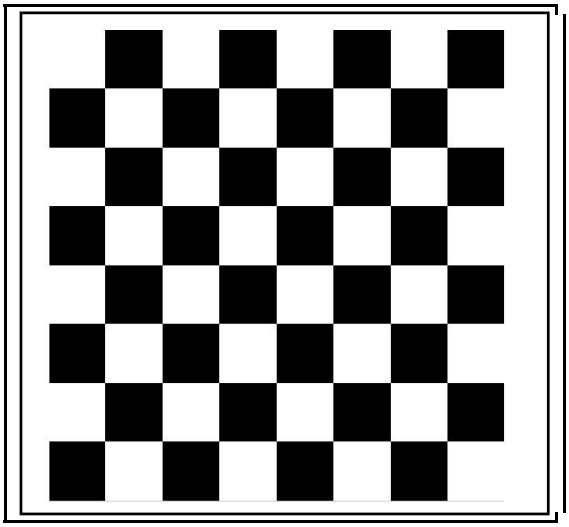

Figura 9 Exemplo de padrão para calibração

Com o uso do OpenCV, o uso de janelas é grandemente facilitado, sendo que o processo de criação é realizado com a instrução *CvNamedWindow(const char* name, int flags),* onde char* é um ponteiro para uma *String* e *flags* é um valor inteiro que indica se a janela é redimensionável (valor 1) ou não (valor 0). Após a criação, as janelas são identificadas e manipuladas usando a *String* como valor de referência para as mesmas.

Para facilitar a manipulação de janelas e permitir uma fácil projeção do padrão xadrez foi desenvolvida uma função com a seguinte estrutura:

*void projetaXadrez(int n, int m, int border, char *pStrJanela*

Nesta função, os parâmetros *n* e *m* são respectivamente o número de colunas e linhas no tabuleiro, *border* é a largura em pixels da borda branca que deve ser deixada nas bordas da imagem e *pStrJanela* é o ponteiro para a String que referencia a janela onde deve ser apresentado o padrão. A função adquire as

dimensões da projeção (ou monitor LCD) em pixels no sistema operacional para criar a imagem com o padrão xadrez e apresentá-los na janela indicada. Quanto maior o número de linhas e colunas existentes na imagem projetada, melhor se torna a calibração da ferramenta devido à distribuição de mais pontos pela tela para o ajuste dos parâmetros intrínsecos e extrínsecos. Entretanto, quanto mais quadrados presentes na projeção, menor eles se tornam e, portanto, mais difícil se torna o processo de identificação. Desta forma, deve-se identificar os maiores valores de *n* e *m* que permitam a identificação adequada de todos os vértices internos dos quadrados.

Após alguns testes, encontrou-se o valor de 19 colunas e 14 linhas e bordas de 30 pixels para a câmera e monitor LCD utilizados. Estes parâmetros criaram retângulos de 64x52. Entretanto, devido à distância, rotação do plano de projeção, efeito de variação de intensidade de acordo com o ângulo de visualização do LCD e baixa resolução da câmera (640x480), os menores retângulos obtidos na captura chegam a ter apenas 17x15 pixels.

A versão mais recente do OpenCV (5 Beta), possui um único tipo de janela implementado, com uma barra de título, botões de controle (minimizar, maximizar e fechar) e borda interna. Entretanto, este tipo de janela não é apropriado para os procedimentos de calibração, pois é necessário criar uma relação entre os pontos criados e os pontos obtidos pela webcam e é difícil obter as dimensões com todos estes elementos adicionais presentes. Devido a este problema foi necessário utilizar funções que vão além do OpenCV. Utilizou-se instruções de mais baixo nível no sistema operacional para obter o manipulador das janelas em uso com o sistema operacional e alterar o estilo das janelas. Assim, conseguiu-se eliminar a barra de título e os botões. Entretanto, não foi possível remover as bordas internas, gerando assim um erro de 2 a 3 pixels devido ao deslocamento das imagens em relação aos limites da tela.

Estudos mostram que, além da imagem projetada, a presença de outros objetos na imagem obtida pela câmera podem comprometer a identificação dos vértices dos quadrados da imagem de calibração [VEZHNEVETS]. Portanto, antes deste processo um simples retângulo colorido é projetado para distinguir a região de projeção do resto do ambiente. A área é identificada a partir da cor e da forma, especificamente, o algoritmo procura um quadrilátero e cria uma máscara. A aplicação desta máscara nos quadros obtidos posteriormente pela câmera permite

que todo o *background* seja eliminado antes do processo de identificação de vértices. Além disso, uma vez identificada a região de projeção é possível eliminar os trechos desnecessários nas laterais para acelerar o processo de calibração.

Os passos da etapa de pré-calibração são ilustrados na **Figura 10**. Da esquerda para a direita tem-se: (a) projeção de retângulo colorido para diferenciação do fundo; (b) identificação do fundo e geração de máscara; (c) projeção de padrão xadrez; (d) aplicação de máscara para eliminação de fundo e redefinição de limites da imagem.

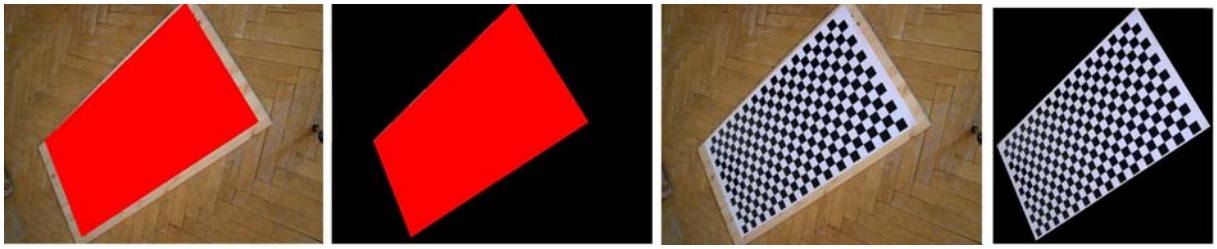

Figura 10 Etapas de pré-calibração

Para a facilitaçao da projeção do retângulo colorido, a seguinte função foi desenvolvida:

```
void projetaCor(int r, int g, int b, char *pStrJanela)
```

Nesta função as componentes R,G e B da cor a ser projetada são passadas respectivamente nas variáveis *r, g, b* e a imagem gerada é então apresentada na janela que é referenciada pela string `pStrJanela`.

Em seguida captura-se a imagem com a câmera e aplica-se uma função para a identificação de retângulos. Em linhas gerais, a função binariza cada um dos canais de cor com diferentes limiares. Depois, aplica-se um algoritmo para a identificação de contornos na imagem. Procura-se ajustar polígonos aos contornos encontrados. Identifica-se um retângulo quando se obtém um polígono de quatro lados, os ângulos de cada vértice são próximos de 90° e a área (número total de pixels contidos no retângulo) é próxima, mas menor que a área total da imagem obtida.

Com o retângulo obtido cria-se uma máscara com uma imagem binária para aplicar-se sobre cada um dos canais de cor das imagens obtidas para eliminar o fundo da imagem.

Experiências demonstraram que algumas vezes devido à grande variação de intensidade dos pixels no monitor LCD, o algoritmo não funciona da forma

esperada e gera retângulos maiores ou menores que a área efetiva de projeção. Por outro lado, apesar de uma série de estudos indicarem que o fundo pode interferir com a identificação dos vértices, este fenômeno não foi observado , portanto, embora o algoritmo de identificação da área de interesse da imagem esteja implementado, este procedimento foi desabilitado. Desta forma diminuiu-se a quantidade de etapas do processo de calibração e aumentou-se a quantidade de imagens capturadas que são efetivamente aproveitadas.

O próximo passo é a identificação de todos os vértices dos quadrados pretos da imagem. A validação deste processo é feita com a comparação dos número de quadrados por linha e coluna com os valores esperados.

A identificação inicial dos vértices é realizada com o uso da seguinte instrução:

```
cvFindChessboardCorners(imageSquares,board_size,image_points_buf, &count, CV_CALIB_CB_ADAPTIVE_THRESH)
```

Esta é uma função do OpenCV na qual os parâmetros foram ajustados para esta aplicação. *ImageSquares* é a imagem na qual se deseja identificar os vértices. *Board*_size é uma variável com duas componentes que indicam o número de linhas e colunas de retângulos presentes na imagem (ou número de linhas e colunas que devem ser procuradas). `Image_points_buf` é o vetor que armazena as coordenadas dos vértices encontrados pela função. Count é um ponteiro para um inteiro que armazena o número de vértices encontrados pelo algoritmo. `CV_CALIB_CB_ADAPTIVE_THRESH` indica que o algoritmo utilizará um limiar variável ao invés de um fixo para binarizar a imagem passada como parâmetro. O limiar utilizado neste processo é calculado como o brilho médio da imagem. Além de identificar os vértices, esta função também precisa ordená-los, na forma de linhas e colunas. Caso todos os vértices sejam identificados e ordenados, esta função retorna um valor diferente de zero. Entretanto, esta função retorna apenas uma aproximação da localização dos vértices. É necessário utilizar utilizar uma segunda função para refinar as coordenadas.

Utiliza-se a seguinte função:

```
cvFindCornerSubPix( imageSquaresgray, image_points_buf, count, cvSize(11,11), cvSize(-1,-1), cvTermCriteria(CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, 30, 0.1))
```

Nesta função `imageSquaregray` é a mesma imagem utilizada para a identificação dos vértices, porém, convertida para a escala de cinzas. `Image_points_buf` é o vetor com as coordenadas dos vértices identificadas com a função `cvFindChessboardCorners` e que conterá as coordenas refinadas por este algoritmo. *Count* é o número de vértices presentes na imagem. `CvSize(11,11)` é passado como metade das medidas da janela de busca para refinamento do vértice, ou seja, a janela de busca estará centrada na coordenada passada como valor inicial e terá dimensões de (11x2+1, 11x2+1) = (23,23). `CvSize(-1,-1)` é passado no lugar do parâmetro *zero_zone* que é a metade das dimensões da região dentro da janela de busca na qual a soma que leva ao refinamento da coordenada não é realizada. O par de valores negativos neste parâmetro indica a soma é realizada em toda a região da janela. O último parâmetro passado na função indica que esta deve terminar o processo iterativo quando uma das condições seja satisfeita: o número de iterações chega a 30 ou o erro na determinação da coordenada é de apenas 0.1.

Uma característica interessante das duas funções mencionadas anteriormente é que, embora as coordenadas sejam usualmente números inteiros por fazerem referência pixels que são elementos discretos, os algoritmos identificam os vértices geram resultados em ponto flutuante. Desta forma é possível levar em consideração o efeito provocando pela amostragem dos sensores digitais da câmera e produzir um resultado que seja mais preciso.

Um dos casos mais simples em que os algoritmos não conseguem identificar os vértices é quando a câmera não visualiza o padrão xadrez por completo ou parte dos retângulos é obstruída por algum objeto estranho. Quando a câmera é posicionada em um local muito distante do monitor LCD os retângulos se tornam muito pequenos e o algoritmo não funciona adequadamente. Este problema pode ser resolvido aproximando-se a câmera ou reduzindo o número de linhas e colunas de retângulos na imagem projetada. Outra situação que leva a uma não identificação do padrão xadrez é a já descrita influência no ângulo de observação na intensidade dos pixels. A figura a seguir apresenta um exemplo de falha de identificação devido à não visualização de todo o padrão projetado. Nestes casos todos os vértices encontrados são representados pela cor vermelha.

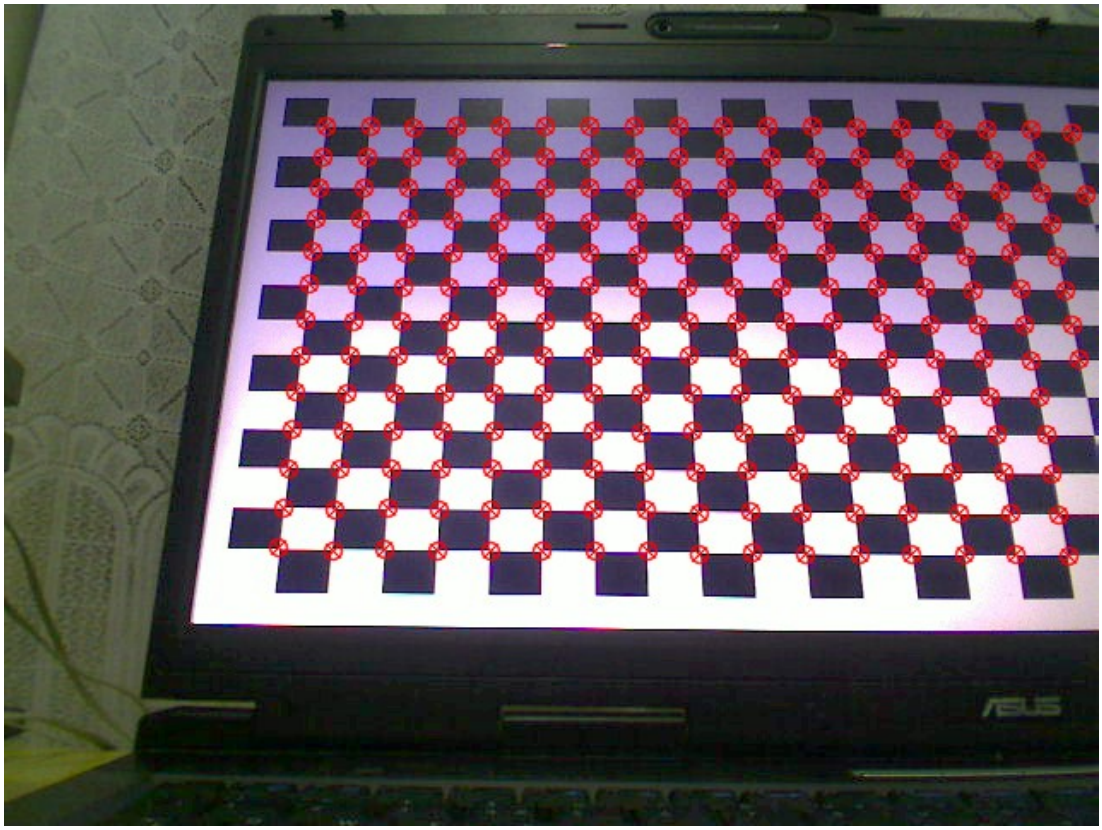

Figura 11 Exemplo de falha de identificação de vértices

Os mesmo motivos descritos anteriormentes podem levar a uma identificação incorreta da posição dos vértices. Assim, apesar da rotina de refinamento de coordenas é possível que algum vértice seja indevidamente identificado o que acarretaria num erro maior na definição dos parâmetros de calibração. Por esta razão a ferramenta apresenta o resultado da tentativa de identificação, para que o usuário corrija a situação que causa a não identificação dos vértices ou para que inspecione visualmente os casos de identificação completa para descartar as imagens nas quais os vértices estejam incorretamente posicionados ou ordenados. Quando se encontra todos os vértices, estes são apresentados com cores diferentes para cada uma das linhas para permitir uma melhor inspeção dos resultados.

As duas figuras a seguir são respectivamente um caso em que os vértices foram posicionados incorretamente e o detalhe do trecho onde o erro foi cometido.

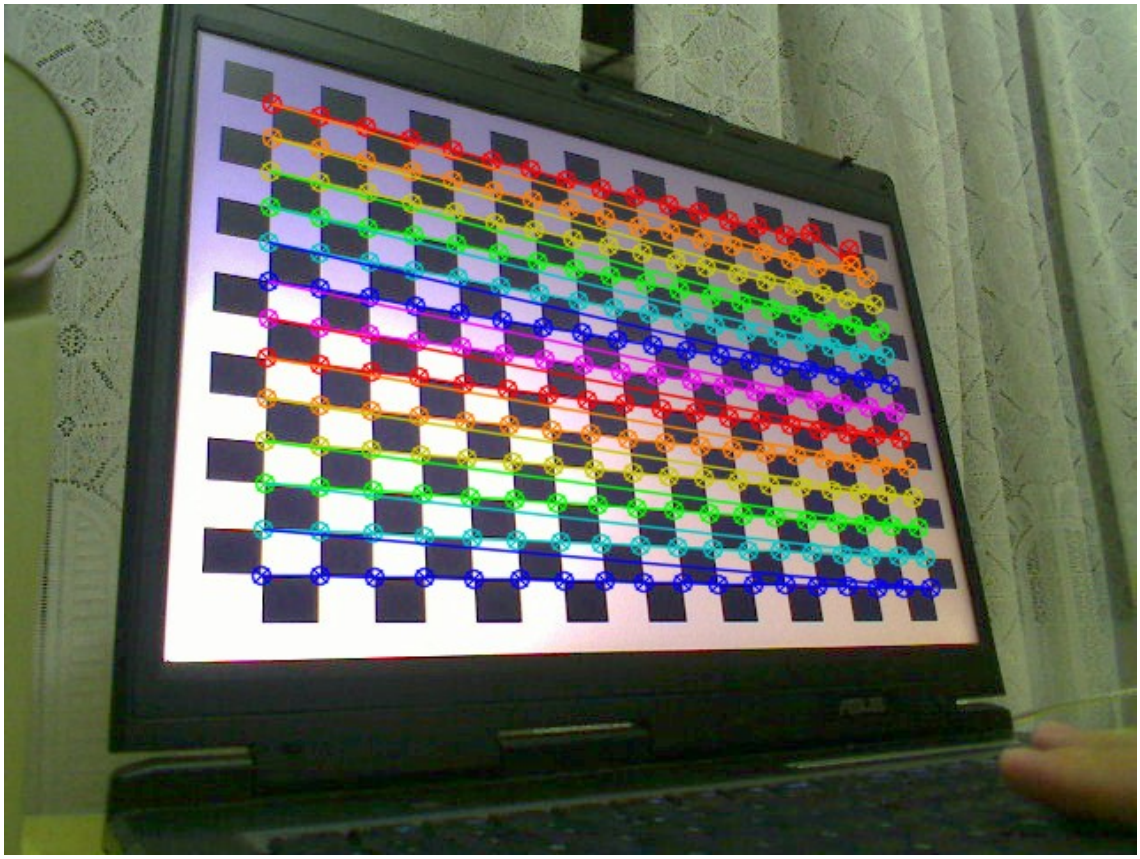

Figura 12 Exemplo de identificação de vértices com falha

Figura 13 Detalhe de falha de identificação de vértices

A figura a seguir é um exemplo de aplicação com sucesso dos algoritmos e consequentemente de identificação e ordenação de todos os vértices internos do padrão xadrez. Observa-se que mesmo no canto superior esquerdo, onde os retângulos são menores e a intensidade do branco é menor, obteve-se um resultado satisfatório.

Figura 14 Identificação correta de todos os vértices dos quadrados

Para que o processo de calibração seja realizado é necessário obter no mínimo três vistas diferentes para que os algoritmos tenham informações suficientes para realizar uma espécie de triangulação e determinar todos os parâmetros necessários de forma a minimizar o erro.

Desta forma, os parâmetros intrinsecos (centro da imagem e fatores de escala segundo os eixos i e j da imagem, α e β), extrínsecos (matriz de rotação 3D, R e vetor de rotação 3D, T) e matriz de coeficientes de distorção da câmera são obtidos com o método de Zhang [COELHO] implementado na biblioteca OpenCV. Com os coeficientes de distorção é possível reverter a distorção causada pelas lentes (distorção radial e tangencial) e com o uso da matriz de rotação e translação é

possível realizar o mapeamento de coordenadas da projeção para coordenadas do sistema operacional.

Uma vez executada a calibração, a ferramenta tem todos os dados necessários para o mapeamento das coordenadas. Após estes passos, todas as conversões de coordenadas são realizadas através de uma função simples:

```
CvPoint novacoordenada = mapear (CvPoint coordenadaproj),
```

sendo o `CvPoint` uma estrutura de dados do OpenCV, que representa um ponto (coordenadas x e y); `coordenadaproj` é o pixel da imagem obtida com a câmera ;a qual se deseja mapear; `novacoordenada` é o pixel correspondente na imagem original; e `mapear` a função que realiza esta conversão.

Devido a esta dificuldade de se determinar a profundidade associada a uma coordenada (x,y), procurou-se métodos alternativos de se realizar o mapeamento de coordenadas entre a imagem projetada e a imagem capturada. A solução encontrada foi a utilização da função *cvFindHomography* do OpenCV. Esta função tem o protótipo apresentado a seguir:

```
void cvFindHomography(const CvMat* src_points, const CvMat*
                      dst_points,  CvMat* homography )
```

Esta função busca determinar a matriz de homografia que associa os pontos de origem (imagem capturada pela câmera) e destino (reprojeção eliminando perspectiva) da seguinte forma:

```
  [x'i]   [xi]
si[y'i]~H*[yi]
  [1  ]  [ 1]
```

Nesta expressão (xi, yi, 1) é a representação homogênea do i-ésimo ponto de origem, (x'i, y'i, 1) é a representação homogênea resultante da multiplicação da matriz de homografia com o i-ésimo ponto de origem, H é a matriz de homografia e *si* é o i-ésimo fator de escala que associa o ponto resultante da multiplicação com sua representação homogênea.

O uso da matriz de homografia possui uma série de vantagens:

- Necessita de apenas uma vista do padrão xadrez para possibilitar a calibração da ferramenta. Este fator torna o processo de utilização muito mais simples para o usuário.
- Reduz a quantidade de memória necessária para a operação, uma vez que apenas um conjuntos de pontos de vértices necessita ser armazenado e apenas uma matriz 3x3 com os resultados
- Reduz o número de operações necessárias para calibração e para o próprio mapeamento de coordenadas. Enquanto o processo de calibração baseado em parâmetros instrínsecos e extrínsecos ajusta uma matriz de parâmetros de câmera, uma matriz de distorção de lente, uma matriz de rotação e translação para cada vista; o método de homografia precisa ajustar os elementos de apenas uma matriz. Para o mapeamento, o primeiro método precisa efetuar um cálculo levando em consideração todas as matrizes geradas, o que torna o processo muito mais dispendioso do ponto de vista de consumo de CPU.

A seguir, apresenta-se uma série de imagens que demonstram o resultado da utilização da matriz de homografia em testes. A Figura 15 é a imagem de resolução de 1280x800 pixels utilizada como testes. Esta imagem foi projetada no monitor LCD em tela cheia com uso da função criada *projetaImagem*.

Figura 15 Imagem Original usada para teste de mapeamento de coordenadas

A Figura 16 é uma imagem de 640x480 pixels do monitor LCD capturado pela câmera. É possível observar o efeito da baixa resolução da câmera, alteração de intensidade de pixels do monitor LCD e a perspectiva pela qual o monitor é observado. No momento desta imagem, o processo de calibração da ferramenta já havia sido realizado e, portanto, a matriz de homografia estava pronta para ser aplicada na imagem.

Figura 16 Imagem obtida pela câmera

A imagem seguinte é o resultado da aplicação da matriz de homografia na imagem obtida pela câmera. A partir da imagem de 640x480 pixels, o processo recria a imagem de 1280x800 presente no monitor, revertendo a perspectiva. O resultado é apresentado na imagem seguinte. Observa-se que, por inspeção visual, a imagem é muito semelhante à imagem original utilizada na projeção. As diferenças são devidas principalmente à diferença de resolução e efeitos de variação da intensidade dos pixels com o ângulo de observação do monitor LCD.

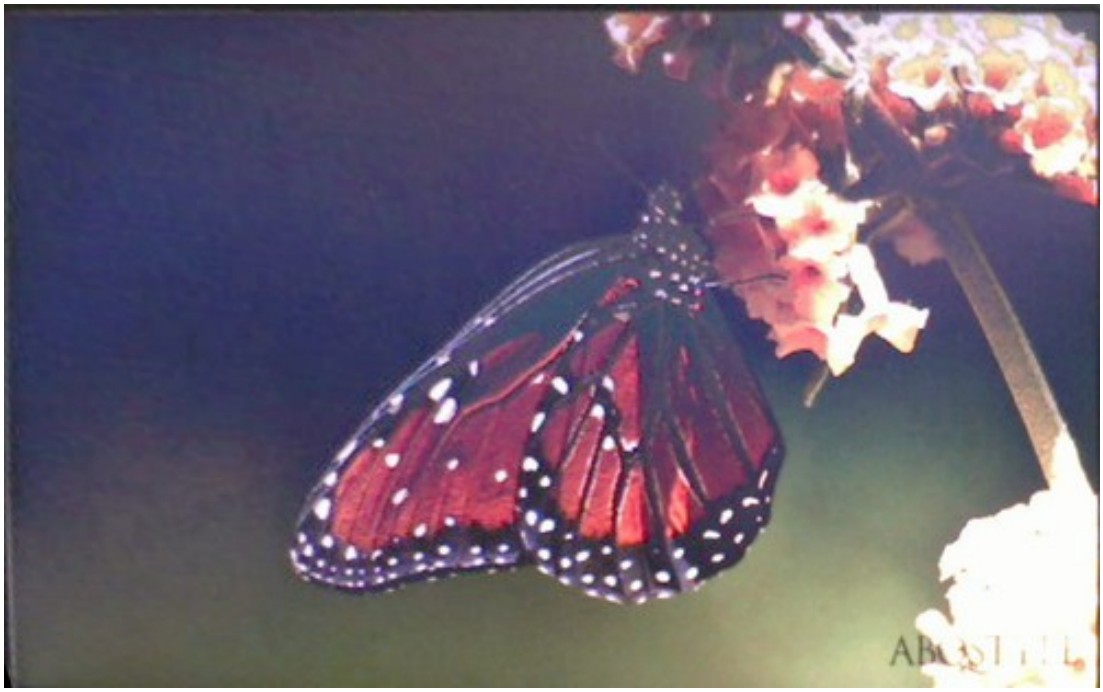

Figura 17 Imagem resultante do mapeamento de coordenadas (1280x800)

Prevê-se a utilização da ferramenta tanto com o conjunto câmera-projetor utilizado em apresentações, quanto para um conjunto câmera-monitor LCD (mostrado na **Figura 18**) como uma alternativa ao uso do mouse. Os dois casos são muito semelhantes para o uso da ferramenta, entretanto na segunda situação a interação não pode ser realizada com o *laser* pois, devido às características do LCD, o *laser* não se destaca sobre o monitor para permitir a aplicação dos métodos de reconhecimento de imagens de forma satisfatória.

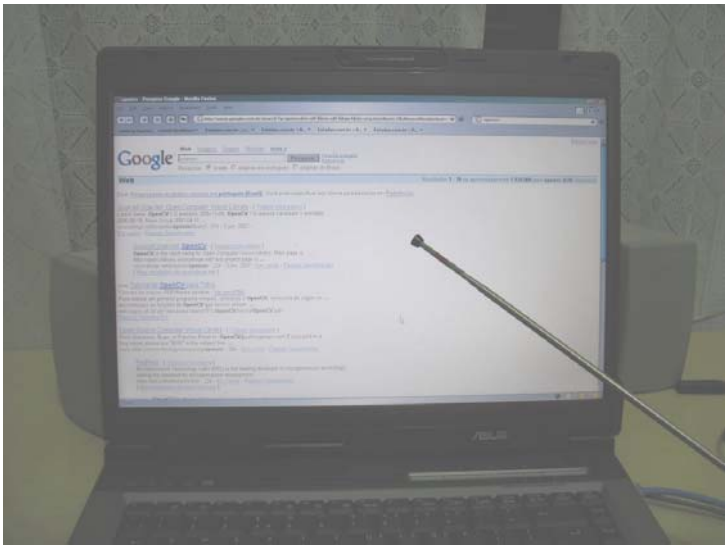

Figura 18 Demonstração de uso de apontador sobre monitor LCD

Caso a calibração não seja possível, uma série de instruções devem ser passadas ao usuário para que este possa melhorar as condições de uso e repetir a operação. Uma vez concluída a calibração, a aplicação retorna à janela de configuração para que o usuário possa modificar as opções ou iniciar o uso da ferramenta.

4.4.3 Reconhecimento de ponteiro.

Este módulo tem o objetivo de reconhecer o movimento e eventos gerados pelo ponteiro e enviar estes dados para o módulo de passagem de eventos ao sistema operacional.

Estão previstos o uso de dois tipos de ponteiros básicos: apontador *laser* e varinha telescópica, sendo que a ferramenta buscará nas imagens apenas o apontador que for escolhido pelo usuário no módulo de configuração.

O apontador *laser* e as varinhas com LEDs são identificado com base em sua cor, brilho e área ocupada na imagem. Desenvolveu-se o objeto CTracker, que implementa as funções do módulo de reconhecimento.

Na busca do ponteiro vermelho (LED ou *laser),* o objeto `CTracker` rastreia a imagem em busca de uma área que apresente componente RGB intensa (por exemplo, maior que 220), componentes azuis e verdes fracas (por exemplo, menor que 100) e de área pequena.

O algoritmo varre a a imagem em busca dos pixels que possuam estas características. Quado um ponto corresponde às características especificadas cria-se um objeto CBlob. Este objeto tem o objetivo de descrever uma área da imagem que apresente uma determinada característica. As propriedades deste objeto são: tipo de área encontrada (podendo apresentar valor 0 para áreas de tonalidade vermelha e 1 para áreas de tonalidade verde), área ocupada (número total de pixels da região que apresentam as propriedades procuradas) e coordenadas do menor retângulo que a envolva.

Utiliza-se este primeiro ponto encontrado como semente de um algoritmo de região de crescimento com conectividade-4 para localizar todos os pixels que estejam conectados à semente e se enquadrem nos parâmetros definidos. Para

cada pixel identificado atualiza-se o CBlob correspondente. Para cada quadro obtido pela câmera atualiza-se o CBlob. Ao final do processamento de um CBlob suas características são comparadas com certos parâmetros. Caso o CBlob não esteja de acordo com os parâmetros o objeto é descartado. Primeiramente, verifica-se a largura e comprimento do retângulo envoltório. Nenhum destes valores pode exceder 20 pixels ou ser menor que 2, pois, a região seria muito grande ou muito pequena. Em seguida a imagem continua a ser varrida em busca de outras regiões de interesse. Podem ser geradas outras CBlobs nesse processo. Após varrer toda a imagem, a CBlob de maior área é considerada como a válida para a passagem de eventos ao sistema operacional.

Para a varinha telescópica (**Figura 19**) estavam previstos alguns casos diferentes. Primeiramente, tentou-se rastrear o movimento da varinha a partir do reconhecimento de linhas geradas pelas laterais do objeto com uso da transformada de Hough, também disponível na biblioteca OpenCV. Entretanto, a presença de diversos elementos na imagem impediram o isolamento adequado das retas para o devido rastreamento. Além disso, a baixa taxa de amostragem associada a um elevado tempo de abertura da câmera geram rastros da movimentação do objeto. Estes rastros tornam inviável a utilização apropriada do algoritmo de identificação de retas.

Tentou-se também criar um padrão de cores que pudesse ser identificado como a extremidade da varinha (**Figura 20**). Entretanto, devido a presença do monitor LCD como uma fonte intensa de luz atrás do objeto, a câmera ajusta sua abertura de forma que este padrão de cores não seja identificável. Devido a estas dificuldades, o ponteiro não pode ser rastreado por outro método que não a identificação dos LEDs.

Figura 19 Apontador para interação da ferramenta

Figura 20 Detalhe do padrão para reconhecimento do apontador

Verificou-se dificuldade na utililização dos LEDs infravermelhos como marcadores dos ponteiros. Estes LEDs seriam interessantes por serem emitirem luz em um espectro invisível ao olho humano, mas serem captados por *webcams*. A **Figura 21** apresenta um LED infravermelho ativo de um controle remoto registrado por uma webcam. Desta forma, os ponteiros seriam mais discretos. Entretanto, a imagem gerada pelas câmeras apresenta os LEDs infravermelhos como pontos brancos. A câmera geralmente ajusta sua abertura e tempo de abertura de acordo com a iluminação observada, em muitos casos este ajuste faz com que áreas da imagem sejam muito brilhantes (devido ao fundo ou devido ao monitor LCD). Estas áreas brilhantes também apresentam a cor branca (RGB=[255,255,255]). Assim, a

utlização dos LEDs infravermelhos não apresenta resultados satisfatórios, por não ser distingüível de áreas do monitor LCD.

Figura 21 Efeito de LED infravermelho de controle remoto na imagem de uma webcam

Para o uso de varinhas para interação com projeção ou monitor LCD é importante que a mesma seja movimentada próxima ao plano de projeção. De outra forma, o posicionamento da câmera pode gerar incoerências no mapeamento das coordenadas.

Uma vez que o tipo de ponteiro e sua posição seja identificada, este módulo passa estes dados para o objeto CMouseEvent que é descrito na seção 4.4.4.

4.4.3.1 Construção de ponteiros

Após os diversos testes de reconhecimento de imagem realizados, montaram-se os ponteiros apresentados e descritos a seguir:

Figura 22 Ponteiro 1 com LED vermelho

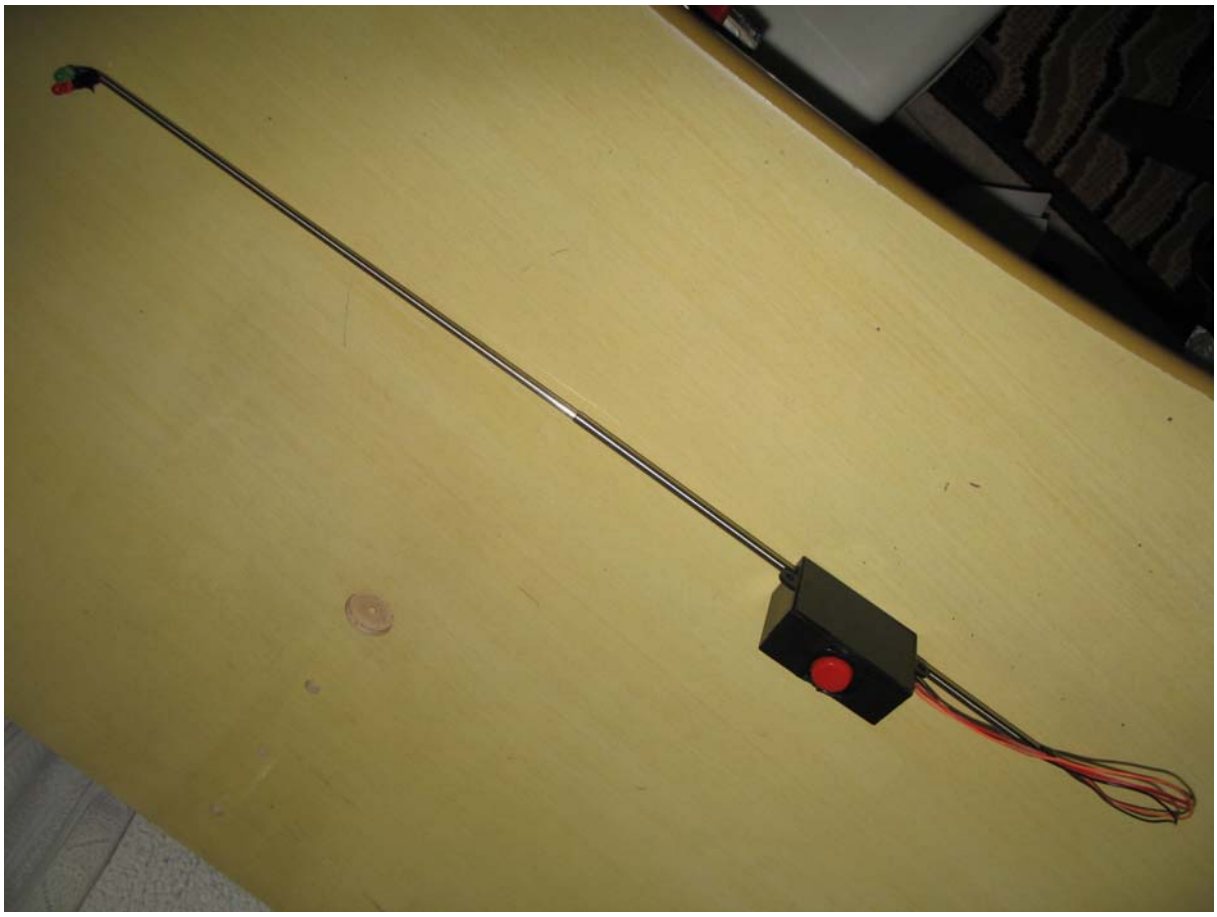

Figura 23 Ponteiro 2 com LED vermelho e verde

Construíram-se dois ponteiros diferentes. Um deles com apenas um LED vermelho (ponteiro 1) e outro com um LED vermelho e outro verde (ponteiro 2) que pode permitir mais eventos. O circuito para ligação de um LED é bem simples, consistindo apenas da fonte de tensão, o próprio LED e um resistor para limitar a corrente e ajustar o nível de tensão. Os valores padrão de tensão e corrente para LEDs coloridos são apresentados a seguir:

Tabela 2: Valores padrão para LEDs de 5mm

Cor	Queda de Tensão	Corrente Máxima
Vermelho	1.8V	0.02 A
Verde	2.1V	0.02 A

No circuito, a queda de tensão no resistor é a diferença entre a tensão da bateria e a queda de tensão no LED e a corrente é a própria corrente de operação do LED. Assim:

$$R = \frac{V}{I}$$

$$R = \frac{(3-1{,}8)V}{0.02A} = 60\Omega$$

Para o LED verde obtemos:

$$R = \frac{(3-2.1)}{0.02A} = 45\Omega$$

Pode-se utilizar o resistor de 68Ω para ambos os LEDs uma vez que nenhum deles excede sua corrente máxima com esta escolha. O LED verde apresenta uma luminosidade intensa, portanto é interessante utilizar uma corrente abaixo da máxima para reduzi-la. Os componentes utilizados foram:

- Antena telescópica de 3 segmentos
- Botão com 2 contatos para ponteiro 1(estabelece contato apenas quando pressionado)

- Botão com 3 contatos para ponteiro 2 (estabelece contato entre A e B ou B e C, apenas quando pressionado)
- Resistor de 68Ω
- Suporte para bateria
- Bateria CR2032 de 3V
- LEDs vermelho e verde difusos de 5mm
- Fios
- Caixa plástica (4x3x2,5cm) para ponteiro 1
- Caixa plástica (6,5x3,2,5cm) para ponteiro 2

Os ponteiros forma montados de forma que pudessem ser segurados e movimendados com facilidade. Selecionou-se uma caixa que fosse pequena e pudesse acomodar todo o circuito. Além disso, que permitisse a fácil substituição da bateria. Buscou-se escolher botões fáceis de serem pressionados. Os fios foram passados por dentro da antena telescópica por razões estéticas. A varinha montada tem cerca de 50 cm. O resultado é apresentado na figura a seguir:

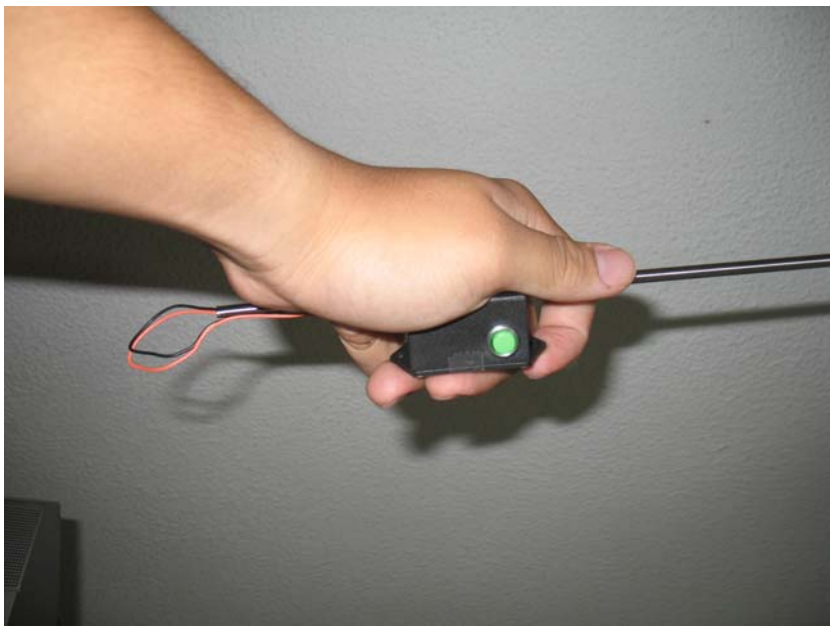

Figura 24 Detalhe de botão do ponteiro 1

A figura a seguir ilustra o uso de ponteiro do tipo 2 com monitor LCD de um notebook.

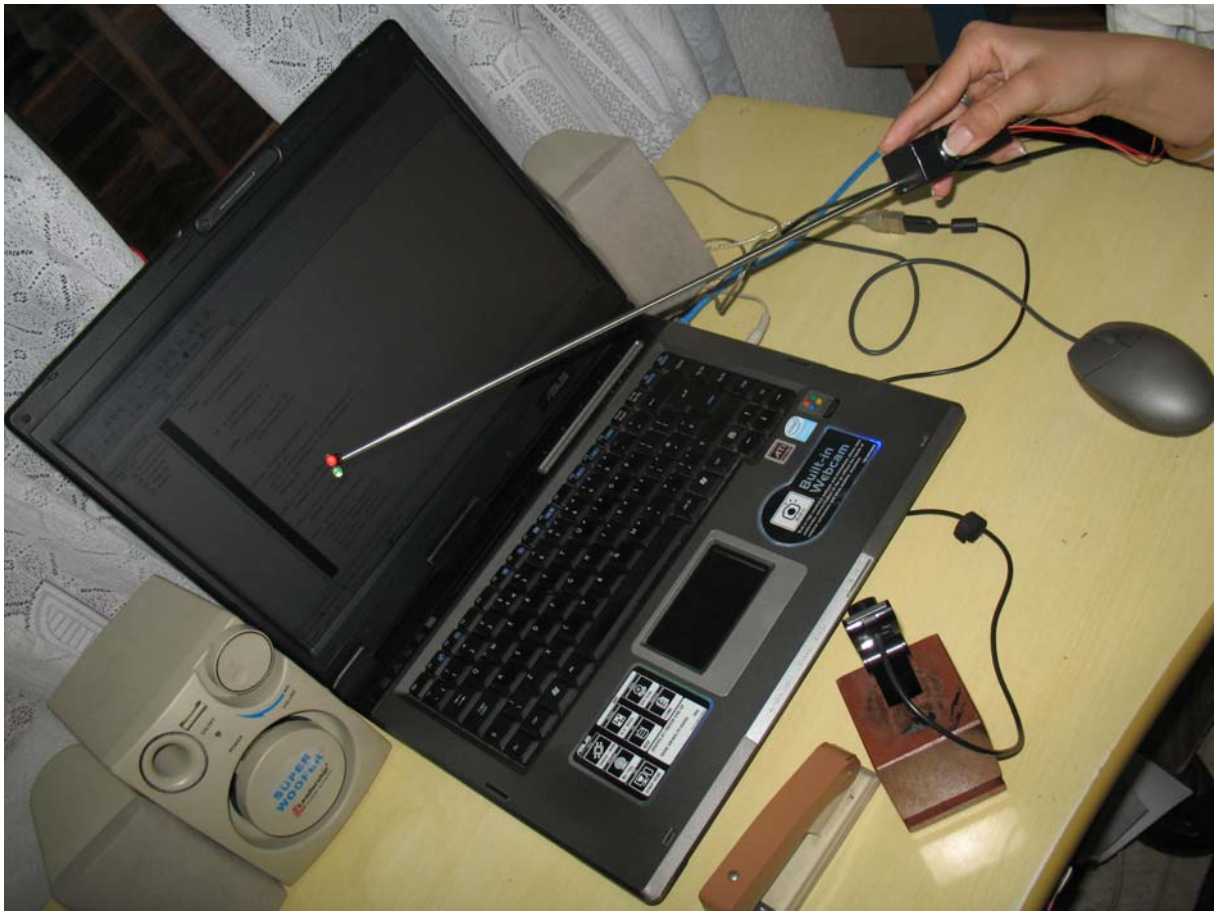

Figura 25 Demonstração de uso de ponteiro 2 com monitor LCD

4.4.3.2 Sistema de redução de uso de CPU

A ferramenta requer o uso constante do processador para analisar a imagem da câmera em busca do ponteiro. Entretanto, dada a natureza dos processos de ensino, sabe-se que boa parte do tempo não haverá nenhum tipo de interação com a projeção e o uso de processador poderia ser evitado. Assim, há um sistema que reconhece esta situação e diminui a taxa de atualização das imagens. Deste modo, em pleno uso a taxa de atualização é de no mínimo 5 quadros por segundo (devido a limitações da câmera, usualmente menor que 16 quadros por segundo), mas após 70 quadros sem nenhuma movimentação de ponteiros observada esta taxa é reduzida para apenas um quadro por segundo. A ferramenta retorna à operação normal quando o ponteiro é novamente encontrado.

Este algoritmo foi implementado no objeto `CMouseEvent`, que é responsável pela tratamento das coordenadas e características dos ponteiros encontrados e pela chamada dos eventos no sistema operacional.

4.4.4 Passagem de eventos ao sistema operacional

Este módulo é responsável pela geração dos eventos necessários no sistema operacional. Na linguagem C, a geração de eventos de *mouse* pode ser realizada através da função *mouse_event* da biblioteca `windows.h` que facilita a utilização da biblioteca de ligação dinâmica do windows (DLL) `user32.dll`. Esta função recebe uma série de parâmetros descritos em [MSDN. Mouse...**].** Desta forma, é possível gerar, no Windows, qualquer tipo de evento do mouse na janela que esteja ativa.

Os parâmetros passados para esta função definem os estados dos botões esquerdo, direito do mouse entre duas opções possíveis: pressionado ou liberado. O sistema de coordenadas utilizado pode ser tanto o absoluto (onde o parâmetro é exatamente a coordenada que se deseja realizar o evento) ou relativo (em que o sistema de coordenadas tem origem na última posição do mouse em uma chamada de sistema). Devido ao mapeamento da área de projeção, o sistema utilizado é o de coordenadas absolutas.

As operações de arrastar-e-soltar (*drag-and-drop*) podem ser realizada através de uma sequência de eventos mais simples do mouse como o pressionamento de um botão (*mouse down)* seguido do movimento do mouse e por fim a liberação do botão utilizado (*mouse up*).

Para a simplificação da codificação futura dos eventos passados ao sistema operacional optou-se pela criação de protótipos para os mais diversos eventos gerados pelo mouse no sistema operacional, mesmo que alguns pudessem ser simplesmente a repetição ou a combinação de outros. A tabela a seguir apresenta os protótipos criados e seus respectivos significados:

Tabela 3: Protótipos de função de passagem de eventos de mouse

Protótipo da função	Operação no sistema operacional

Protótipo da função	Operação no sistema operacional
void movimentamouse(int x, int y);	Movimenta mouse para as coordenadas (x,y)
void mouseleftdown(int x, int y);	Pressiona o botão esquerdo no mouse nas coordenadas (x,y)
void mouseleftup(int x,int y);	Libera o botão esquerdo no mouse nas coordenadas (x,y)
void mouseclick(int x, int y);	Gera um evento de clique duplo do botão esquerdo do mouse nas coordenadas (x,y)
void mouseclickduplo(int x, int y);	Gera um evento de clique do botão esquerdo do mouse nas coordenadas (x,y)
void mouseclicktriplo(int x, int y);	Gera um evento de clique triplo do botão esquerdo do mouse nas coordenadas (x,y)
void mouserightdown(int x, int y);	Pressiona o botão direito no mouse nas coordenadas (x,y)
void mouserightup(int x,int y);	Libera o botão direito no mouse nas coordenadas (x,y)
void mouserightclick(int x, int y);	Gera um evento de clique duplo do botão direito do mouse nas coordenadas (x,y)
void mouserightclickduplo(int x, int y);	Gera um evento de clique do botão direito do mouse nas coordenadas (x,y)

A estratégia de desenvolvimento destas instruções permite que trabalhos futuros possam portar a ferramenta de realidade aumentada em diferentes sistemas operacionais, ou mesmo, em versões futuras do Windows sem ter de compreender e modificar a totalidade do código desenvolvido neste projeto.

Como a identificação do ponteiro e de seu estado é realizada pela amostragem da imagem da câmera, todas as funções de eventos do mouse (com exceção da movimentação) dependem da combinação de uma série de amostras. Devido a isto, os eventos do mouse são encapsulados no objeto chamado de `CMouseEvent`. Os principais métodos disponíveis neste objeto são `initCMouseEvent()`, que simplesmente inicializa o objeto; `addEstadoMouse2()`, que é utilizado para acrescentar ao objeto uma estrutura

composta pela posição do ponteiro e do estados de seus botões e *addVazio2*(), que indica que a amostragem da câmera não apresenta o ponteiro em uma coordenada válida. A função `addEstadoMouse2` recebe como parâmetros o tipo de ponteiro encontrado e a coordenada em que se localiza.

A sequência de resultados da amostragem é avaliada pelo próprio objeto `CMouseEvent`, que também se encarrega da chamada da função correspondente no sistema operacional com o uso da biblioteca `windows.h`. Inicialmente tentou-se identificar os eventos através de comparação da seqüência de resultados das amostras com modelos pré-determinados. Por exemplo, um evento de clique com o botão esquerdo do mouse seria gerado quando fosse identificada a sequência de exatamente um vazio (amostragem na qual não se identifica ponteiro), uma amostra na qual o ponteiro associado ao botão esquerdo é identificado, seguido por dois vazios. Entretanto, devido à taxa de amostragem da *webcam* ser variável de acordo com a luminosidade e aos sensores da câmera apresentarem certa persistência de imagens, esta abordagem não foi possível. A persistência de imagens na câmera faz com que o LED acesso seja detectável além do período em que de fato estave ativo. Assim, uma ativação mesmo curta era observável em uma série de quadro. A própria sensibilidade do botão e a forma de operação por parte do usuário faziam com que o ponteiro fosse detectado em mais quadros que esperado. Além disso, a movimentação do ponteiro gerava um rastro que, de acordo com as normas estabelecidas para identificação do ponteiro, eram identificado como um vazio. Assim, criou-se regras mais robustas para o tratamento a identificação dos ponteiros e as funções `addEstadoMouse()` e `addVazio()`, que implementam o antigo algoritmo, foram abandonadas.

Criou-se uma estrutura de lista que armazena não apenas o resultado de quadros, mas sim agrupamentos de eventos. Assim, os elementos da lista reunem todas as identificações consecutivas de um determinado ponteiro em um único registro. Assim, caso seja registrado o ponteiro vermelho em 5 quadros consecutivos, será gerada apenas uma entrada. Este elemento da lista registrará o tipo de ponteiro identificado (vermelho ou verde, por exemplo), o número de ocorrências e a coordenada da primeira e última ocorrência. Nesta abordagem, um evento de clique é identificado quando existe um elemento da lista indicando amostras vazias, seguido de um elemento da lista indicando o ponteiro vermelho

com número de ocorrências menor que 5, seguido de outro elemento vazio na lista. Além do agrupamento de eventos da mesma categoria, foi necessário criar uma série de regras para filtrar resultados espúrios. Assim, caso a identificação do ponteiro vermelho falhe e a ferramenta registre uma única amostra vazia, esta amostra é descartada. O mesmo ocorre com qualquer outro evento. Uma série de amostras vazias com uma única identificação de ponteiro no meio, tem este elemento estranho eliminado. A regra que define a execução do evento *mouseDown* é semelhante, porém a repetição de amostras com ponteiro deve ser maior que 5. O evento *mouseUP* que o acompanha é gerado quando após uma série de mais de 5 repetições do ponteiro é seguida por amostras vazias. O evento de clique duplo é gerado quando dois eventos de clique são gerados com um espaçamento de amostras vazias menor que 5 quadros. Como durante os testes da ferramenta não se detectou variação significativa da taxa de amostragem da câmera o uso de regras baseadas em número de quadros apresenta um resultado consistente para o uso da ferramenta.

A utilização destas regras para determinação dos eventos criou uma nova dificuldade, que era o controle da memória ocupada pela ferramenta. Assim, periodicamente e de acordo com certas regras *o array* que armazena os resultados da identificação do ponteiro deve ser apagado, de forma a evitar o consumo excessivo de memória. Desta forma, por exemplo após os eventos de *mouseDown* e clique duplo já não existe mais a necessidade de avaliar amostras passadas, portanto, as mesmas podem ser eliminadas.

As mesmas regras de definem a remoção de amostras de um determinado tipo sem nenhuma repetição é utilizada para filtrar os pontos que são passados ao objeto de *mouse gestures*.

Para suporte ao módulo de *mouse gestures* implementou-se funções para a passagem de eventos do teclado e para a geração de comandos do sistema operacional. A passagem de eventos do teclado é gerado com um método do objeto CSendKeys. Este método tem o prótotipo a seguir :

```
bool SendKeys (string KeysString, bool Wait = false)
```

Neste método KeyString é uma string que concatena toda a sequencia de teclas que deseja-se enviar. Nesta *KeyString* também são concatenadas algumas estruturas pré-definidas que representam as teclas especiais do teclado (ALT, TAB, CONTROL, etc). O objeto SendKeys é baseado no trabalho de código aberto de

Lallous que por sua vez é baseado no trabalho na linguagem Delphi de Ken Henderson.

A lista de estruturas pré-definidas é apresentada na lista a seguir:

Tabela 4: Códigos para geração de eventos de teclado

Tecla	Código gerado
BACKSPACE	{BACKSPACE}, {BS}, or {BKSP}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
DEL or DELETE	{DELETE} or {DEL}
SETA PARA BAIXO	{DOWN}
END	{END}
ENTER	{ENTER} or ~
ESC	{ESC}
HELP	{HELP}
HOME	{HOME}
INS or INSERT	{INS}
SETA PARA A ESQUERDA	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
PRINT SCREEN	{PRTSC} (reservado para uso futuro)
SETA PARA A DIREITA	{RIGHT}
SCROLL LOCK	{SCROLL}
TAB	{TAB}
SETA PARA CIMA	{UP}
F1	{F1}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}
F6	{F6}
F7	{F7}
F8	{F8}
F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}
F13	{F13}

Tecla	Código gerado
F14	{F14}
F15	{F15}
F16	{F16}
SOMA (teclado numérico)	{ADD}
SUBTRAÇÃO (teclado numérico)	{SUBTRACT}
MULTIPLICAÇÃO (teclado numérico)	{MULTIPLY}
DIVISÃO (teclado numérico)	{DIVIDE}
+	{PLUS}
@	{AT}
APPS	{APPS}
^	{CARET}
~	{TILDE}
{ }	{LEFTBRACE} {RIGHTBRACE}
( )	{LEFTPAREN} {RIGHTPAREN}
TECLA WINDOWS DIREITA/ESQUERDA	{LWIN} {RWIN}
TECLA WINDOWS	{WIN} equiavalente a {LWIN}
TECLA WINDOWS	@
SHIFT	+
CTRL	^
ALT	%

A passagem de eventos do teclado é semelhante a passagem de eventos do mouse. Primeiramente inicializa-se a DLL user32.dll e então utiliza-se a instrução `SendKeys::SendWait()[MSND. How to...]`.

Operações do sistema operacional são realizadas com a chamada `system(<comando>)` da biblioteca de utilitários padrão da linguagem C `<stdlib.h>`. Com esta instrução é possível executar aplicativos, abrir arquivos e manipular o sistema operacional (criar, mover e apagar arquivos e diretórios; alterar variáveis de ambiente, etc).`(CPLUSPLUS)`.

A passagem de comandos diretamente ao sistema operacional, embora seja simples de ser realizada através de uma instrução direta na linguagem C++ foi implementada através do protótipo de função apresentado a seguir, onde *s* é a string do comando que se deseja executar.

```
int comandoSO ( const char *s  )
```

Embora não seja o foco deste trabalho, projetos futuros podem acrescentar diversos recursos que estejam disponíveis no sistema operacional através de DLLs.

4.4.5 *Mouse-gestures*

Este módulo é responsável pela reconhecimento de trajetos pré-determinados percorridos pelo ponteiro com um determinado LED ativo. Quando identificados, a ferramenta executa um comando definido previamente pelo usuário. Por exemplo, o caminho do ponteiro indo de baixo para cima e depois da direita para a esquerda pode gerar o comando de execução do software “calculadora” do Windows XP.

Os gestos estariam listados no módulo de configuração onde o usuário poderia definir uma ação para cada um deles. Entre as tarefas pode-se citar a geração de eventos do teclado (com uso do objeto CsendKeys), por exemplo, ESC, ENTER, F9, Ctrl+C, Ctrl+V; execução de programas, etc.

A proposta inicial era o desenvolvimento deste módulo com base no sistema de reconhecimento de gestos de Konstantin Boukreev que consiste numa rede neural treinada para o reconhecimento de 29 padrões diferentes de trajetos. Esta rede neural possui as seguintes características [BOUKREEV] :

- Camama de entrada de 32 sinapses
- Camada escondida de 32 neurônios
- Camada de saída de 29 axônios
- Camadas completamente conectadas
- Função de tranferência logarítmica-sigmoidal
- Algoritmo de treinamento incremental, método padrão de *back-propagation*

Entretanto, considerando o treinamento da rede neural envolvido, tempo de processamento necessário para cada ação do usuário para a determinação do gesto e o consumo de memória necessária, optou-se por abandonar o uso da rede

neural e adotou-se uma abordagem simplificada, mas que atendesse aos requisitos de número de gestos mínimos (8) reconhecidos pela ferramenta.

Este módulo foi implementado como um objeto chamado de CMouseGesture. Este objeto trabalha em três fases distintas: inicialização (variáveis são apagadas e objeto é preparado para receber dados), acumulação de dados (objeto recebe os pontos que compõe o trajeto) e finalização (determinação da trajetória realizada pelos ponteiros e chamada de eventos no sistema operacional).

Quando o objeto `CTracker` identifica a repetição de amostras que contenham o ponteiro por mais de 5 quadros, chama-se a função initGesture() e passa-se as coordenadas dos pontos encontrados como parâmetros da função `addPoint()`. Todos os pontos seguintes também são acumulados no objeto `CMouseGesture`. As mesmas regras que filtram resultados espúrios que são encontrados durante a identificação de ponteiros são aplicadas ao pontos passados a `CMouseGesture`. Quando a sequência de amostras nas quais o ponteiro é identicado é interrompida, os pontos correspondentes ao gesto são finalizados. Assim, `CTracker` chama a função `concludeGesture()` finalizando o processamento.

. Quando a função `concludeGesture()` é chamada, ocorre a análise dos pontos inseridos. Entre dois pontos consecutivos determina-se um vetor que inicia-se no primeiro ponto e termina no segundo. Verifica-se o ângulo do vetor simplesmente pela maior componente do vetor (em x ou y). A partir desta análise, classifica-se o vetor entre ‘direita’, ‘esquerda’, ‘para cima’ ou ‘para baixo’. Para cada uma das direções concatena-se uma diferente letra em uma string de controle: ‘D’ (para vetor a direita), ‘E’ (esquerda), ‘C’ (para cima) ou ‘B’ (para baixo). Elimina-se repetições consecutivas de vetores na mesma direção. Quando avalia-se todos os pontos inseridas, a string de controle apresenta uma sequência de letras que determina unicamente o gesto realizado. Em seguida, verifica-se se há algum tipo de comando associado à string produzida e chama-se a função do módulo de passagem de eventos ao sistema operacional correspondente. A imagem a seguir apresenta um exemplo simples de identificação de mouse gestures.

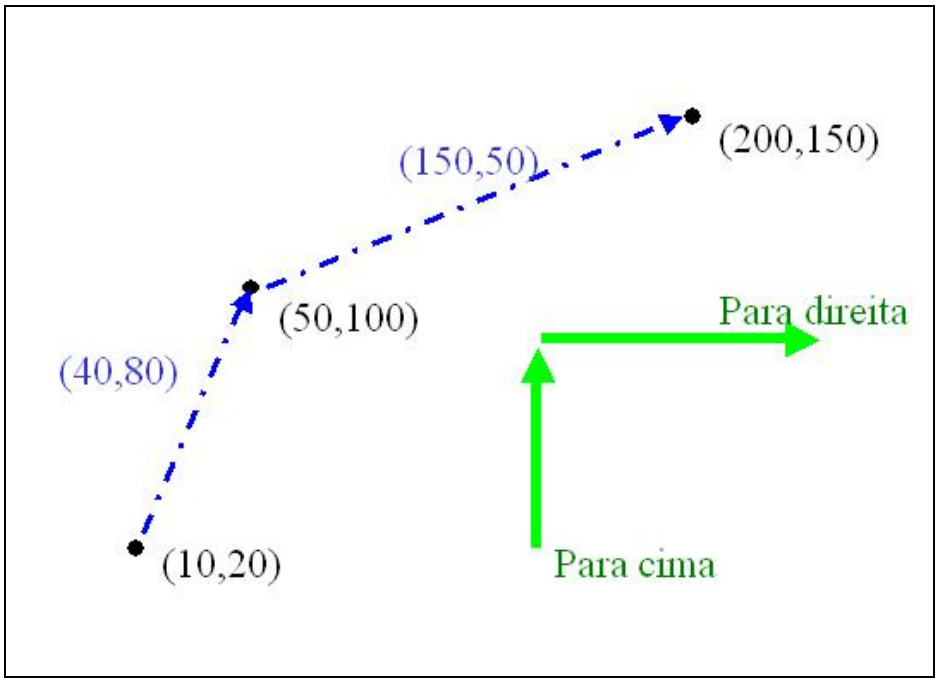

Figura 26 Exemplo de identificação de *gesture*

Neste exemplo amostra-se apenas três pontos diferentes (representados em preto), sendo o primeiro ponto aquele localizado na parte inferior direita da imagem e o último na parte superior direita. Em seguida, definem-se os vetores indicados em azul. Para a definição dos vetores resultantes verifica-se a maior componente dos vetores. Para o primeiro vetor (50 - 10, 100 - 20) obtém-se (40,80) e o vetor é classificado 'para cima'. Para o segundo (150,50), classifica-se como 'para a direita'. Este resultado é ilustrado pelos vetores em verde na parte inferior esquerda da imagem. Com este resultado ("Cima-Direita") procura-se por algum tipo de ação associada e a executa por intermédio do módulo de Passagem de Eventos ao Sistema Operacional.

Alguns sistemas existentes que operam com *mouse gestures* costumam classificar as direções entre 8 possíveis tipos (os quatro utilizados neste trabalho somado às diagonais), entretanto, considerando a baixa taxa de amostragem e alguma dificuldade no manuseio de *lasers* e varinhas telescópicas evitou-se utilizar um modelo complexo que pudesse apresentar instabilidade nos resultados.

Uma vantagem interessante da substituição da rede neural pelo sistema descrito nesta seção está no número de possíveis gestos identificáveis pela ferramenta. Enquanto a rede neural foi inicialmente modelada para identificar um total de 29 gestos, o sistema baseado em vetores consegue identificar até 12 gestos

com apenas três pontos amostrados e 36 com quatro pontos. Basicamente o primeiro vetor pode estar em qualquer uma das direções (4), o segundo vetor para ser considerado deve ser diferente do primeiro, logo restam três opções e assim por diante. Logo, o sistema implementado é capaz de identificar um número ilimitado de gestos.

4.5 Aplicação conceito

Foi desenvolvida uma simples aplicação-conceito com o objetivo de demonstração da ferramenta, além de permitir testes de desempenho, acurácia e usabilidade.

4.6 Testes

A ferramenta desenvolvida é avaliada a partir dos seguintes indicadores avaliados durante a implementação e posteriormente com o uso da aplicação-conceito:

- Precisão no mapeamento da posição do ponteiro em coordenadas da imagem real
- Precisão na identificação dos eventos gerados pelo ponteiro
- Quantidade de memória RAM necessária na execução
- Tempo de CPU utilizado

A precisão na identificação dos eventos depende basicamente da precisão do mapeamento de coordenadas e precisão na identificação do ponteiro

Deseja-se que a ferramenta apresente taxa de amostragem mínima de cinco quadros por segundo e erro máximo entre mapeamento de posição do ponteiro e coordenada esperada de no máximo de 30 pixels, sendo este erro definido como a distância euclidiana entre as coordenadas:

$$Dist^2 = [(Xmapeado^2 - Xesperado^2) + (Ymapeado^2 - Yesperado^2)]$$

Define-se *(Xmapeado, Ymapeado*) como as coordenadas do mapeamento da posição do ponteiro e (*Xesperado,Yesperado*) como coordenadas esperadas deste processo.

A taxa de amostragem mínima será controlada pelo tempo de CPU utilizado, uma vez que a taxa com a qual a *webcam* opera está relacionada à iluminação recebida por esta, sendo um fator que não pode ser alterado pela ferramenta.

4.7 Correções

As correções durante o período de desenvolvimento da ferramenta são realizadas periodicamente de acordo com a evolução das etapas descritas no cronograma. Além da identificação e tratamento de erros também pode ocorrer o aperfeiçoamento de algoritmos ou mesmo a mudança de abordagem de solução de um problema visando um produto final mais robusto e de fácil operação. Um exemplo desta estratégia foi a substituição do método de mapeamento de coordenadas com base em parâmetros de câmera pelo uso de uma matriz de homografia, que resultou num processo mais rápido, com menor uso de memória e de muito mais fácil operação do ponto de vista do usuário.

5 RESULTADOS

Analisou-se o cenário tecnológico atual no campo de reconhecimento de imagens e de visão computacional. Identificaram-se bibliotecas e métodos relevantes para o desenvolvimento da ferramenta.

O estudo dos artigos e documentos técnicos permitiu uma melhor compreensão dos procedimentos necessários para a realização das três funcionalidades básicas deste projeto: mapeamento de coordenadas, reconhecimento de imagens e passagem de eventos ao sistema operacional.

Como estratégia de desenvolvimento, o problema foi dividido numa série de módulos que permitem o desenvolvimento e correção mais claros e simples de se realizar. Além disso, as referências bibliográficas relevantes deste projeto contém instruções importantes que facilitam a implementação e evitam a repetição de erros de pesquisas anteriores.

Espera -se obter um software de utilização intuitiva, com rotinas de inicialização simples e com desempenho que não prejudique outras aplicações no computar. Dessa forma, não pode uma utilização dos recursos da CPU em excesso. Como parâmetros de avaliação tem-se a precisão no mapeamento das coordenadas, na qual espera-se um erro máximo de 30 pixels (com metodologia definida na seção 4.5) e taxa mínima de atualização de 5 quadros por segundo. Assim, o programa capturará a imagem da câmera a cada 0,2 segundos para o rastreamento de movimentos e procura de eventos do ponteiro.

Obteve-se algoritmos para a obtenção dos parâmetros instrísecos e extrínsicos com grande precisão, além do cálculo da matriz de homografia. Desenvolveu-se funções para a facilitação do uso de janelas, o que permitiu a simplificação do processo para apresentação de imagem colorida (importante para geração de máscaras) e do padrão xadrez (criado também em tempo de execução). A ferramenta tem uma interface com o usuário simplificada em uma pequena lista de comandos que controlam os estados durante todo o processo de calibração até que a ferramenta esteja pronta para realizar a interação do usuário com o sistema operacional através da projeção. O reconhecimento do padrão xadrez projetado em um monitor LCD e capturado com a *webcam* mostrou-se muito robusto e o mesmo

resultado é esperado o uso do projetor multimídia uma vez que elimina-se o problema da variação de intensidade dos pixels devido ao ângulo de observação da tela LCD.

De acordo com a bibliografia utilizada, a identificação de vértices do padrão xadrez pode ser prejudicada pela presença de outros objetos no fundo. Para evitar este problema, que impediria o mapeamento de coordenadas, desenvolveu-se um algoritmo para a eliminação das áreas da imagem que não correspondam à projeção. Este processo é detalhadamente descrito na seção sobre a rotina de calibração. Entretanto, testes mostraram que a função não era necessária. Como em algumas situações futuras este procedimento pode ser necessário, as funções foram mantidas no código embora estejam desabilitadas.

Criou-se protótipos de funções mesmo para processos mais simples com o intuito de facilitar estudos, correções e projetos futuros, como por exemplo, versões do mesmo aplicativo para outros sistemas operacionais.

A calibração de câmera realizada com quatro diferentes vistas e com o uso do padrão xadrez com 19 colunas e 14 linhas obteve os seguintes resultados:

- **Matriz da câmera:**

Corresponde aos parâmetros intrínsecos da câmera. Esta apresentada na seguinte forma:

$$A = \begin{bmatrix} f_x & 0 & Cx \\ 0 & f_y & Cy \\ 0 & 0 & 1 \end{bmatrix}$$

Os parâmetros intrínsecos são os seguintes:

- *fx* – distância focal no eixo x (em pixels)
- *fy* – distância focal no eixo y (em pixels)
- (Cx, Cy) – centro da imagem (utilizado para cálculo da distância radial)

Os valores obtidos foram:

$$A = \begin{bmatrix} 547.95794678 & 0 & 309.66113281 \\ 0 & 543.58044434 & 247.36889648 \\ 0 & 0 & 1 \end{bmatrix}$$

É importante observar que o sistema de coordenadas está sendo transformado de uma imagem de 640x480 obtido com a câmera para uma resolução de vídeo utilizada no monitor LCD de 1280x800. Esta diferença de escalas se reflete no processo de calibração como maiores fatores de escala nos parâmetros intrínsecos.

- **Coeficientes de distorção:**

Estes coeficientes estão na seguinte forma:

$$D = \begin{bmatrix} k1 & k2 \\ p1 & p2 \end{bmatrix}$$

Sendo k1 e k2 os coeficientes de distorção radial e p1 e p2 os coeficientes de distorção tangencial. O OpenCV não trabalha com coeficientes de distorção de maior ordem. Portanto, como o erro obtido é pequeno, pode-se afirmar que esta aproximação modela bem a câmera utilizada. Os valores obtidos no teste foram:

$$D = \begin{bmatrix} 0.0500573608554928 & ,-0.1914204808431150 \\ -1.0049040713547811e-003 & -2.8432082278423903e-004 \end{bmatrix}$$

- **Erros de reprojeção por vista:**

Este erro é calculado aplicando-se as transformações descritas pelos parâmetros instrínsecos e extrínsecos ao conjuntos de pontos das coordenadas dos vértices dos retângulos da imagem projetada. Este processo é chamado de reprojeção. O erro é determinada pela média das somas das distâncias euclidianas entre os pontos reprojetados e os pontos levantados no processo de identificação de

vértices da imagem capturada. Usando esta metodologia, o erro em cada uma das vistas foi calculado e é apresentado a seguir (o último termo corresponde à vista na qual o usuário utilizaria a ferramenta):

$$\begin{bmatrix} 0.2049732371273204, \\ 0.2089685945429354 \\ 0.1901386945675581 \\ 0.1945747636322282 \end{bmatrix}$$

Média: 0.1996638224675105

- **Vetor de rotação da última vista:**

Este vetor composto de r1,r2 e r3 é uma forma simplificado da matriz de rotação (apresentada a seguir). A transformação de vetor de rotação para matriz de rotação (e vice-versa) é realizada com a função *cvRodrigues2* do OpenCV. O vetor de rotação encontrado foi:

$$\begin{bmatrix} -0.57094008 & 0.50401402 & -2.99972177 \end{bmatrix}$$

- **Vetor de Translação da última vista:**

Este vetor descreve a translação tx, ty e tz que foi identificada na última vista:

$$\begin{bmatrix} 385.28704834 \\ , 405.49856567 \\ 1.23693091e+003 \end{bmatrix}$$

- **Matriz de rotação da última vista:**

A matriz de rotação é utilizada em conjunto com o vetor de translação, os parâmetros intrínsecos e os coeficientes de distorção para realizar a reprojeção dos pontos. A matriz calculada a partir do vetor de rotação é apresentada a seguir:

$$\begin{bmatrix} -0.93088210 & -0.01480143 & 0.36501980 \\ -0.10530502 & -0.94589579 & -0.30690709 \\ 0.34981337 & -0.32413274 & 0.87895882 \end{bmatrix}$$

- **Taxa de atualização de quadros**

A taxa de atualização de quadros foi obtida registrando-se o horário do sistema operacional nos instantes que antecediam a captura da imagem pela câmera. Desta forma, o intervalo de tempo entre quadros consecutivos pôde ser medido e a taxa de atualização obtida estava entre 12 e 15 quadros por segundo. Como este valor é o mesmo tempo de atualização da câmera sem que haja qualquer tipo de processamento, pode-se concluir que os algoritmos são suficientemente eficientes para que todo o processamento associado a um quadro (reconhecimento de ponteiros, mapeamento de coordenadas, etc) seja realizado entre o intervalo de tempo de captura de dois quadros.

- **Reconhecimento de Ponteiros**

Diversas regras e algoritmos foram criados e aplicados ao reconhecimento do LED vermelho. A modificação desta programação pode levar a métos de reconhecimento de outros ponteiros.

- **Passagem de eventos ao sistema operacional**

Programou-se todos os eventos mais importantes de passagem de eventos de mouse para o sistema operacional. O objeto SendKeys é capaz gerar praticamente qualquer seqüência de eventos no teclado, sendo também de fácil uso.

- **Mouse Gestures**

Criou-se um objeto em C++ para tratar uma seqüência de pontos e reconhecer alguns padrões de trajetos. Esta estrutura está implementada de forma a

identificar 12 diferentes trajetos com apenas duas diferentes direções na movimentação ou 36 contando com 3 diferentes direções.

6 DISCUSSÃO

Embora alguns dos procedimentos para tratamentos de imagens já estejam implementados na biblioteca OpenCV, é imprescindível garantir as condições de uso dos mesmos, por exemplo, relata-se uma dificuldade de identificação de todos os vértices dos quadrados nas rotinas de calibração. Este problema pode ser solucionado com os procedimentos mencionados na seção 4.4.2. Entretanto, nos testes já realizados não foi encontrada a dificuldade para a identificação dos vértices mesmo com a baixa qualidade da imagem da câmera, baixa resolução e efeitos de variação de intensidade de pixel com a inclinação do monitor LCD. Mesmo assim, os algoritmos para criação de máscara e eliminação de fundo são mantidos com a ferramenta, porém, desabilitados.

Alguns dos métodos necessitam de maior esforço computacional, como por exemplo, a reversão de distorção de lentes, e podem ser substituídos ou mesmo suprimidos visando ao atendimento de todos os requisitos de projeto, em especial o desempenho e precisão da ferramenta. Portanto, avaliou-se o valor dos parâmetros de câmera, o erro envolvido na distorção de lentes e o tempo necessário para efetuar a reversão. Como a distorção de lente observada foi baixa, pode-se considerar que não há qualquer distorção, o que implica em um erro acumulado maior, entretanto, mantido dentro dos requisitos pré-estabelecidos. O método de mapeamento de coordenadas com uso dos parâmetros intrínsecos e extrínsecos da câmera encontrou uma séria dificuldade na determinação da profundidade associada aos pixels na imagem obtida pela câmera. Durante pesquisas para a solução deste problema encontrou-se um método alternativo para relacionar os dois sistemas de coordenada. A determinação da matriz de homografia permitiu um mapeamento preciso das coordenadas, com a vantagem de reduzir drasticamente o número de cálculos necessários, já que o uso dos parâmetros de câmera necessitariam de duas multiplicação com matrizes 3x3 e uma soma de vetores. Além do tempo de CPU necessário, o uso da matriz de homografia reduz o número de imagens de calibração de um mínimo de três para apenas uma, o que torna o processo de aprendizagem de uso da ferramenta muito menor. Outra vantagem deste método é a menor quantidade de memória necessária durante o processo de calibração.

Esta ferramenta serve como um substituto do mouse. Entretanto, como se utiliza do reconhecimento de imagens ao invés de um dispositivo físico com sensores é necessário observar algumas limitações. A principal delas é a resposta a eventos que estará relacionada a taxa de amostragem das imagens da câmera. Por exemplo, um evento de clique é identificado quando dentro de um pequeno intervalo de tempo o usuário aciona e desliga um LED colorido da varinha. Todavia, se esta operação de ligar e desligar for realizada muito rapidamente, especificamente entre dois quadros consecutivos obtidos pela ferramenta, o evento não será capturado. Estas mesmas limitações se aplicam ao uso dos *mouse gestures.*

Este projeto encontrou uma série de instabilidades durante a implementação de métodos de identificação de ponteiros, como é de se esperar para sistemas de reconhecimentos de imagem. Entre as principais dificuldades encontradas estão o reconhecimento de diferentes LED além do vermelho e problemas de aplicação de transformada de Hough (reconhecimentos de linhas). Por outro lado, após a aplicação de uma série de regras específicas, a utilização dos LEDs vermelhos pode ser aprimorada, de forma a validar a ferramenta. A modificação dos ponteiros pode permitir o uso da ferramenta em diferentes circunstâncias. Por exemplo, o uso de LEDs maiores (como os de 10mm podem permitir o uso da ferramenta a uma maior distância e ponteiras *laser* com um ângulo de abertura suficientemente grande pode permitir o uso da ferramenta com projetores).

Trabalhos futuros podem aplicar regras semelhantes e aprimorar os ponteiros de forma a aumentar as funcionalidades da ferramenta.

Esta ferramenta foi desenvolvida para o sistema operacional Windows, mas o único módulo dependente de plataforma é o de passagem de eventos ao sistema operacional. Assim, basta reescrever este trecho para uma outra plataforma, por exemplo o Linux, para se obter uma nova versão funcional da ferramenta, uma vez que o OpenCV também possui bibliotecas para o Linux. Isto também mostra a importância da modularização do sistema, pois a utilização de protótipos de funções tornam mais transparentes para um futuro programador os procedimentos envolvidos.

7 CONCLUSÕES

O avanço tecnológico no mundo e a difusão de computadores e outros equipamentos eletrônicos no ambiente escolar abrem espaço para o desenvolvimento de novas ferramentas direcionadas à educação. Este projeto se insere neste contexto utilizando a visão computacional para oferecer maior interatividade aos processos de ensino.

A divisão da ferramenta em módulos que interagem através de passagem de parâmetros para funções, além de facilitar a implementação e correções, facilitará o desenvolvimento de versões para diferentes sistemas operacionais; a integração da ferramenta com ambientes de video-conferência em que a interação de um dos participantes com uma projeção ou monitor LCD podem ser passada para todos os demais; e o desenvolvimento de jogos. O desenvolvimento dos algoritmos visando um baixo uso de CPU, embora tenha levado algumas simplificações do projeto resultou em uma ferramenta leve e eficiente.

Apesar da dificuldade no mapeamento de coordenadas com uso dos parâmetros instrínsecos e extrínsecos, o desenvolvimento realizado pode ser utilizado em outros tipos de aplicação. A matriz de homografia se mostra uma ferramenta com bons resultados para conversão das coordenadas.

A identificação de ponteiros, assim como outras aplicações de reconhecimento de imagens, é sujeita a uma série de instabilidades, assim, foi necessário desenvolver algoritmos mais robustos e simplificar a forma de operação da ferramenta. As limitações tanto de processamento quanto de memória mostraram-se um desafio adicional ao desenvolvimento. Apesar de algumas instabilidades, o protótipo apresenta um resultado satisfatório.

8 CRONOGRAMA

Tabela 5: Cronograma do período de projeto

Tarefa	Início	Término
Estudo sobre ambiente/ferramentas de desenvolvimento	05/03/2007	13/03/2007
Configuração de ambiente de desenvolvimento	14/03/2007	20/03/2007
Levantamento de problemas a serem abordados no projeto	12/03/2007	27/03/2007
Elaboração do plano de trabalho	22/03/2007	12/04/2007
Preparação de apresentação do plano de trabalho	17/04/2007	11/05/2007
Levantamento/avaliação de ponteiros para interação com a projeção	17/04/2007	11/05/2007
Pesquisa de métodos para processamento de imagens	14/05/2007	28/05/2007
Pesquisa sobre interação com o sist. Operacional	14/05/2007	19/05/2007
Definição de metodologia de avaliação da ferramenta	21/05/2007	25/05/2007
Pesquisa sobre rotinas para adaptação	28/05/2007	01/06/2007
Pesquisa sobre *mouse gestures*	04/06/2007	08/06/2007
Elaboração de relatório final	14/05/2007	15/06/2007
Preparação de apresentação final	11/06/2007	25/06/2007

Tabela 6: Cronograma do período implementação

Tarefa	Início	Término
Implementação de máquina de estados Manipulação de câmera Geração e apresentação de imagens	23/07/2007	10/08/2007
Mapeamento de coordenadas com base em homografia	13/08/2007	24/08/2007
Implementação da passagem de eventos de mouse e comandos ao sistema operacional.	27/08/2007	31/08/2007
Preparação de relatório técnico inicial	20/08/2007	06/09/2007
Implementação de identificação de movimentos específicos do mouse (*mouse gestures*)	16/09/2007	23/09/2007
Implementação da passagem de eventos de teclado ao sistema operacional.	24/09/2007	03/10/2007
Preparação de apresentação para avaliação intermediária	08/10/2007	21/10/2007
Implementação de aplicação-conceito	22/10/2007	29/10/2007
Desenvolvimento de ponteiros	22/10/2007	29/10/2007
Identificação da movimentação de ponteiros	22/10/2007	04/11/2007
Identificação de eventos do ponteiro	04/11/2007	11/11/2007
Elaboração de relatório final	01/11/2007	16/11/2007
Preparação de apresentação e pôster	10/11/2007	03/12//2007
Preparação de demonstração	24/11/2007	07/12/2007

8 BIBLIOGRAFIA

AZUMA, R. T. (1993) “Tracking Requirements for Augmented Reality”, **Communications of the ACM, 36(7):50-51, July.**

BOUKREEV, K. **Mouse Gestures Recognition.** 2001. Disponível em <http://www.codeguru.com/Cpp/misc/misc/mouseandcursorhandling/article.php/c3819/ > Acesso em 22 de maio de 2007.
UMBAUGH, Scott E. **Computer Vision and image processing :** a practical approach using CVIPtools, 1a ed., Indianapolis, Prentice Hall, jun 1999, 505p.

COELHO, M. C. F. S. P. ; TAVARES, J. M. R. S., **Método de Calibração de Câmaras proposto por Zhang** [Porto] **:** Laboratório de Óptica e Mecânica Experimental - Faculdade de Engenharia da Universidade do Porto, 2003, 11p. Disponível em <http://paginas.fe.up.pt/~tavares/downloads/publications/relatorios/Relatorio_zhang.pdf>. Acesso em 25 de maio de 2007.

CPLUSPLUS. **Cstdlib (stdlib.h)**. Disponível em <http://www.cplusplus.com/reference/clibrary/cstdlib/.>. Acesso em 11 de junho de 2007.

MIRANDA, F. R et al**. ARHockey: Um Jogo em Realidade Aumentada Baseada em Projetores.** In: **Simpósio Brasileiro de Jogos de Computador e Entretenimento Digital**, 2006, Recife, **Anais...** Recife, 2006 Disponível em <http://www.sbc.org.br/bibliotecadigital/download.php?paper=530> . Acesso em 30 de abril de 2007.

MORGAN, Drew. **Installing OpenCV Under Windows.** Disponível em http://www.comp.leeds.ac.uk/vision/opencv/install-win.html. Acesso em 15 de maio de 2007.

MSDN. **How to: Simulate Mouse and Keyboard Events in Code.** Disponível em http://msdn2.microsoft.com/en-us/library/ms171548.aspx. Acesso em 06 de junho de 2007.

MSDN. **Mouse Event Function.** Disponível em http://msdn2.microsoft.com/en-us/library/ms171548.aspx. Acesso em 06 de junho de 2007.

OPENCV. Disponibiliza download da biblioteca e links relevantes. Disponível em http://sourceforge.net/projects/opencvlibrary/. Acesso em 02 de abril de 2007

SCHENEIDER. **Lousas Interativas: Smart Technologies.** Disponível em: <http://www.scheiner.com.br/produtos_sb.php?id=66&op=descricao>. Acesso em 9 de abril de 2007.

UMBAUGH, Scott E. **Computer Vision and image processing :** a practical approach using CVIPtools, 1a ed., Indianapolis, Prentice Hall, jun 1999, 505p.

VEZHNEVETS, V., VELIZHEV, A. **GML C++ Camera Calibration Toolbox**.Disponível em <http://research.graphicon.ru/calibration/gml-c++-camera-calibration-toolbox.html>. Acesso em 21 de maio de 2007.

WOODS, E. et al. **MagicMouse: an Inexpensive 6-Degree-of-Freedom Mouse**. Proceedings of Graphite 2003, 2003, Melbourne. Disponível em <http://www.hitl.washington.edu/artoolkit/Papers/2003-Graphite-MagicMouse-Inexpensive6DOF.pdf>. Acesso em 15 de maio de 2007.