UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS DE QUIXADÁ
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

PAULO FILIPE DOS SANTOS DANTAS

LOUSA DIGITAL BASEADA EM VISÃO COMPUTACIONAL

QUIXADÁ

2015

PAULO FILIPE DOS SANTOS DANTAS

LOUSA DIGITAL BASEADA EM VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Bacharelado em Sistemas de Informação da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: computação

Orientador Prof. Dr. Críston Pereira de Souza

QUIXADÁ

2015

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca do Campus de Quixadá

D213l Dantas, Paulo Filipe dos Santos

Lousa digital baseada em visão computacional / Paulo Filipe dos Santos Dantas.– 2015.

56 f. : il. color., enc. ; 30 cm.

Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Bacharelado em Sistemas de Informação, Quixadá, 2015.

Orientação: Prof. Dr. Críston Pereira de Souza

Área de concentração: Computação

1. Visão por computador 2. Ferramentas de aprendizagem 3. Algoritmos I. Título.

CDD 004

PAULO FILIPE DOS SANTOS DANTAS

LOUSA DIGITAL BASEADA EM VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Graduação em Sistemas de Informação da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: computação

Aprovado em: _____ / junho/ 2015.

BANCA EXAMINADORA

_____
Prof. Dr. Críston Pereira de Sousa (Orientador)
Universidade Federal do Ceará – UFC

_____
Prof. Dra. Andreia Libório Sampaio
Universidade Federal do Ceará – UFC

_____
Prof. Dr. Rubens Fernandes Nunes
Universidade Estadual do Ceará – UFC

Aos meus pais...

AGRADECIMENTOS

Agradeço a Deus por ter me dado saúde e força para superar as dificuldades e por ter me proporcionado mais esta conquista.

Agradeço a toda a minha família, especialmente a minha mãe Ivanilda, meu pai Francisco, por sempre acreditarem em mim e me apoiarem em todos os momentos de minha vida.

Agradeço ao meu orientador, Prof. Dr. Críston Pereira de Sousa, pela paciência, incentivos, ensinos e dedicação, neste trabalho, nos grupos de estudo e na minha vida acadêmica como um todo. Agradeço também a oportunidade de participar do grupo de pesquisa em visão computacional de onde se originou este trabalho e de onde levo amizades.

Ao Prof. Dr. Davi Romero, meu orientador durante quase dois anos no grupo PET, por seu incentivo, apoio, paciência e ensinamentos, que serão levados comigo para toda vida.

Ao Prof. Msc. Wladimir Araújo Tavares por ter contribuído no meu aprendizado de programação durante as reuniões do Grupo de Estudos para Maratona de Programação. Pela amizade, apoio e incentivo.

À professora Tânia pelo acompanhamento e sugestões que deu a esse trabalho. Foram auxílios valiosos.

Agradeço aos Professores Camilo Almendra, Carla Ilane, Victor Almeida, Jefferson de Carvalho, Arthur Callado, Samy Soares, Aragão, Atslands rocha, Diego Andrade, Flávio Rubens, Ricardo Reis, Jeandro Bezerra, Alberto Sampaio, que contribuíram e participaram ativamente em minha formação acadêmica.

À minha namorada, Marisângela, pelo incentivo, apoio e compreensão em todos os momentos.

Em nome de Rafael Cavalcante, Tiago Lobo, Samuel Freitas, Geovanny Filho, Emerson Fernandes, Suellem Cavalcante, Ciro Dantas, Marcelo Barbosa, Paulo Ivo, Mardson Ferreira, Felipe Freitas, Italo Pessoa, Danilo Rodrigues, Nicloas Monte, Araújo Filho, Otávio Araujo, Wellyngton Amaral, Tiago Montezuma, Tiberio Macedo, Vinicius Freitas, Rafael Ramalho, Junior Fernandes, William Rosseno, Daniel Wolney, Zarathon Maia, Lina Raylly, Junior Leonel, Salomão Sales, Alex Oliveira, Leonara Braz, Jefson Ferreira, Linesterfeson Sampaio, Esterfeson Sampaio, agradeço a todos os meus amigos, essenciais em todas as etapas da minha vida.

À banca avaliadora, por prontamente aceitar fazer parte desse momento tão especial. Aos companheiros de trabalho, funcionários e desconhecidos que contribuíram sem perceber para que esse sonho se realizasse.

RESUMO

Quadros brancos têm o seu uso bem difundido em todos os tipos de ambientes voltados para estudo e ensino, porém, estes quadros possuem algumas limitações. Para solucionar estas limitações muitos sistemas de lousa digital estão sendo desenvolvidos e lançados no mercado. Alguns destes sistemas possuem superfícies especiais capazes de identificar as interações do usuário e, devido a estas superfícies, estes sistemas possuem preço elevado, o que impede que algumas instituições adquiriram tais ferramentas. De forma a contornar o obstáculo do preço, alguns sistemas de lousas digitais de baixo custo foram desenvolvidos. Entretanto, a maioria destes sistemas utiliza infravermelho. Neste trabalho desenvolvemos uma lousa digital utilizando algoritmos e técnicas de visão computacional – para identificar as ações do usuário – em conjunto com a API padrão do Windows, utilizada para realizar as ações do usuário. Em nosso projeto utilizamos apenas os equipamentos já presentes na maioria dos ambientes de estudo e ensino: quadro branco, projetor, notebook e a webcam integrada ao notebook. Isso nos permitiu descartar o uso de qualquer equipamentos adicionais, como infra vermelho.

Palavras-chave: Lousa digital. Visão computacional. Análise e remoção de fundo. OpenCV.

ABSTRACT

The whiteboard which use is very known into all environments focused to studying and teaching, however, these frameworks have some limitations. To solve them, many digital whiteboard systems have been developed and launched on the market. Some of them have special surfaces that are able to identify the user's interactions and because of this, they have a very high price, preventing that some institutions acquired such tools. Trying to solve this problem, some low price whiteboards digital systems were developed. However, most of them use infrared. In this paper, we have developed a digital whiteboard using algorithms and techniques of computer vision – to indentify the user actions – together with the Windows standard API, used to perform the user's actions. At our project we used only the equipments already present in most of studying and teaching environments: whiteboard, projector, notebook and built- in webcam to the notebook. This allowed us discard the use of any additional equipment, as infrared.

Keywords: Digital Whiteboard. Computer Vision. Background analysis and subtraction. OpenCV.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Este projeto consiste na construção de uma lousa digital de baixo custo por meio de técnicas de visão computacional. Alguns projetos encontrados na literatura usam emissor e/ou sensor infravermelho, a exemplo de *LoCoBoard* (SOARES, 2009) e *PlayAnywhere* (WILSON, 2005). No presente projeto construímos uma lousa digital sem a necessidade de equipamentos adicionais como emissor e sensor de infravermelho. Para isso usamos apenas quadro branco, uma webcam comum e um projetor.

Quadros brancos existem na maioria das universidades e escolas. Seu uso é muito frequente e muitas vezes indispensável em vários tipos de reuniões, como sessões de *brainstorm*, palestras, salas de aula etc.

> Pois oferece um grande espaço comum para os participantes concentrarem sua atenção e expressar suas idéias de forma espontânea. Não é apenas eficaz, mas também econômico e fácil de usar – tudo que você precisa é uma placa plana e várias canetas de tinta seca (ZHANG, 2005, p 1 tradução nossa).

> O sucesso é atribuível a duas qualidades principais: facilidade de uso e rapidez. Somos ensinados a desenhar com uma caneta em uma idade muito precoce, e para que todos saibam como escrever, desenhar e apagar em qualquer quadro branco. Além disso, usar o quadro é rápido e fluido: não há tempo de ativação (se não apenas remover a tampa de uma caneta) (BERARD, 2003, p. 1 tradução nossa).

Nestas ocasiões, principalmente nas aulas, usam-se frequentemente o quadro branco e um projetor de vídeo, que auxiliam na apresentação de conteúdos digitais que estão armazenados em computadores pessoais.

No entanto, existem algumas limitações no uso dos quadros brancos. Dado que o quadro tenha sido usado (escrito e/ou desenhado) começam os problemas: o palestrante/professor tem que parar a sua apresentação e esperar que os ouvintes copiem todo ou uma parte do que está no quadro. O fato de interromper a reunião/aula para que os ouvintes tomem notas do que foi escrito no quadro, além de ser dispendioso, interrompe a fluidez de qualquer aula/apresentação. Costumamos dizer que "corta o raciocínio" do que está sendo apresentado. Outro problema é quando o quadro está cheio e devemos apagar o que foi escrito para conseguirmos mais espaço e continuarmos a escrever. Quando optamos por apagar o que está no quadro é difícil escolher o que apagar porque cada parte é considerada importante. Além disso, o conteúdo que foi produzido durante a reunião é difícil de compartilhar com quem não está presente. Embora em alguns casos sejam feitas videoconferências para solucionar o problema de compartilhamento, muitas vezes os participantes remotos não conseguem ver o que é feito nos quadros brancos (BERARD, 2003; ZHANG, 2005).

A fim de solucionar alguns dos problemas que podemos encontrar em reuniões com quadros brancos comuns, muitos sistemas de lousas digitais interativas foram lançados recentemente, tais como a *Smartskin* (REKIMOTO, 2002) e *CollaBoard* (KUNZ; NESCHER; KUCHLER, 2010), que usam telas especiais e alguns equipamentos que permitem detectar o toque na tela. Apesar destes sistemas oferecerem uma boa experiência ao usuário, surgem outros problemas tais como o preço e o cuidado especial que equipamentos deste tipo exigem. O preço deste tipo de equipamento é muito alto, e nem todas as instituições podem arcar com estes custos. As telas são elementos chave deste tipo de sistemas. Frequentemente ocupam muito espaço e requerem um cuidado especial. Se de alguma forma algo danificar a superfície, o sistema todo ficará inutilizado devido à dependência da superfície.

Outros tipos de sistemas surgiram para tentar contornar o obstáculo do preço. Estes sistemas usam fortemente técnicas de visão computacional e poucos hardwares adicionais. Alguns usam canetas infravermelho e um filtro acoplado à câmera, como *LocoBoard* (SOARES, 2009) que permite o controle total do hardware através de uma caneta que emite luz infravermelho. Através do filtro adicionado à câmera é possível identificar com precisão o local onde a caneta está e realizam os comandos com base nisso. Outro exemplo interessante é o *PlayAnywhere* (WILSON, 2005) que permite interação sem o uso de canetas especiais. No entanto, também usa o filtro infravermelho. A técnica usada nessa aplicação é analisar a sombra das mãos do usuário.

Embora o custo da utilização do infravermelho seja baixo, o objetivo principal deste trabalho é construir uma lousa digital sem equipamentos adicionais, utilizando técnicas de visão computacional. Com isso reduzimos ainda mais os custos de uma lousa digital usando apenas os equipamentos já presentes na maioria das universidades: quadro branco, projetor, notebook e a webcam integrada ao notebook.

Na Seção 2 é apresentada uma revisão bibliográfica, onde abordamos os conceitos chave deste trabalho e trabalhos relacionados. Na Seção 3 é detalhado o uso do software proposto, onde falamos sobre suas funcionalidades, restrições e características. Na Seção 4 descrevemos parte do processo de desenvolvimento, onde explicamos o uso dos algoritmos e as dificuldades que nos deparamos no decorrer da implementação do software, bem como a resolução de alguns destes problemas e sugestões de como proceder com problemas que não foram resolvidos. Por fim a Seção 5 traz as considerações finais e sugestões de trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

Nas seções a seguir apresentamos os conceitos chave relacionados a este trabalho. Iniciamos com uma revisão sobre o conceito de Visão computacional (Seção 2.1). Logo após apresentamos a biblioteca OpenCV (Seção 2.2), que contem as implementações dos algoritmos de processamento de imagens mais comuns o que nos deu apoio no desenvolvimento do software. Na Seção 2.3 apresentamos alguns trabalhos relacionados, vale ressaltar que não foram encontrados trabalhos que tenham a mesma proposta do nosso trabalho encontramos apenas o trabalho de ZHANG (2005) em que a proposta de um de seus módulos se assemelha a nossa proposta. Maior parte dos trabalhos encontrados propõe a criação de lousas digitais utilizando infravermelho ou superfícies especiais

2.1 Visão computacional

Basu e Li (1993) definem o objetivo de estudo da área de visão computacional como: "obter informações sobre o ambiente através da captura e processamento de imagens" (p 14 tradução nossa). No entanto, é necessário definir a forma como o computador deve capturar essas informações e, além disso, fazer com que o mesmo simule uma parte do cérebro, o pensamento humano, sendo esse um processo complicado de se desenvolver, visto que o sistema de visão humano é bastante complexo.

O sistema visual humano reconhece facilmente os objetos que são capturados por ele no que tange a formas, cores, profundidade e dimensões. Somos capazes de identificar objetos com certa precisão apenas com o olhar. Quando olhamos para um objeto qualquer podemos facilmente dizer sua(s) cor(es), seu formato, seu tamanho, informar se há luz incidindo sobre ele, identificar a sombra gerada ou se há alguma sombra incidindo sobre ele. Podemos olhar para fotos e identificar pessoas facilmente. Vale ainda ressaltar que essa capacidade de reconhecimento das características dos objetos e das imagens que vemos vai além do nosso sistema de visão, tendo origem na nossa capacidade de guardar características de objetos em nossa memória. Em outras palavras, nós possuímos em nosso cérebro uma "base de dados" e uma forma fácil de realizar comparações e inferências sobre objetos observados.

Em Marques Filho, Vieira e Neto (1999), é apresentada uma análise mais detalhada do olho humano e de como percebemos as cores e todas as outras características dos objetos ao nosso redor, bem como as limitações do aparelho visual humano. Também é feita uma comparação entre o sistema visual humano e um sistema visual artificial. Os sistemas

utilizados nesta comparação são apenas de captura, ou seja, uma comparação entre a capacidade humana de captação de objetos e características (o que o olho consegue transmitir para o cérebro) e o que a câmera consegue transmitir para o computador. No entanto, não só as câmeras podem servir de entrada de imagens, aparelhos como ultrassom e sensores de calor, por exemplo, podem retornar uma saída visual em forma de imagem para os dispositivos e computadores. É válido enfatizar que nenhum dos dois sistemas é formado apenas pelos seus 'subsistemas' de captura.

No sistema de visão humana o olho realiza a captura das características do ambiente. Essas informações são enviadas ao cérebro, que as interpreta através de um sofisticado processo de cognição (AMCHAM BRASIL BELO HORIZONTE, [200-?]). Já em sistemas de visão computacional, o dispositivo de captura, que comumente é a câmera, é o responsável pelo reconhecimento e envio das imagens captadas ao computador que, por sua vez, tentará simular o cérebro humano, completando assim os itens básicos de um sistema de visão computacional.

Em Conci, Azevedo e Leta (2008), visão computacional é considerada uma subárea da computação gráfica, a qual se relaciona com a análise de imagens digitais. Ainda segundo estes autores: "Esta área trata da extração de informações das imagens e da identificação e classificação de objetos presentes nessa imagem" (CONCI, AZEVEDO e LETA, 2005. P. 5), que tem o propósito de ensinar o computador a reconhecer os objetos que estão ao alcance de seu campo de visão e dar apoio às tomadas de decisão de um programa. Esta característica é aplicada a diversas áreas, tais como medicina, realidade aumentada, reconhecimento ótico de caracteres (*Optical character recognition – OCR*), visão de máquina, dentre outras (RIOS, 2010).

2.2 OpenCV

OpenCV (*Open Computer Vision*) é uma biblioteca de código aberto com mais de 2500 algoritmos otimizados tanto para visão computacional quanto para aprendizagem de máquina. Os algoritmos variam desde tratamentos simples de imagens (transformação de colorido para cinza, filtros de otimização de cores etc.) até algoritmos mais complexos como os de reconstrução 3D, reconhecimento de faces, rastreamento de objetos, entre outros. A biblioteca OpenCV está disponível para as plataformas *Windows, android, ubuntu* e *ios*. Tem suporte às linguagens C, C++, Pyton e recentemente JAVA. No entanto, é escrito nativamente em C++ (OPENCV, 2013).

Foi originalmente desenvolvido pela Intel em 1999. No ano de 2006, foi lançada uma versão beta. Em 2009, foi lançada a segunda versão que trazia importantes mudanças, tais como suporte a C++ (LAGANIÈRE, 2010). Embora a versão mais recente seja a 3.0.0, a versão utilizada no desenvolvimento deste projeto é a 2.4.10 devido a problemas de configuração de máquina para uso das versões mais novas e a facilidade da integração desta versão com a ferramenta de desenvolvimento da Microsoft, o Microsoft Visual Studio Professional 2014.

Algumas das vantagens do uso de OpenCV em projetos de visão computacional é o fato de os algoritmos mais usados já estarem implementados, otimizados, e seu uso é realizado por chamadas simples de funções C++. Além disso, existe uma comunidade de desenvolvedores crescente e grandes empresas utilizando-a tais como Google, Yahoo, Intel, IBM, Sony, Honda, Toyota (OPENCV, 2013). Alguns dos algoritmos usados neste trabalho são explicados a seguir.

2.2.1 Fast Feature Detector

O algoritmo *Fast Feature Detector* é usado para detectar de forma rápida pontos de interesse em uma imagem. Pontos de interesse são usados em muitos algoritmos tais como reconhecimento de objetos, reconstrução 3D, rastreamento, correspondência entre imagens, entre outros (ROSTEN; DRUMMOND, 2006).

O algoritmo de detecção de pontos em imagens de Harris e Stephens (1988), o *Harris Detector*, é um dos mais conhecidos na literatura. Em LAGANIÈRE (2010) temos uma definição mais detalhada da forma como é calculado um ponto de interesse (*harris corner*). De forma geral, no algoritmo *Harris Detector*, um canto (ponto de interesse) é definido pela intensidade de mudança na derivada de um pixel. Essa derivada é calculada tomando como base uma certa quantidade dos pixels ao redor do pixel examinado.

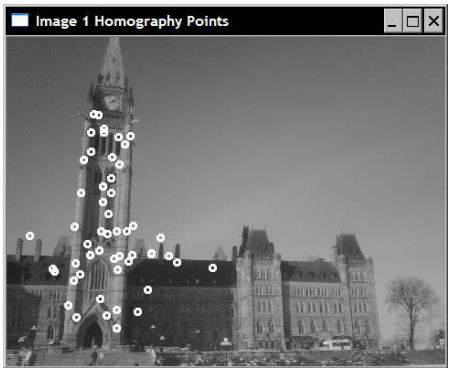
Embora seja eficiente, o algoritmo *Harris Detector* exige o cálculo das derivadas da imagem e é uma operação que exige muito do computador (LAGANIÈRE, 2010). O algoritmo *FAST* (ROSTEN; DRUMMOND. 2006), é usado para reconhecimento de cantos em imagens de forma rápida e menos custosa em termos de processamento. O algoritmo *FAST* se tornou um dos mais usados. O algoritmo *FAST* também leva em conta em sua análise os pixels que se encontram ao redor do pixel a ser examinado. Para o algoritmo FAST, um canto em uma imagem, ou ponto chave, é definido quando “um arco de pontos contíguos de comprimento superior a 3/4 do perímetro circular que se encontra em todos os pixels diferem

significativamente a intensidade do ponto central, em seguida, um ponto-chave é declarado". (LAGANIÈRE, 2010, p. 204, tradução nossa).

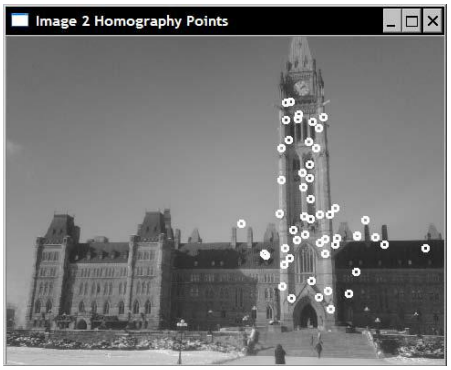
2.2.2 Correção de perspectiva

Como não especificamos uma posição exata para a câmera, teremos o problema da distorção de perspectiva. Para resolver esse problema, realizamos uma transformação de perspectiva. Para transformação de perspectiva precisamos de dois elementos: a matriz de homografia e os pontos de transformação (tanto os pontos iniciais, do plano distorcido, quanto os finais, do plano sem distorção). Os pontos são obtidos automaticamente através do algoritmo FAST, explicado na seção anterior (2.2.1).

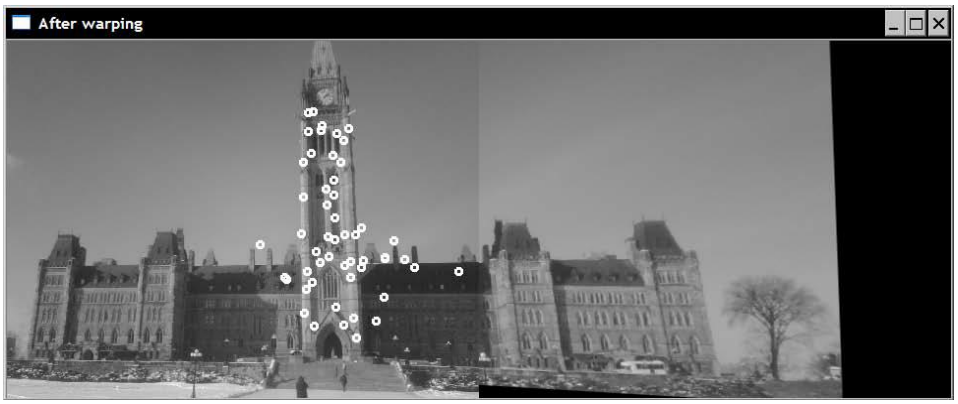
Figura 1 – Exemplo de junção de duas imagens com dois pontos de vista de uma mesma cena, separadas por uma rotação.



(a) – imagem 1.



(b) – imagem 2.



(c)- Imagem resultante após a junção
Fonte: (LAGANIÈRE, 2010).

A matriz de homografia é uma matriz 3x3 que pode ser usada tanto para correção de perspectiva quanto para encontrar a correspondência entre duas imagens de uma mesma cena, que podem ser combinadas para formar uma imagem maior, como pode ser visto na Figura 1. Os pontos que estão marcados como círculos brancos nas Figuras 1 (a) e 1 (b) são os pontos chaves que as duas imagens têm em comum (LAGANIÈRE, 2010).

A matriz de homografia pode ser obtida através da função do *OpenCV findHomography* (Opencv Reference Manual, 2.3). Essa matriz é usada pelo algoritmo *warpPerspective* (Opencv Reference Manual, 2.3), da biblioteca *OpenCV*, para calcular a transformação de perspectiva. O algoritmo *findHomography* pode ser chamado passando até cinco parâmetros, no entanto apenas os dois primeiros parâmetros são obrigatórios:

- Um vetor de pontos que representam as coordenadas do plano original (Imagem distorcida).
- Um vetor de pontos que representam as coordenadas do plano de destino, após a correção (imagem não distorcida).
- Método – Método usado para computar a matriz de homografia. Possíveis métodos:
    - 0 – um método comum usando todos os pontos
    - CV_RANSAC – método robusto baseado no RANSAC
    - CV_LMEDS – método robusto de Menor Mediana
- ransacReprojThreshold: Erro máximo permitido reprojeção para tratar um par de pontos
- Uma máscara opcional de saída.

Após a obtenção da matriz de homografia aplica-se o algoritmo *warpPerspective* que possui até 7 parâmetros, embora apenas quatro deles sejam obrigatórios: a imagem de entrada, a imagem de saída, a matriz de homografia e o tamanho da imagem de saída.

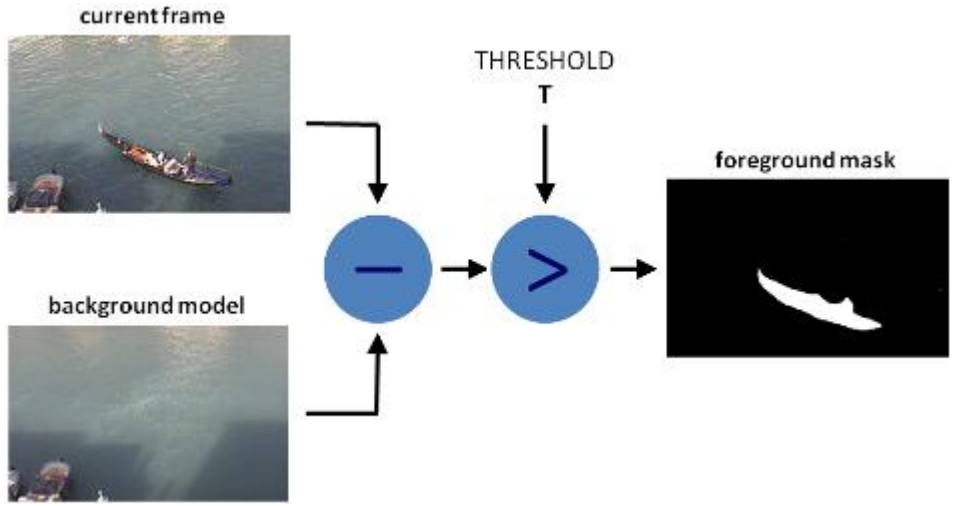
2.2.3 Background subtractor (MOG)

Em visão computacional, subtração de fundo é um passo importante para muitos tipos de aplicações, tais como vigilância visual, interface homem-máquina, aplicações com baixa largura de banda, rastreamento e detecção de movimento em câmeras CFTV (Circuito Fechado Televisão) (Nascimento, 2011) (KAEWTRAKULPONG; BOWDEN, 2002). O algoritmo *Background Subtractor* (MOG) da biblioteca OpenCV tem por objetivo realizar a subtração do fundo de vídeos (segmentação em tempo real).

O algoritmo MOG realiza a subtração do fundo tendo como base uma imagem de referência (ou modelo de fundo), que é calculada ao longo da cena. Para cada novo frame do vídeo, o fundo é subtraído usando a imagem de referência e, após a subtração, é aplicada uma limiarização na imagem (KAEWTRAKULPONG; BOWDEN, 2002). O resultado do processamento é uma imagem binária, em preto e branco, que contém os pixels em cor branca

que representam mudanças na cena. Uma representação geral do funcionamento do algoritmo pode ser visto na Figura 2. O resultado do processamento pode ser visto na imagem *foreground mask*. Na Figura 2, temos duas imagens que serão comparadas/subtraídas, onde Currentframe é o frame atual do vídeo e background model é a imagem que representa o fundo da cena aprendido pelo algoritmo. O THRESHOLD é a limiarização da imagem, que a transforma em uma imagem binária (apenas preto e branco).

Figura 2 – Funcionamento do algoritmo de subtração de fundo.



Fonte: OpenCV Documentation (2013)

2.3 Trabalhos relacionados

Muitas são as propostas de lousas digitais (*whiteboard*). Algumas utilizam superfícies especiais, tais como a *SmartSkin* (REKIMOTO, 2002), a qual usa uma superfície touch com sensores capazes de identificar com precisão o local do toque da mão na superfície usada. A superfície é feita com vários sensores de eletricidade, e quando o usuário toca na superfície esses sensores identificam a posição do toque. Assim como em nossa proposta, este usa um projetor para exibir as informações na mesa. No entanto, este permite múltiplos toques e múltiplos usuários, enquanto no nosso caso permitimos apenas um usuário e um toque. Por outro lado, nossa abordagem não utiliza hardware especial.

Também na mesma linha de lousas com superfícies especiais temos a *CollaBoard* (KUNZ; NESCHER; KÜCHLER, 2010), que utiliza um display de LCD e um módulo interativo vendido comercialmente, o qual permite interação com a tela através do uso de canetas. Da mesma forma, eles também utilizam câmera, embora esta câmera seja usada apenas para transmitir a imagem do utilizador para uma segunda lousa, ou para uma projeção

de vídeo qualquer. Outro exemplo de lousa digital que também usa uma superfície especial é a PN-L602B (SHARP, 2013), cuja superfície de apresentação/interação possui custo elevado.

Em contrapartida aos projetos que usam superfícies especiais, há projetos que propõem transformar a superfície do quadro branco comum, ou qualquer outra superfície tais como mesas, paredes e até mesmo o piso, em um dispositivo de saída e interação. Estes projetos costumam ser mais baratos, pois geralmente usam câmeras, projetores, computadores e quadros brancos, e alguns também usam infravermelho e iluminador infravermelho. A maioria destes itens, tais como projetor, quadro branco e computadores, já são comuns em universidades e ambientes de ensino em geral. Projetos deste tipo funcionam de forma parecida, sendo distinguidos entre os que usam e os que não usam infravermelho.

Um exemplo de projeto que usa infravermelho é a *PlayAnywhere: A Compact Interactive Tabletop Projection-Vision System* (WILSON, 2005), que tem o propósito de transformar qualquer superfície plana em uma tela, e que permite a interação através de múltiplos toques e rastreamento de objetos. Para isso, um dispositivo foi montado usando um projetor, uma câmera com filtro infravermelho e um iluminador infravermelho. O filtro infravermelho faz com que a câmera capture apenas o que não reflete a luz do iluminador infravermelho, como sombras e objetos que não tenham a capacidade de refletir a luz, tais como marcadores. A sombra é a característica que é usada para identificar o toque.

Ao pôr a mão na superfície é gerada uma sombra que é extraída da imagem e quando essa sombra tem certa espessura, o toque na superfície é acionado. De forma semelhante os marcadores, placas de plástico de 1,4″ de diâmetro com uma figura impressa em cor preta colada na parte superior, são identificados na cena, pois possuem pontos característicos que não refletem a luz infravermelha. A *PlayAnywhere* também é capaz de rastrear folhas de papel e projetar objetos virtuais para que o usuário possa manipulá-los em tempo real, transformando-a em um marcador de realidade aumentada.

Outro projeto com infravermelho é a *LoCoBoard: Low-Cost Interactive Whiteboard Using Computer Vision Algorithms* (SOARES, 2009). Este projeto usa projetor, câmera, computador, filtro infravermelho para a câmera e uma caneta que emite luz infravermelha. O fato de se usar infravermelho torna o trabalho simples e eficiente, pois quando o usuário quer realizar alguma interação com o computador, como um clique no mouse, apenas se aciona a caneta que emite a luz infravermelha e a câmera captura o ponto, pois com o filtro adicionado à câmera apenas luz infravermelha é capturada. Todas as outras informações que um vídeo normal teria como a imagem do usuário e a imagem do que é

projetado na parede, bem como todos os outros ruídos da cena, são desconsiderados, pois estes não emitem luz infravermelha.

Semelhante ao que construímos no presente trabalho, a *PlayAnywhere* e a *LoCoBoard* permitem o controle do computador através da interação do usuário com a superfície, no entanto não será permitido o rastreamento de objetos como é feito na *PlayAnywhere.* No presente trabalho usamos apenas a webcam que já vem nos notebooks, ou uma webcam simples de baixa resolução acoplada ao notebook ou PC. Por outro lado, a *PlayAnywhere* e a *LoCoBoard* usam filtro e iluminadores infravermelho. Assim como na *LoCoBoard realizamos* o controle do hardware, o mouse mais especificamente, e a sua aplicação ao uso de projetores comuns que são facilmente encontrados em ambientes de ensino.

Também existem os projetos que não usam superfícies especiais, e que não se beneficiam do artifício do uso do infravermelho. No entanto, em nossa pesquisa a maioria destes se restringe a apenas compartilhar o que foi escrito e/ou projetado em reuniões e em aulas. Um exemplo deste tipo de lousas digitais é visto em ZHANG (2005), descrito na seção a seguir.

2.3.1 ZHANG (2005)

ZHANG (2005) aborda o problema das lousas digitais com um software robusto composto de quatro módulos:

- *"Whiteboard Scanning and Image Enhancement"* (Digitalização de quadro branco e aprimoramento de imagem);
- *"Real-time Whiteboard Processing and Collaboration"* (Processamento e colaboração de quadro branco em tempo real);
- *"Whiteboard Archiving"* (Arquivamento do quadro branco);
- *"A Projector-Whiteboard-Camera System"* (Um sistema de Projetor-quadro branco e câmera).

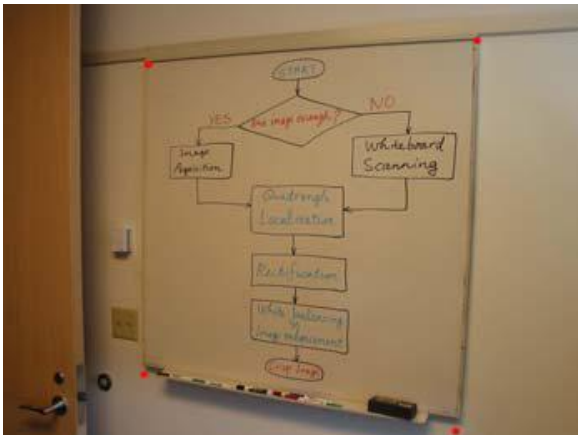
A seguir será descrito cada um destes módulos.

2.3.1.1 Digitalização de quadro branco e aprimoramento de imagem

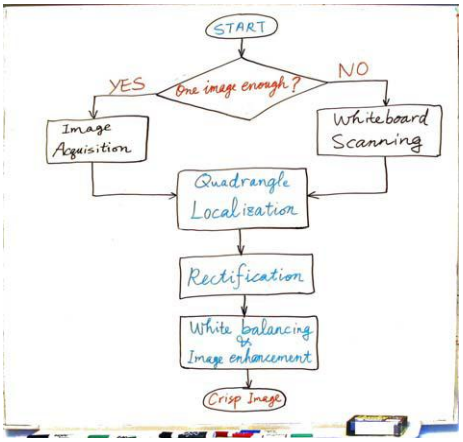
O primeiro e mais simples é o modulo de digitalização do quadro branco. Através de imagens do quadro capturadas por uma câmera digital é gerado uma nova imagem com apenas o que foi escrito na lousa, como ilustrado na Figura 3. Para isso são identificados os

quatro cantos do quadro (Figura 3 (a)) e eliminado o que não representa interesse na imagem, ou seja, o que está fora do quadrado formado pelos cantos da lousa. Logo após é realizada uma correção de perspectiva, tomando como base para isso os cantos do quadro.

Figura 3– Exemplo de entrada e saída do módulo de digitalização do quadro branco com uma imagem



(a) imagem capturada da câmera.



(b) Imagem após processamento

Fonte: Zhang (2005)

O último passo para se chegar ao resultado final do processamento (Figura 3 (b)) é realizar um balanceamento da cor de fundo branca da imagem original para o branco real. Isto envolve dois procedimentos. O primeiro é estimar a cor do quadro, visto que não é um branco ‘puro’ devido a vários fatores como iluminação e reflexo. E o segundo é a transformação do branco do quadro (que possui seus ruídos e variações) em branco real, e o aumento da saturação das cores do que foi escrito no quadro, resultando em uma imagem mais nítida.

Além destes processos, este módulo também permite realizar o escaneamento de quadros de tamanho maior, sendo necessário coletar mais de uma fotografia. Esse processo é permitido através da matriz de homografia entre as imagens. O resultado pode ser visto na Figura 4. Na Figura 4 (a) temos as imagens de entrada, na Figura 4 (b) temos a imagem montada a partir das três primeiras (Figura 4 (a)), e na Figura 4 (c) temos o resultado do processamento explicado anteriormente.

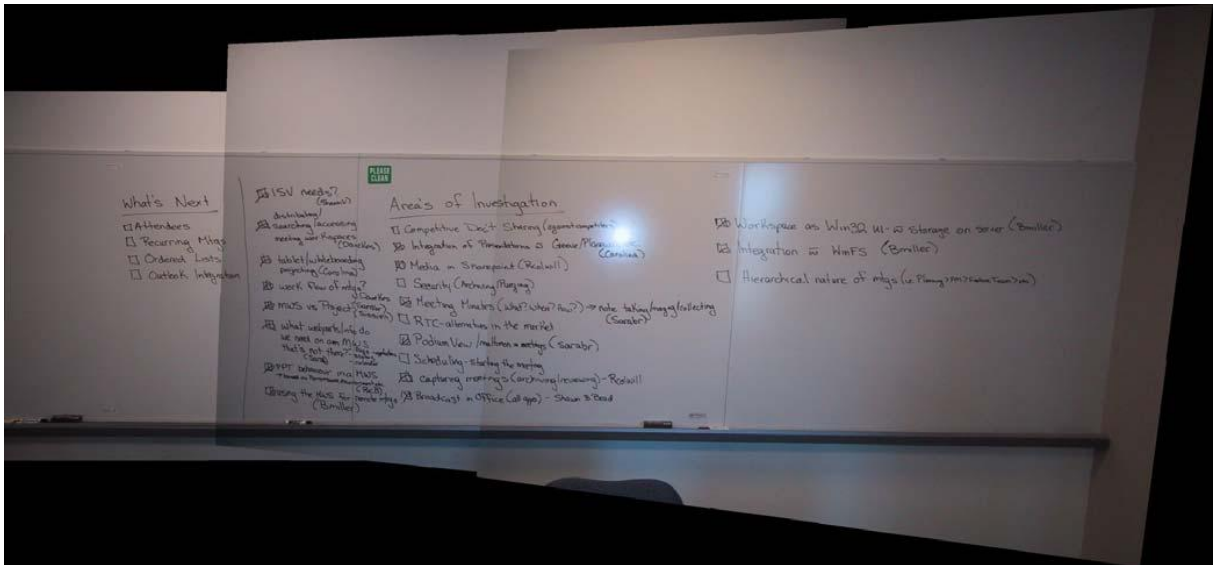
2.3.1.2 Processamento e colaboração de quadro branco em tempo real

O segundo modulo do projeto de Zhang (2005) é o módulo de apoio a reuniões em tempo real. A proposta é que em reuniões que tenham participantes em teleconferência o conteúdo do quadro seja capturado por uma câmera, seja processado e posteriormente enviado para os participantes, e o conteúdo do quadro que os participantes remotos veem seja montado em suas telas. É feita uma comunicação com os usuários remotos de maneira que eles possam

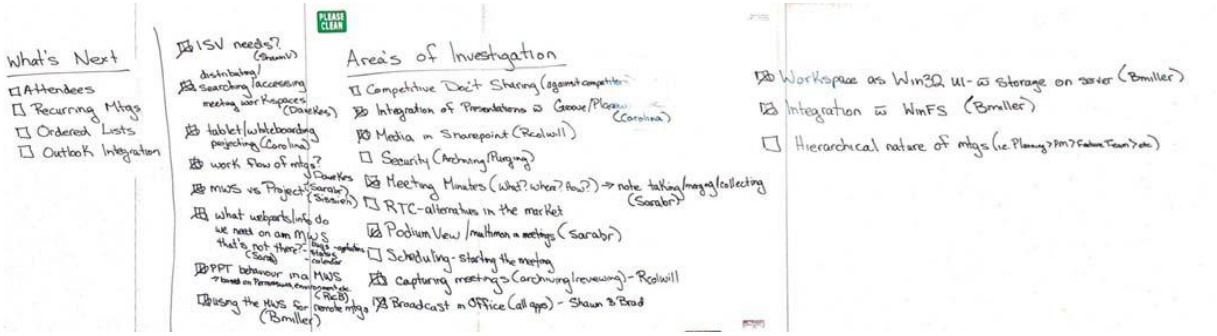
interagir com o conteúdo do quadro, seja escrevendo anotações ou inserindo desenhos conforme sua necessidade.

Figura 4 – Exemplo de entrada e saída do módulo de digitalização do quadro branco com várias imagens.

(a) Imagens originais



(b) Imagens unidas pela matriz de homografia



(c) Resultado final do processamento da imagem (b)

Fonte: Zhang (2005)

2.3.1.3 Arquivamento do quadro branco

O terceiro módulo não permite a interação do usuário remoto, e se preocupa em registrar a reunião de forma que seja possível navegação no vídeo gravado. A navegação se dá de duas maneiras: 1 – identificação dos quadros chaves, que são frames no vídeo que contêm

todo o conteúdo de um determinado momento da reunião ou conferência; 2 – indexação visual em que são fornecidas maneiras de o usuário navegar de forma não linear para acessar o áudio gravado da reunião, tanto através do uso dos quadros chaves quanto dos traços da caneta.

2.3.1.4 Um sistema de Projetor-quadro branco e câmera

O quarto módulo apresenta um software que se aproxima mais da proposta do presente trabalho. Este módulo demonstra um software que integra câmera, projetor, quadro branco e o computador. Essa integração é feita da seguinte forma: a câmera captura a apresentação feita pelo projetor, nessa imagem capturada é aplicado um algoritmo para realizar um tratamento de remoção do fundo, que é a projeção do documento que está sendo exibido pelo projetor. Para se realizar esse procedimento deve-se primeiro extrair a área de interesse, que é apenas a parte da imagem na qual é projetada a apresentação. O restante da imagem, bordas que não sejam projeção, devem ser descartadas. Após o recorte, é aplicado o algoritmo para extrair apenas o que foi escrito no quadro branco, chamado de *"visual echo cancelation"*.

2.3.1.4.1 Visual Echo Cancelation

O algoritmo *visual echo cancelation* tem como objetivo recortar o fundo da imagem capturado pela câmera, utilizando para isso a informação da imagem que está sendo projetada. Ou seja, este algoritmo tenta fazer um bom recorte comparando a imagem capturada com a imagem sendo projetada. Qualquer objeto que interfira na projeção, como o usuário, deve ser identificado.

Antes de aplicar o algoritmo *visual echo cancelation* são necessários dois tratamentos na imagem: o primeiro é a correção de perspectiva, que também é usada no módulo 1; e a segunda é a calibração das cores que, em resumo, é o cálculo da média das cores das imagens registradas pela câmera.

Para realizar a calibração de cores, Zhang (2005) realiza quatro passos. No primeiro a imagem projetada pelo computador é dividida em 32 x 32 blocos retangulares, resultando em 1024 blocos de imagem. No segundo busca-se reduzir o espaço de cor. Como as cores dos pixels em imagens geralmente são representadas no padrão RGB, no qual cada componente de cor varia de 0 a 255, temos 256 intensidades de vermelho, 256 intensidades de verde e 256 intensidades de azul, resultado em 256 x 256 x 256 possibilidades de cores. Para não se trabalhar com todas as cores, as cores são quantizadas em um espaço de cor de 9 x 9 x 9, que resulta em um espaço de verificação muito menor. No terceiro passo, realiza-se 5

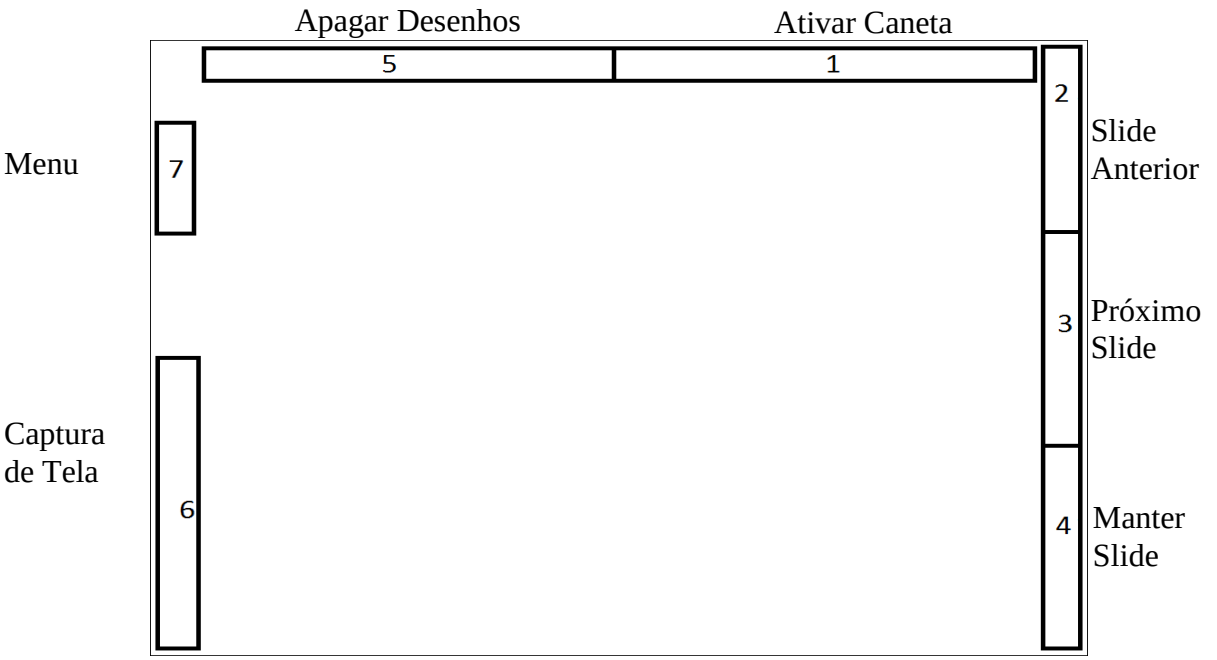
capturas das cores projetadas, e armazena-se o valor da cor de cada pixel desses 5 frames. Por fim, calcula-se e armazena-se a média e a variância da cor dos pixels em cada bloco, usando os 5 frames capturados anteriormente. Para se ter uma correspondência nas cores armazenadas e nas cores capturadas, as cores são armazenadas no espaço reduzido de 9 x 9 x 9.

Após o tratamento na imagem e o cálculo da média das cores é aplicado o *visual echo cancelation* em cada novo frame da imagem, usando as cores médias dos blocos armazenadas anteriormente. É verificado em cada quadro se houve alguma alteração significante nos valores de cores médias dos blocos contidos neste novo frame. Se houver uma mudança importante na cor do pixel, ele recebe a cor que foi identificada a mudança, caso contrário, o pixel não recebe uma cor, ficando com o seu valor RGB vazio (0, 0,0). Com isso é possível se extrair apenas o que foi escrito no quadro branco e gerar uma nova imagem com o que foi escrito no quadro. Essa imagem pode ser combinada com o documento e enviada para os participantes remotos.

3 A LOUSA DIGITAL

Quando estamos em uma apresentação de slides temos duas ações simples que são comumente realizadas, passar para o próximo slide e voltar o slide. Estas ações/funções compõem o uso mais simples de uma lousa digital. Para o nosso projeto adicionamos mais três ações: realizar anotações na tela, apagar todas as anotações da tela e capturar tela. No documento de requisitos, disponível no apêndice A, definimos as funcionalidades, as premissas e as restrições do software proposto. Com este documento definimos o escopo do software e suas restrições. Para o Software estar pronto para o uso temos antes que realizar algumas configurações, que são descritas na Seção 3.1.

Figura 5 – Divisão das regiões para ações do usuário.



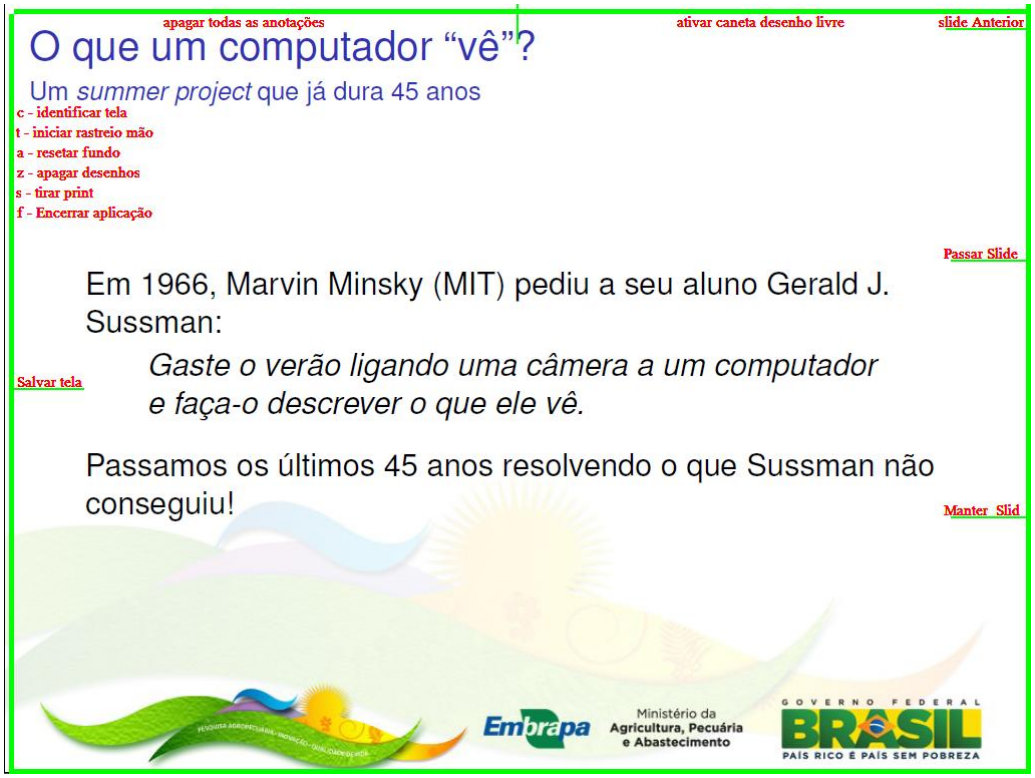
Fonte: Elaborada pelo autor

As ações/funções do software são realizadas tomando como base a posição da mão do usuário na tela projetada. Para isso realizamos uma divisão das bordas da tela de forma a posicionar as ações nestas bordas. Dividimos estas bordas em 7 regiões distintas e cada região está relacionada com uma função do software. A Figura 5 apresenta estas sete regiões, cada uma associada a uma ação específica do software. Estas regiões e ações são descritas nas próximas seções. A região 1 está relacionada à ação de ativar a caneta para desenho livre (Seção 3.2). As regiões 2 e 3 estão relacionadas com a ação de mudança de slide na apresentação, voltar um slide e avançar um slide, descritos nas Seções 3.3 e 3.4, respectivamente. A região 4 representa a ação de o usuário sair (retirar o braço) da tela de

apresentação, mantendo o slide atual (Seção 3.5). A região 5 relaciona-se com a ação de apagar todos os desenhos/anotações realizadas na tela (Seção 3.6). A região 6 realiza a ação de captura de tela (Seção 3.7). E por fim a região 7 contém um breve menu, onde são apresentados os comandos e as teclas que o usuário tem que usar para realizar cada comando, bem como a configuração do software (Seção 3.8). Essa divisão está visível para o usuário através de alterações na área de trabalho, onde são impressos textos e linhas indicando cada divisão e qual ação está relacionada. Podemos ver essa divisão das bordas na Figura 6, onde temos um exemplo de uso com um slide.

Temos outra alteração na área de trabalho que corresponde às linhas impressas nas extremidades da apresentação, formando um quadrado que engloba toda a tela (ver Figura 6). Esta alteração é usada para indicar se o usuário pode ou não posicionar a mão na área de apresentação. Caso o usuário possa posicionar a mão, as linhas serão desenhadas na cor verde. Quando as linhas estão impressas na cor vermelha, indica que o software está realizando a aprendizagem do fundo e que o usuário não deverá posicionar a mão na área projetada, pois isso afetaria o correto armazenamento da imagem que representa o fundo da tela e acarretaria em mau funcionamento do software. As linhas serão impressas na cor vermelha nas ações de mudança de slide (ver Seções 3.3 e 3.4).

Figura 6 – Exemplo de alteração da área de trabalho para identificação das regiões.



Fonte: Elaborada pelo autor

3.1 Configuração

Para podermos usar o software temos antes que realizar algumas configurações. Esta configuração se dá em dois passos: identificar a área de interesse e iniciar rastreio da mão. Para isso temos que obedecer algumas premissas importantes. Antes de iniciar o software o usuário já deve estar no modo de apresentação usando o slide. Após a configuração o usuário tem que trazer o software de apresentação (a janela no Windows que contém a apresentação) para o primeiro plano, de forma que as teclas de seta para direita e esquerda funcionem para realizar a mudança de slides. Isso pode ser feito apenas clicando no item da barra de tarefas que corresponde ao software de apresentação; Para que a apresentação receba foco.

Figura 7 – Exemplo de imagem de saída depois de pressionada tecla ‘c’.

Fonte: Elaborada pelo autor

Para identificar a área de interesse, o usuário deve pressionar a tecla ‘c’. Neste momento surge uma nova janela que contém uma imagem igual à que está sendo capturada pela câmera. No entanto, esta imagem possui quatro círculos que são adicionados automaticamente às extremidades de cada canto da área apresentada, conforme Figura 7. Caso estes círculos não estejam posicionados conforme a Figura 7, o usuário deve pressionar a tecla

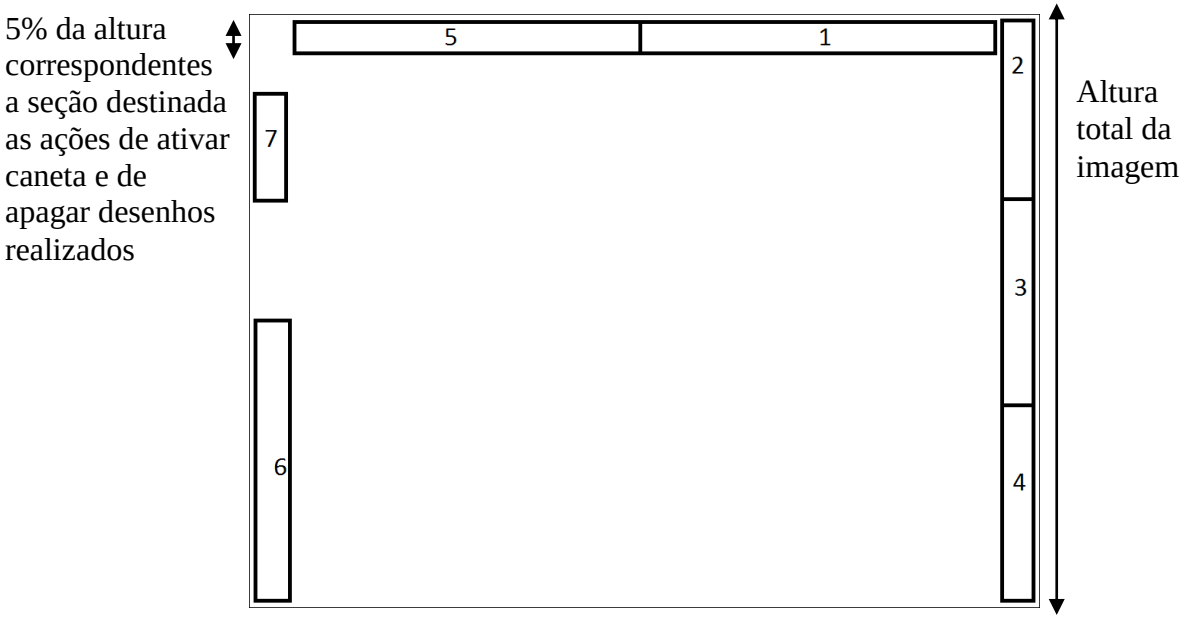
‘c’ novamente. Caso o usuário pressione a tecla ‘c’ várias vezes e o software não consiga posicionar os círculos nos locais corretos, deve-se mudar a posição da câmera de forma que a área projetada pelo computador fique o mais próximo possível das extremidades do vídeo/frame capturado pela câmera.

Após o usuário identificar que os círculos estão na posição correta, o próximo passo é iniciar o rastreio da mão pressionando a tecla ‘t’. Logo após ser pressionada a tecla ‘t’ surge uma janela com uma imagem que representa o fundo da cena. Esta imagem deve conter apenas a tela projetada na lousa. Neste momento o usuário deve estar atento para não se posicionar na frente da câmera, o que impede um correto armazenamento da imagem que corresponde ao fundo da tela. Caso ocorra do usuário estar presente na imagem que representa o fundo da tela, deve-se pressionar a tecla ‘a’. Esta tecla é usada para realizar uma nova captura da imagem que será usada como fundo da cena. Após a correta captura da imagem de fundo da tela, o usuário deve clicar no ícone localizado na barra de tarefas que corresponde ao software usado para apresentação, para que o mesmo receba foco.

3.2 Ativar caneta para desenho livre (Região 1)

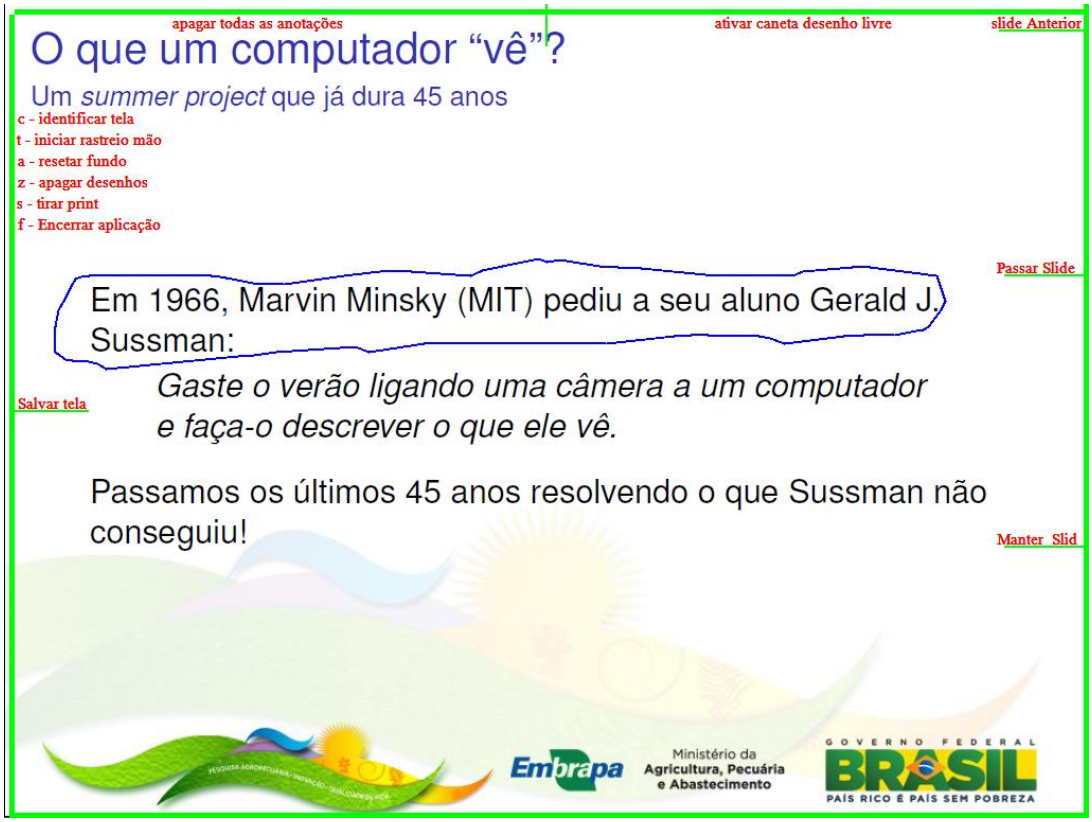
A região 1 é uma pequena faixa na parte superior da que corresponde a 5% da altura da imagem e é posicionada na segunda metade da borda superior (ver Figura 8). Usamos esta região para dar início à ação de realizar desenhos/anotações na tela. Após o usuário realizar a devida configuração do software, e quando o mesmo posicionar a mão nesta parte da borda superior, será impresso em forma de texto no canto superior esquerdo a seguinte frase: “Selecione ponto de inicio do desenho”. Após isso, o usuário deve levar a mão para o ponto onde deseja iniciar a anotação e manter a mão parada por um intervalo de tempo, o que informa para o software que o usuário deseja iniciar a anotação neste ponto. Como o mouse acompanha a mão do usuário, para o mesmo ter certeza que sua mão está parada deve conferir que o mouse não se movimenta, caso se movimente o software não poderá identificar que a mão está parada. Quando o software identificar o ponto de início, a tela irá piscar apagando todos os comandos impressos na tela (ver comandos na Figura 6). O usuário deve esperar até que os comandos/ações apareçam novamente na tela, juntamente com a frase “Desenhando” no canto superior esquerdo, e então neste momento o usuário pode movimentar a sua mão produzindo assim as linhas que representam as suas anotações. Para parar de realizar anotações na tela, basta manter a mão parada no ponto desejado.

Figura 8 – Faixa superior correspondente a ação de ativar caneta para desenho livre e a ação de apagar todos os desenhos da tela.



Fonte: Elaborada pelo autor

Figura 9 – Exemplo de anotação em apresentação.



Fonte: Elaborada pelo autor

Após o software identificar um novo ponto de parada da mão, a tela irá piscar novamente redesenhando os comandos e as anotações realizadas. Neste momento, o usuário

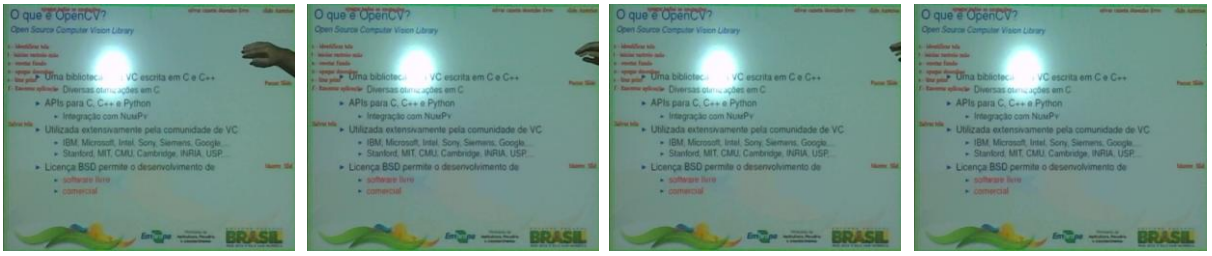
pode realizar qualquer nova ação do software, seja ela iniciar um novo desenho, apagar todos os desenhos, mudar de slide ou realizar uma captura de tela. Na Figura 9 podemos ver um exemplo em que o usuário realizou um destaque em um determinado texto na apresentação.

3.3 Slide anterior (Região 2)

A região 2 corresponde ao primeiro terço da borda direita (ver Figura 5 e Figura 6). As ações relacionadas com mudança de slide são atreladas ao movimento de saída da tela de apresentação. Isto é necessário devido à necessidade de armazenamento de uma imagem que represente o fundo da tela. Veremos mais detalhes sobre esta característica na Seção 4.4

Para realizar a mudança de slide para o slide anterior, o usuário deve realizar o movimento de saída da tela (área projetada) removendo sua mão pelo lado direito na região 2. Podemos ver um exemplo da ação de voltar o slide na Figura 10, onde o usuário realiza o movimento de saída da tela removendo a mão passando pela área designada para esta ação. Após a mudança do slide as cores das bordas exteriores ficam em vermelho, o que indica que o usuário não deverá posicionar a mão na área projetada (ver Seção 3). Após a devida aprendizagem da imagem que representa o fundo da cena, as bordas voltaram a ser desenhadas na cor verde, o que indica que o usuário pode posicionar a mão na área projetada.

Figura 10 – Exemplo da ação de voltar slide.



(a) (b) (c) (d)

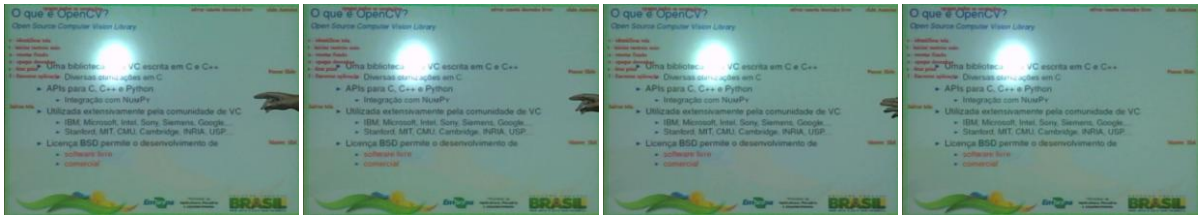
Fonte: Elaborada pelo autor

3.4 Próximo slide (Região 3)

A região 3 corresponde à região mediana da borda direita (ver Figura 5 e Figura 6) e está relacionada à ação de passar para o próximo slide. No momento que o usuário deixar a tela removendo a mão por completo e passando por esta região (ver Figura 11), a ação de passar slide será identificada. Após o software identificar que o usuário deixou a tela passando por esta região, será realizada a mudança de slide. Assim como na ação de slide anterior (Seção 3.3), as bordas irão ficar vermelhas, o que indica que o usuário não deve posicionar a

mão na área projetada. Após as bordas voltarem a ser desenhadas na cor verde, o usuário poderá realizar novas ações. Ao ocorrer uma mudança de slide, seja ela passar para slide anterior ou para o próximo slide, as anotações presentes nesta tela serão removidas.

Figura 11 – Exemplo de ação próximo slide.



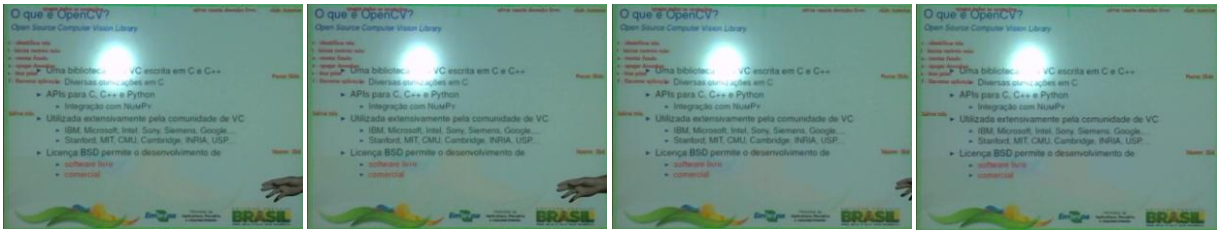
(a) (b) (c) (d)

Fonte: Elaborada pelo autor

3.5 Manter mesmo slide (Região 4)

A região 4 corresponde à região inferior da borda direita (ver Figura 5 e Figura 6). Esta região foi escolhida para o caso de o usuário querer sair da tela e manter o slide atual, visto que o usuário ocasionalmente pode querer sair da área projetada sem realizar qualquer ação. Podemos ver um exemplo na Figura 12. Pois esta é a altura natural da mão do usuário. Neste momento o software identifica que o usuário quer sair da tela sem realizar ação alguma então a apresentação se mantém no slide atual e com suas respectivas anotações, caso existam.

Figura 12 – Exemplo de saída da tela projetada sem realizar ações de mudança de slide.



(a) (b) (c) (d)

Fonte: Elaborada pelo autor

3.6 Apagar todos os desenhos (Região 5)

A região 5 é posicionada na primeira metade da borda superior da tela e correspondente a 5% da altura da imagem (ver Figura 8). No momento em que o usuário posicionar a mão nesta região, todas as anotações realizadas na tela serão apagas. A tela irá piscar apagando todas as alterações realizadas pelo software na tela projetada.

3.7 Salvar captura de tela (Região 6)

A região 6 é posicionada na parte inferior da borda esquerda. No momento que o usuário posicionar sua mão nesta região, o software irá salvar capturas de tela e salvar estas imagens no disco do usuário. Como o software não possui uma interface gráfica com janelas, não é possível solicitar em qual pasta o usuário deseja salvar estas capturas de tela. Assim, foi escolhido uma pasta default com o nome de “screenCaptures” no disco C: do usuário (C:\ screenCaptures). Estas imagens são salvas com o nome de “screen_X.bmp”, onde X representa um número que irá servir para o usuário saber qual a ordem das capturas de tela. Estes números não necessariamente são sequenciais. No entanto, o usuário poderá identificar a ordem das capturas devido a essa numeração ser gerada em ordem crescente.

3.8 Comandos no teclado (Região 7)

A região é posicionada na parte superior da borda esquerda. Esta região é usada para os comandos que podem ser realizados pressionando teclas (ver Figura 5 e Figura 6). Estes comandos são usados para realizar a configuração do software (Seção 3.1), bem como corrigir algum armazenamento de imagem de fundo errado (Seção 3.1). Também estão disponíveis os comandos de apagar desenhos pressionando a tecla ‘z’, o comando de realizar captura de tela, pressionando a tecla ‘s’ e o comando de finalizar a aplicação de lousa digital, pressionando a tecla ‘f’. Para que o software possa identificar esses comandos é necessário que o software em primeiro plano seja uma das janelas que exibem as imagens, seja a imagem da câmera ou a imagem de fundo.

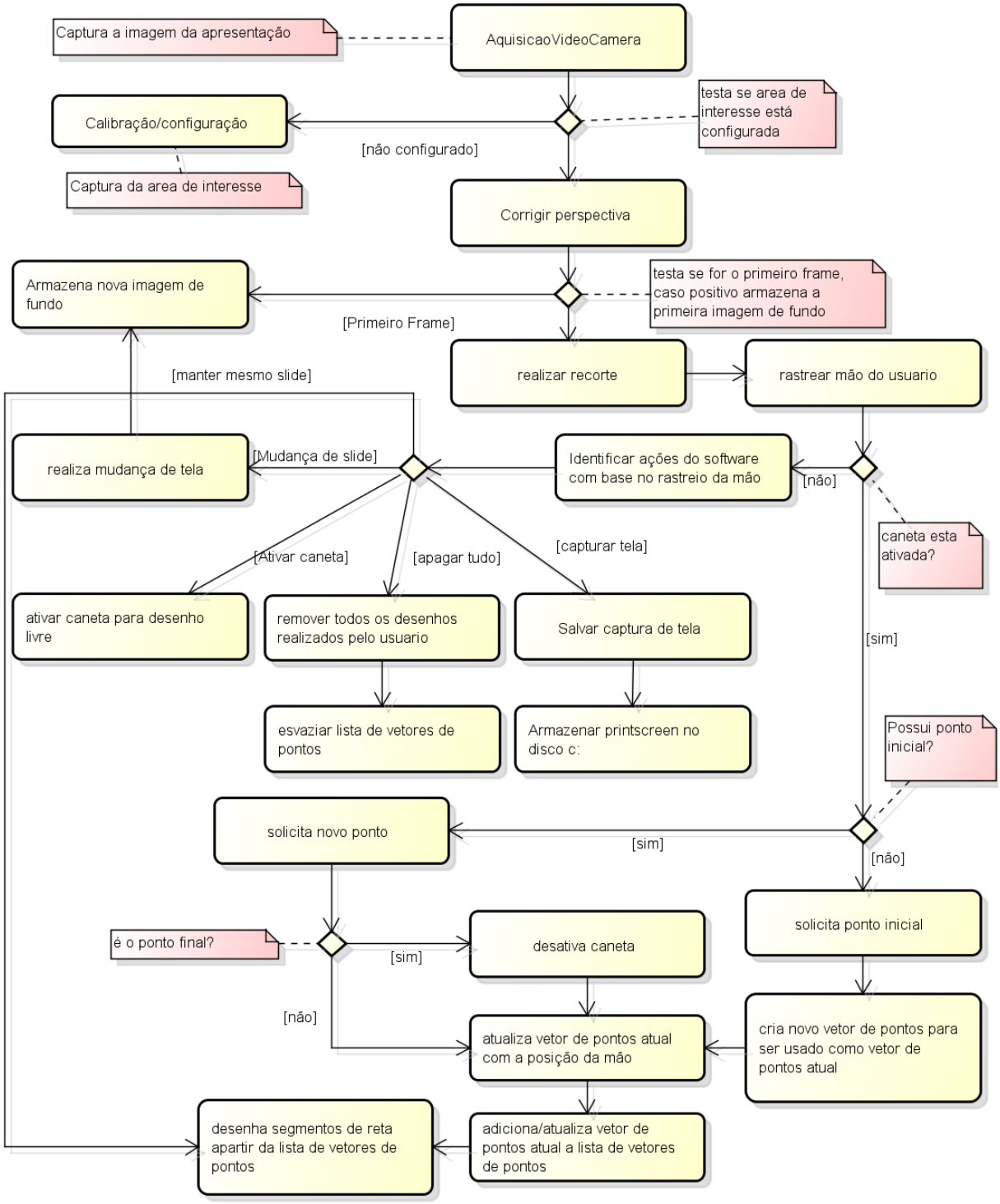
3.9 Modelagem da interface e testes com usuário

No inicio do desenvolvimento convidamos duas especialistas em IHC para usar o software. Estas receberam explicações de uso da lousa e nos deram sugestões de melhorias relacionadas ao uso do software. Estas melhorias estão relacionadas à forma de dar um feedback para o usuário, indicando que o mesmo está realizando ou realizou alguma ação. Também indicaram que deveria haver testes com usuários finais. Os testes foram realizados de forma informal, ou seja, sem roteiro definido e sem documentação apropriada.

Realizamos testes informais com usuários onde podemos perceber alguns problemas relacionados à usabilidade. Onde podemos realizar uma correção na disposição das ações do software. Descrevemos este problema e sua solução na Seção 4.6.5.

4 IMPLEMENTAÇÃO

Figura 13 – Fluxograma do software .



Fonte: Elaborada pelo autor

Dividimos o sistema em 6 módulos: aquisição, identificação da área de interesse, recorte, rastreio da mão, identificar ações e realizar ações. Os módulos podem ser vistos na Figura 13, na qual vemos a sequência de execução de cada um deles. No primeiro módulo, o

de aquisição, capturamos o vídeo vindo da câmera e transformamos/dividimos em imagens para que sirvam de entrada para os módulos seguintes (Seção 4.1). No módulo de identificação de área de interesse, extraímos do frame apenas a área na qual contenha a imagem projetada e realizamos a correção de perspectiva (Seção 4.2)

O módulo de recorte é onde realizamos a extração/recorte da mão do usuário contida no frame, usando a imagem que representa o fundo do frame e a imagem que contém a mão do usuário. Para conseguirmos a imagem que representa o fundo do frame tentamos usar o algoritmo MOG (Seção 2.2.3), que não atendeu às necessidades do nosso software (Seção 4.3). Desenvolvemos uma técnica de aprendizado discreto que explicamos na Seção 4.4. O módulo de rastreio da mão é usado para mover o mouse e para dar suporte ao módulo de identificar ações.

As ações são as funções básicas do software explicadas nas seções anteriores. E por fim, temos o módulo de realizar ações, o qual realiza a comunicação com as bibliotecas do sistema para realizar as ações básicas do software. Na Figura 13 temos uma descrição mais detalhada de como é o funcionamento do software e exibimos uma visão geral das ações que são tomadas a cada iteração do software.

Nas seções seguintes descrevemos os principais passos usados na implementação do software proposto no projeto, que consistem em: definição das configurações da câmera e da forma de coleta de dados (Seção 4.1); processo de correção de perspectiva (Seção 4.2); processo de identificação da posição da mão do usuário na imagem (Seção 4.4); posicionamento do mouse e anotações na imagem (Seção 4.5); e por fim os problemas encontrados (Seções 4.6 e Figura 22).

4.1 Coleta de dados (vídeos de testes)

As imagens de teste foram coletadas de uma webcam integrada a um notebook com as seguintes características: 30 frames por segundo, 2.0 megapixels e 640x480 de resolução de imagem. Esta configuração é apropriada, pois é facilmente encontrada nos notebooks atuais. Foram capturados dez vídeos, que foram usados nos testes iniciais. Os vídeos coletados têm duração de 3 minutos. Com esta duração é possível capturar os frames necessários para realizar o recorte e registrar alguns movimentos de interação do usuário com a lousa, como ilustra a Figura 14

Figura 14 – Exemplo de iteração do usuário com a superfície da lousa.

Fonte: Elaborada pelo autor

4.2 Correção de perspectiva

Conforme a Figura 15 (a), a área projetada pode não formar um retângulo, devido à diferença de posicionamento entre a câmera e o projetor. Portanto, para facilitar a identificação de posições na imagem, realizamos um procedimento de "correção de perspectiva" (Seção 2.2.2), conforme ilustrado na Figura 16.

Figura 15 – Exemplo de entrada e saída do algoritmo *fast feature detector*.

(a) Imagem de entrada do algoritmo

(b) imagem de saída do algoritmo

Fonte: Elaborada pelo autor

Para realizar a correção da perspectiva é aplicado à imagem o algoritmo *FAST* (ROSTEN; DRUMMOND. 2006), como descrito na Seção 2.2.1. Este algoritmo recebe como entrada uma imagem e retorna um vetor de pontos de interesse, que podem ser desenhados na imagem, conforme ilustrado na Figura 15 (b). O vetor de pontos representa 'cantos' (ver Seção 2.2.1) encontrados na imagem.

Utilizamos apenas os quatro cantos mais externos para realizar a correção de perspectiva. Estes cantos podem ser vistos nas Figura 16 (a) e Figura 7, com círculos os destacando. Esta correção é feita com o uso do algoritmo *find Homography* presente na biblioteca OpenCV (LAGANIÈRE, 2010). O resultado deste processo é uma imagem retangular (Figura 16 (b)), que facilita o rastreamento da mão do usuário

Figura 16– Exemplo de correção de perspectiva.

(a) Imagem de entrada do algoritmo de perspectiva com os quatro cantos marcados em vermelho

(b) Imagem de saída do algoritmo de correção de perspectiva. Retângulo perfeito da área projetada.

Fonte: Elaborada pelo autor

4.3 Recorte

Após a correção de perspectiva, estamos interessados em identificar a mão do usuário na imagem. Para isso, avaliamos nesta etapa estratégias disponíveis na literatura para remoção de fundo em vídeos. Usamos o algoritmo *background subtractor* (MOG) da biblioteca OpenCV, que foi descrito na Seção 2.2.3. O algoritmo MOG implementa o algoritmo de Kaewtrakulpong Bowden (KAEWTRAKULPONG, 2002). O algoritmo funciona bem para alguns vídeos e algumas aplicações de tempo real, no entanto para o nosso caso o seu desempenho no recorte não foi aceitável, pois o algoritmo apresentou algumas instabilidades. Quando ocorria alguma interação do usuário com o quadro, a mão do usuário refletia muita luz, vinda do projetor, o que causava alteração em toda a imagem da cena, fazendo com que o algoritmo identificasse que ouve mudança em grande parte do fundo da imagem, o que não acontecia de fato, pois a mudança era pequena (apenas o braço do usuário).

Outro fator que contribuiu para que descartássemos o algoritmo MOG é o seu fator de aprendizagem do fundo. Devido ao fato do algoritmo usar o histórico dos fundos anteriores para calcular a próxima remoção de fundo, com o passar do tempo o algoritmo passa a identificar o usuário como sendo uma parte do fundo se ele permanecer parado, o que

impossibilita a identificação da posição do braço do usuário na cena com precisão. Na Figura 17 podemos ver dois exemplos disso, na Figura 17 (a) temos um exemplo de aprendizado em um fundo colorido, e na Figura 17 (b) um exemplo de aprendizado do fundo em um fundo totalmente branco.

Figura 17 – Exemplos de aprendizagem do fundo usando o algoritmo MOG e utilizando aprendizagem discreta.

(a) Imagem de fundo, segundo algoritmo MOG, em cenário colorido.

(b) Imagem de fundo, segundo algoritmo MOG, em cenário totalmente branco.

(c) Aprendizagem em fundo colorido, através de aprendizagem discreta (Seção 4.4)

(d) Aprendizagem em fundo branco, através de aprendizagem discreta (Seção 4.4)

Fonte: Elaborada pelo autor

Como podemos ver nas Figura 17 (a) e Figura 17 (b), na parte direita existe uma leve sombra/borrão em relação às imagens de fundo utilizando a aprendizagem discreta (Figura 17 (c) e Figura 17 (d)). Isso ocorre devido ao algoritmo começar a identificar o usuário como parte do fundo. Esta sobra/borrão faz com que a mão do usuário não seja identificada com precisão ao sair da tela. Quando o usuário estiver com a mão posicionada na borda direita e for realizada a subtração das imagens (ver Figura 2) a mão se confunde com o fundo aprendido. Podemos perceber mais claramente na Figura 17 (b), onde vemos que o

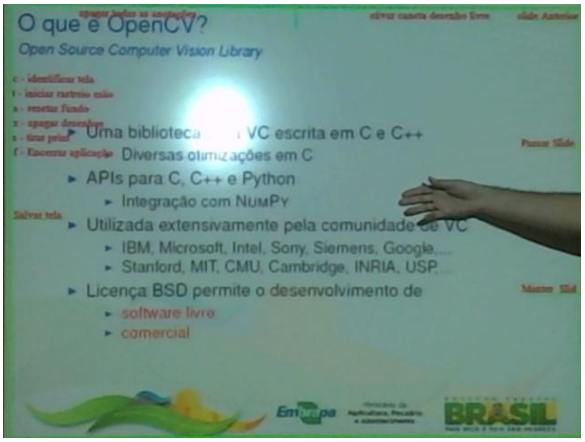
algoritmo aprendeu inclusive a cor da blusa do usuário. Podemos ver o fundo que deveria ser o resultado do algoritmo, nas Figura 17 (c) e Figura 17 (d) para os casos de fundo colorido e de fundo branco, respectivamente.

Outra dificuldade em relação ao uso deste algoritmo é que o mesmo é muito sensível a mudanças. Pequenas mudanças como a movimentação da seta ao acompanhar a mão do usuário são identificadas como mudanças consideráveis na imagem de fundo, o que gera instabilidade, fazendo com que o mouse se movimente sem a interação do usuário. Devido a essas instabilidades do algoritmo MOG, realizamos uma aprendizagem discreta customizada para a nossa aplicação.

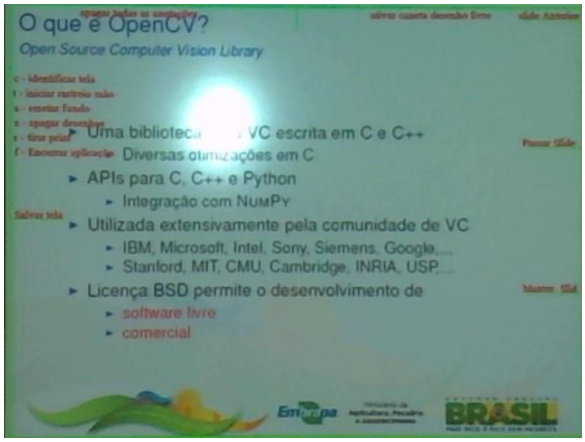
4.4 Recorte com aprendizado discreto de fundo

Para contornar as dificuldades encontradas no algoritmo MOG, implementamos uma estratégia na qual podemos aprender o fundo e usá-lo de forma mais simples. Nesta estratégia aprendemos o fundo da imagem utilizando apenas uma captura de imagem, realizada a cada evento de mudança de slide, e não usamos o histórico assim como no algoritmo MOG.

Figura 18 – Exemplos de imagem atual da câmera e seu respectivo fundo.



(a) Imagem atual capturada pela câmera e realizada a correção de perspectiva



(b) Imagem armazenada usada para representar o fundo da imagem atual capturada pela câmera

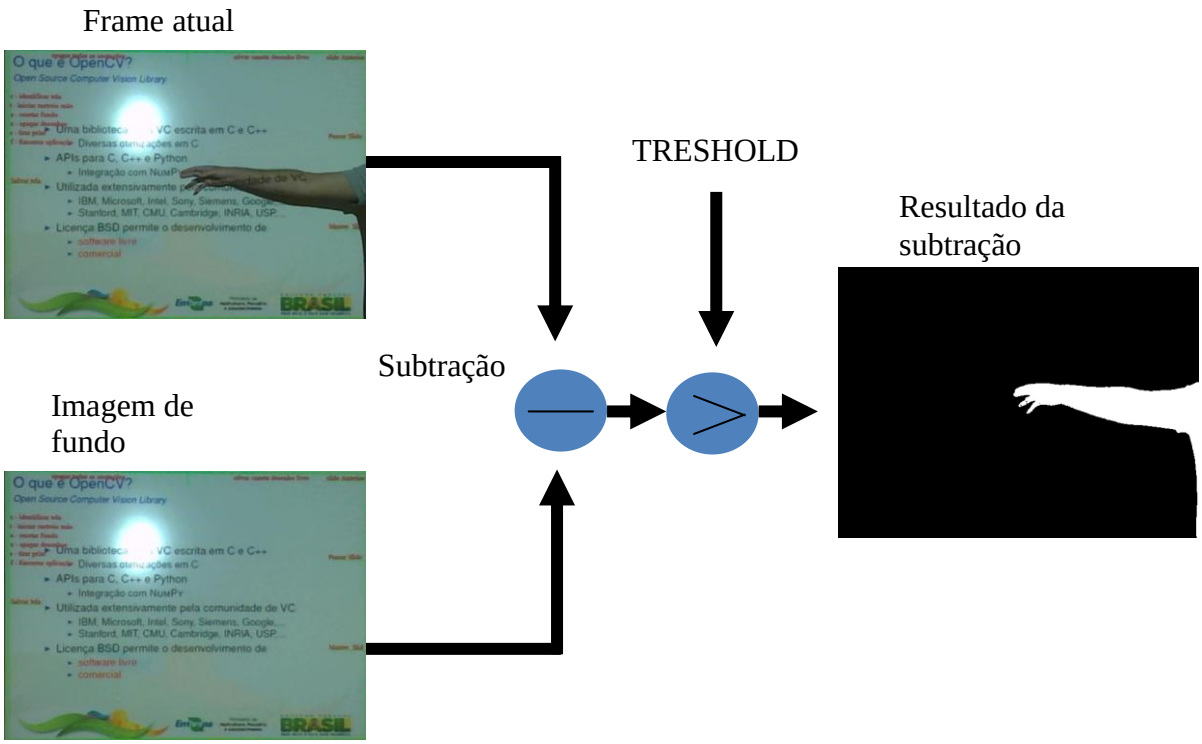
Fonte: Elaborada pelo autor

Como vimos na Seção 2.2.3, algoritmos de remoção de fundo necessitam de uma imagem que represente o fundo, e uma segunda imagem que represente a cena atual. Podemos ver um exemplo destas duas imagens na Figura 18 onde a Figura 18 (a) representa a imagem atual da câmera e a Figura 18 (b) representa o fundo neste momento. Para conseguirmos a

segunda imagem, a que representa o fundo da cena, adotamos a estratégia de associar as mudanças de slide com a saída do usuário da área projetada, como visto nas Seções 3.3 e 3.4.

As mudanças de slide de fato representam a mudança da imagem de fundo, visto que a imagem de fundo é a imagem que está sendo projetada pelo computador. Então, quando mudamos esta imagem projetada mudamos a imagem de fundo por completo. Associando esta mudança com a saída do usuário, teremos um intervalo de tempo em que imagem de fundo e a imagem resultante do algoritmo de correção de perspectiva são as mesmas. Este é o momento no qual realizamos o aprendizado (armazenamento) da imagem de fundo, trocando a imagem antes armazenada pela imagem atual capturada pela câmera (e realizada a correção de perspectiva).

Figura 19 – Entrada, processamento e saída do algoritmo de remoção de fundo.



Fonte: Elaborada pelo autor

Após o devido aprendizado da imagem de fundo, podemos realizar uma remoção de fundo apenas subtraindo as duas imagens. Como vimos na Seção 2.2.3, onde explicamos o funcionamento do algoritmo MOG, quando temos a imagem de fundo – ou modelo de fundo como é descrito pelo MOG – basta apenas realizar uma subtração nestas imagens e, após isso, aplicar uma limiarização. A subtração é possível devido à estrutura de dados usada para representar uma imagem. No OpenCV uma imagem é representada como uma matriz, onde a altura e largura da imagem representam os índices máximos da imagem. Podemos ver o resultado final do processamento na Figura 19, onde temos a representação das imagens de

fundo e imagem atual, realizando a subtração e logo após uma limiarização (Threshold), o resultado final é uma imagem binária (apenas preto e branco) onde o que está em cor branca representa a alteração na cena.

Com o nosso algoritmo de aprendizagem discreta de fundo contornamos os problemas encontrados no algoritmo MOG, pois o aprendizado da imagem de fundo acontece apenas uma vez (em cada mudança de slide), diferente do MOG que usa o histórico de todos os frames do vídeo. O fato de realizarmos apenas um armazenamento simples da imagem de fundo também traz vantagens em relação ao algoritmo Visual Echo Cancelation. Este também usa a imagem projetada para realizar o aprendizado da imagem de fundo, porém seu aprendizado é realizado percorrendo toda a matriz que representa a imagem e calculando a média das cores que são apresentadas em um processo chamado "calibração de cores" (ver Seção 2.3.1.4.1). Este processo levaria mais tempo do que o simples armazenamento da imagem de fundo, como foi feito em nossa implementação.

4.4.1 Retornar para a tela no momento da captura do fundo

Como realizamos apenas um armazenamento simples da imagem capturada pela câmera, podemos ter um problema relacionado a isso. O usuário pode, acidentalmente, posicionar a mão na área projetada durante o armazenamento desta imagem. Para contornar este possível problema adicionamos a impressão de linhas nas extremidades das quatro bordas da apresentação, que ficaram na cor vermelha indicando que o usuário não pode posicionar a mão neste momento. No entanto, se mesmo assim o usuário posicionar a mão na área projetada, teremos uma imagem de representação do fundo que não contém apenas o que está sendo projetado, e sim uma imagem do que está sendo projetada adicionado da mão do usuário. Para contornarmos este problema, verificamos se a imagem de fundo foi armazenada corretamente, conforme descrito a seguir.

O momento de armazenar uma nova imagem que representa o fundo da tela é após um evento de mudança de slide. Após este evento acontecer, teremos um intervalo de tempo em que o usuário não deverá posicionar a mão na área projetada (cor das bordas estará vermelha). Estendemos este intervalo de tempo de forma que, após o software armazenar a imagem que representa o fundo da tela, na iteração seguinte o usuário ainda esteja impedido de posicionar a mão na tela. Assim, temos um intervalo de tempo em que a imagem de fundo e a imagem capturada pela câmera devem ser iguais, ou seja, não existem alterações na cena.

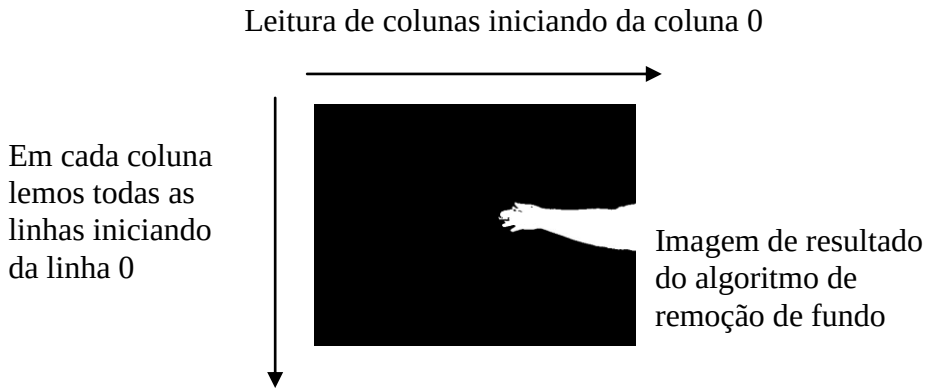
Quando aplicarmos a etapa de subtração nestas duas imagens, a imagem resultante deverá estar totalmente na cor preta indicando que não existem alterações na cena. No

entanto, se a imagem de fundo não possuir apenas o que está sendo projetado (o usuário posicionou a mão no momento da captura, por exemplo), a imagem resultante da subtração não estará totalmente na cor preta. Sendo assim, para verificar se a imagem de fundo foi armazenada corretamente, apenas validamos se a imagem resultante da subtração está totalmente preta. Caso não esteja, indicamos que o fundo deve ser capturado novamente nas próximas iterações.

4.5 Posicionar mouse e realizar anotações na imagem projetada

O mouse é posicionado com base no recorte (ver Seção 4.4). Com o resultado do algoritmo de recorte de fundo podemos posicionar o mouse no ponto correspondente à mão do usuário. Como a imagem resultante do algoritmo de recorte é uma imagem binária, contendo apenas pixels nas cores pretas (RGB 0, 0, 0) e branca (RGB 255, 255, 255) podemos percorrer esta imagem (que é representada na forma de matriz) e verificar onde o usuário está apontando/interagindo com a lousa apenas checando as cores dos pixels. Realizamos esta verificação percorrendo a imagem (matriz) da esquerda para direita e do topo para a borda inferior, ou seja, percorremos a matriz que representa a imagem coluna por coluna (ver Figura 20). Esta abordagem gerou a restrição de o usuário somente poder usar a lousa estando posicionado no lado direito da apresentação.

Figura 20 – Sentido em que é verificada a posição da mão do usuário.



Fonte: Elaborada pelo autor

Após a identificação do local onde o usuário está interagindo com a lousa, realizamos a movimentação do mouse usando uma regra de três simples na qual, a partir do ponto do vídeo onde ocorre a interação, é feito o cálculo do ponto correspondente na tela do computador que está sendo projetada. Uma vez calculado este ponto, podemos usar a API do Windows para mover o mouse para esta posição.

Esta movimentação do mouse é usada para verificar quais ações o software irá realizar. Usando a posição do mouse na tela, e consequentemente o ponto onde o usuário está interagindo com a lousa, verificamos qual ação o usuário deseja realizar. Dentre estas opções está a de realizar anotações na tela. Depois de identificarmos que o usuário deseja realizar esta ação (ver Seção 3.2), iniciamos o armazenamento dos pontos onde o usuário movimentou sua mão e armazenamos estes pontos em um vetor que representa o histórico de pontos onde a anotação foi realizada. A cada movimento da mão do usuário armazenamos um novo ponto neste vetor e atualizamos a linha desenhada sobre a tela.

De posse dos pontos onde o usuário realizou a interação, podemos realizar o desenho da linha, que representa a anotação do usuário. Realizamos estes desenhos usado a biblioteca do Windows chamada GDIPlus (ou GDI+). Também usamos esta biblioteca para realizar as alterações na área de trabalho (ver Seção 3 e Figura 6)

> O Windows GDI + é uma API baseada em classes para programadores C / C++. Ela permite que aplicativos usem elementos gráficos e texto formatado tanto nos dispositivos de vídeo e quando nos dispositivos de impressão física. Aplicações baseadas na API Microsoft Win32 não acessam hardware de gráficos diretamente. Em vez disso, GDI + interage com controladores de dispositivos em nome dos aplicativos. GDI + também é suportado por aplicações Microsoft Win64. (MICROSOFT, 2015c, Tradução nossa)

Através do método “DrawLines” podemos desenhar uma sequência de linhas conectadas usando como entrada o vetor de pontos armazenado. Este processo se repete a cada nova entrada no vetor de pontos, ou seja, a cada vez que identificamos uma nova posição da mão do usuário, e acaba quando identificarmos que o usuário parou de realizar anotações na tela (ver Seção 3.2). Quando identificamos esta ação de parar de realizar anotações, interrompemos o armazenamento de novas posições da mão do usuário no histórico de pontos usado na anotação e continuamos a desenhas as linhas somente com estes pontos. As linhas só deixarão de existir no momento que o usuário realizar a ação de apagar todos os desenhos (ver Seção 3.5). Neste momento removemos todos os pontos armazenados.

4.6 Problemas encontrados e algumas soluções

Durante o desenvolvimento nos deparamos com alguns problemas. Nesta seção iremos descrevê-los e explicar como conseguimos resolver alguns deles. Nas subseções seguintes explicamos os problemas: instabilidade na movimentação do mouse e impossibilidade do usuário ver o mouse (Seção 4.6.1), movimentar o mouse em tela expandida (Seção 4.6.2), desenhar ações do usuário em tela expandida (Seção 4.6.3),

necessidade do uso do padrão de projeto Singleton em algumas classes (Seção 4.6.4) e problemas relacionados à usabilidade.

4.6.1 Instabilidade na movimentação do mouse e impossibilidade do usuário ver o mouse

Um vídeo é composto de uma sequência de várias imagens, atingindo cerca de 30 imagens por segundo em webcams comuns do mercado. A cada imagem dá-se o nome de "frame". No caso do nosso software realizamos todo o processamento da imagem, realizando todas as suas transformações (explicadas nas seções anteriores) a cada frame. Um destes processamentos é o de verificar onde o usuário está interagindo com a lousa e movimentar o mouse para este ponto específico na tela (Seção 4.5). Devido a termos muitas imagens processadas por segundo, cada imagem processada pode possuir uma pequena diferença entre a imagem atual e a imagem anterior. Esta diferença pode não necessariamente significar uma movimentação da mão do usuário. Esta diferença acaba gerando dados falsos no processamento do software, o que causa uma instabilidade no ponteiro do mouse, que irá se movimentar em várias posições próximas durante este processamento. Para contornar este problema, realizamos uma suavização das posições registradas pelo software, que consiste em movimentar a mão do usuário utilizando uma média entre um curto histórico de posições do mouse. Utilizamos um histórico de três posições e realizamos a média entre estes pontos. Com isso reduzimos o ruído e a movimentação do mouse fica mais suave, embora acabe gerando atraso na movimentação do mouse.

Nos testes iniciais de desenvolvimento estávamos posicionando o ponteiro do mouse no ponto exato onde está a mão do usuário. Isto gerou um pequeno problema de usabilidade. O ponteiro é desenhado à direita do ponto de interação o que dificulta a visão de onde o ponteiro do mouse está. A solução adotada foi posicionar o mouse alguns pixels para a esquerda da tela, fazendo com que o mouse esteja sempre posicionado bem próximo à mão do usuário, e fazendo com que o mesmo veja claramente o ponteiro projetado na lousa.

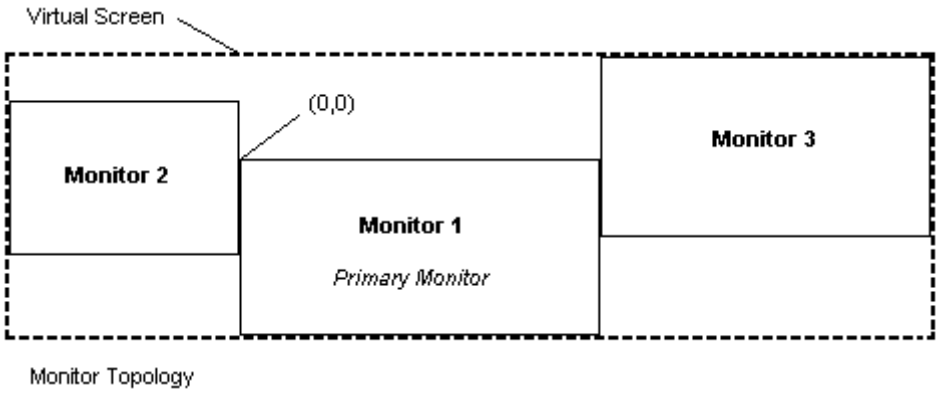
4.6.2 Movimentar o mouse na tela expandida

Para realizar a movimentação do mouse temos que antes ter as medidas da tela do computador. Isto é simples de obter quando temos apenas uma tela usando a API do Windows, através do método "*GetSystemMetrics*". Com este método podemos obter as informações de altura e largura da tela do computador (MICROSOFT 2015d) e realizar a movimentação do mouse usando a regra de três simples para realizar o mapeamento da

posição identificada na imagem para a posição real na tela projetada (ver Seção 4.5). No entanto isto só funciona bem para o caso de uma tela.

Quando temos telas expandidas, temos que trabalhar com o conceito de tela virtual e movimentar o mouse tomando como base os vários monitores contidos nesta tela. A tela virtual é uma representação de todas as telas conectadas ao computador, formando um retângulo onde todos os monitores estão inclusos (MICROSOFT 2015 e). Podemos ver um exemplo na Figura 21 onde existem 3, monitores sendo o primário (o que contém a coordenada X e Y (0, 0)) o que está posicionado ao centro.

Figura 21– Exemplo de tela virtual que comporta 3 monitores.



Fonte: MICROSOFT 2015d

Conforme a disposição dos monitores na Figura 21, caso seja necessário posicionar o mouse no canto superior do monitor 2, temos que usar valores negativos para as coordenadas X e Y do ponteiro. E caso seja necessário posicionar o mouse no canto superior direito do monitor 3, teríamos que usar um valor positivo para a coordenada X (este valor é obtido somando os valores de largura dos monitores 1 e 3) e um valor negativo para a coordenada Y. No nosso software consideramos sempre que a projeção está sendo realizada no monitor 2, e que este monitor está posicionado à esquerda do monitor principal, e com o canto superior esquerdo alinhado com o canto superior direito do monitor principal. Desta forma os valores de coordenada X e Y usados para movimentar o mouse serão sempre números positivos.

4.6.3 Desenhar ações do usuário em tela expandida

De forma semelhante à movimentação do mouse, o desenho na tela usando GDI Plus (MICROSOFT 2015c) funciona de forma mais simples para o caso de termos apenas uma tela. Basta apenas escolher o método de desenho desejado (desenho de linhas, polígonos diversos ou desenho de texto) e passar as coordenadas realizando uma regra de três simples. No entanto, para realizarmos isto em monitores diferentes, temos que usar o conceito de

"*Device Context*", usando a função CreateDC (MICROSOFT 2015a). Neste caso, o device é o monitor usado para realizar os desenhos.

O uso desta função causou alguns problemas na implementação do software, pois para conseguirmos criar um "*Device context*" que referencie para o monitor correto tentamos usar o método "*EnumDisplayDevices*", que obtém informações sobre os devices do computador (MICROSOFT 2015b). Nesta função conseguimos criar o "*Device Context*" apropriado para cada tela. No entanto, esta função possui um custo elevado de processamento devido a enumerar vários dispositivos. Resolvemos criar o "*Device Context*" apenas passando o nome do device, que pode ser "\\\\.\\DISPLAY2" para o caso de uso do segundo monitor (tela expandida) e "\\\\.\\DISPLAY1" para o caso de usar apenas uma tela, assim contornamos o problema de desempenho relacionado ao método "*EnumDisplayDevices*". Criar o "*Device Context*" desta forma trouxe mais desempenho para o software em comparação a usar o método "*EnumDisplayDevices*", pois este último enumera todos os devices do computador.

4.6.4 Necessidade do uso do padrão de projeto Singleton em algumas classes

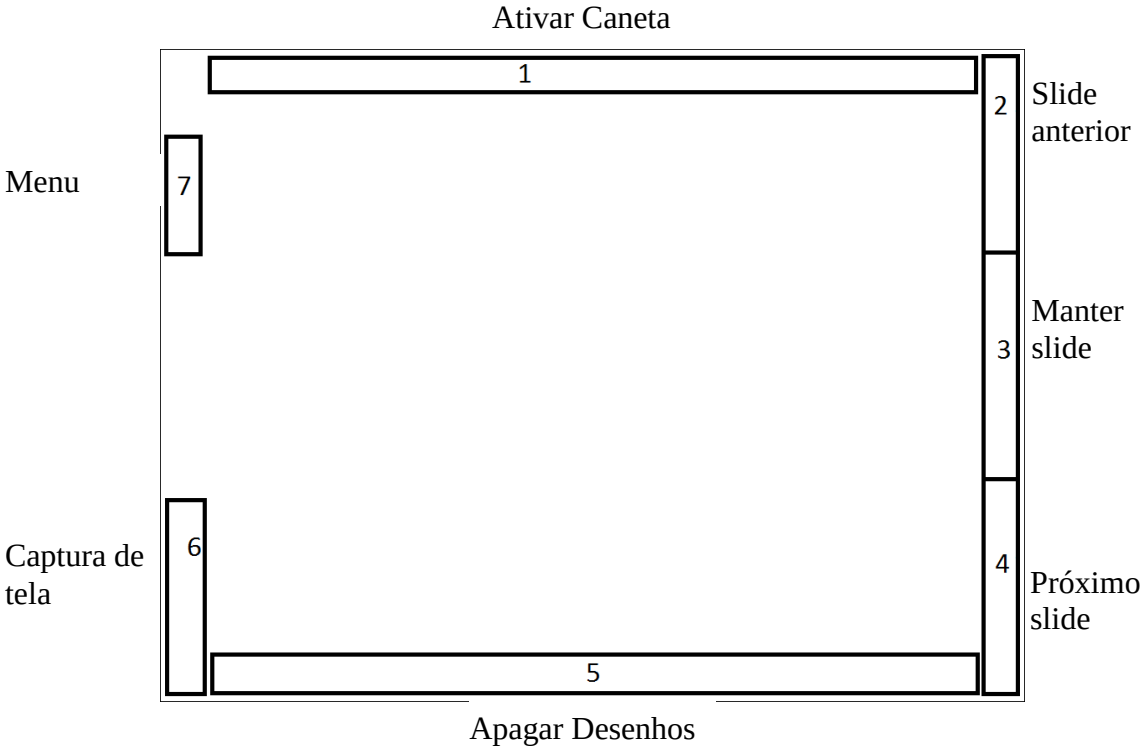
O software foi desenvolvido usando os conceitos de orientação a objetos. Com isso temos muitos momentos em que temos que instanciar uma classe antes de usá-la. Como o processamento de vídeos se dá em forma de loop, em cada iteração realizamos o processamento de um frame. Neste cenário, instanciar objetos de forma repetitiva muitas vezes por segundo causou um uso de memória muito elevado enquanto realizávamos testes iniciais do software. Também há classes, como a que armazena a imagem de fundo, que devem possuir apenas uma instância devido a seus atributos não poderem receber novos valores padrão a cada iteração.

Devido a estes problemas, aplicamos o padrão de desenvolvimento de software Singleton, que torna a própria classe responsável por sua instância. Desta forma sempre que necessitamos de uma nova instância de um objeto desta classe, a mesma irá nos retornar apenas um ponteiro para a instância deste objeto já existente. Caso ainda não exista, realiza uma instanciação deste objeto e armazena o ponteiro para uso futuro (GAMMA 2007). Com esse padrão reduzimos o consumo de memória, e a complexidade de desenvolvimento em classes e objetos que possuem atributos que necessitam se manter imutáveis ao decorrer das interações de captura do vídeo. Um exemplo disto é a classe responsável por realizar os desenhos na tela. Estes desenhos só podem sofrer alguma alteração caso o usuário deseje apagar os desenhos ou adicionar novos. Caso nenhuma destas opções ocorram, é preciso manter o vetor de pontos que representa a anotação do usuário (ver Seção 4.5).

4.6.5 Problemas relacionados à usabilidade

A disposição das ações do software foram inicialmente dispostas conforme a Figura 22. Devido a esta organização, alguns usuários, em certos momentos, realizavam ações que não tinham a intenção de realizar. Na Figura 22 podemos ver que as ações de passar para o próximo slide e de apagar desenhos da tela estavam localizadas na borda inferior, nas regiões 4 e 5, respectivamente.

Figura 22 – Modelagem inicial de divisão das ações do usuário dispostas na tela.



Fonte: Elaborada pelo autor

Vimos que em alguns casos, logo após o usuário realizar o desenho, o mesmo leva sua mão até a parte inferior da tela, apagando assim os desenhos realizados (região 5 da Figura 22). Também ocorria de o usuário levar a mão à parte correspondente a passar slide, quando na verdade o mesmo apenas desejava sair da tela e manter o mesmo slide. Devido ao resultado destes testes, decidimos realizar a reorganização das ações do usuário (ver Figura 5 e Figura 6). Podemos ver esta reorganização descrita na Seção 3 e em suas subseções.

4.7 Problemas encontrados e não resolvidos

4.7.1 Demoras relacionadas ao tempo de processamento

O software foi desenvolvido no modo single thread, o que traz a necessidade de uso de algumas otimizações no desenvolvimento dos algoritmos usados. Uma destas otimizações é o uso do padrão Singleton, descrito na Seção 4.6.4. Algumas ações e funções do software são custosas em termos de processamento, como o uso de GDI Plus utilizado na impressão das anotações na tela e das alterações na área de trabalho. Temos que realizar estes desenhos constantemente e esta ação tem um tempo de processamento elevado, principalmente para desenhar as anotações do usuário. Estas anotações são compostas de várias linhas e vários pontos armazenados. Estas linhas são desenhadas várias vezes por segundo.

Outro processo custoso é o de identificar a posição da mão do usuário. Este passo é realizado analisando por completo uma imagem que possui vários pixels. Então a cada iteração da captura do vídeo realizamos uma varredura na imagem de resultado do processamento de remoção de fundo (ver Seção 4.4) até conseguirmos encontrar a mão do usuário. Além destes processamentos citados temos vários outros que devem ser realizados em um curto intervalo de tempo, como a correção de perspectiva.

Devido a este volume de processamento realizado em cada iteração temos alguns problemas relacionados a atraso, como por exemplo, uma leve demora no momento de mudar a cor das bordas externas (que indicam o momento que o usuário pode ou interagir com a lousa) após realizarmos uma ação de mudança de slide. Outro problema que temos relacionado ao atraso é a movimentação do mouse.

A movimentação do mouse está com um atraso para acompanhar a mão do usuário devido a dois fatores: o processamento, explicando anteriormente, e a suavização. A suavização faz com que o mouse se mova apenas no ponto médio entre alguns pontos. Devido a isso, quando temos uma movimentação com pontos muito distantes uns dos outros poderemos perceber o atraso do mouse em relação à mão do usuário.

Propomos como trabalhos futuros adotar uma abordagem de desenvolvimento em multi-thread como forma de resolver estes problemas relacionados ao tempo de processamento e atrasos. Pode-se usar um thread para realizar a captura e processamentos da imagem, um segundo thread que tenha a responsabilidade de tratar as alterações na tela e movimentação do mouse. Para isso é necessário usar threads sincronizados, pois não podemos ter um thread funcionando independentemente do outro. Não podemos ter, por exemplo, o

thread responsável pela captura e processamento de imagem funcionado por um intervalo de tempo e só depois o segundo thread realizando as anotações. Isso faria com que o software não funcionasse corretamente.

4.7.2 Problemas relacionados à usabilidade

Em testes realizados com voluntários percebemos que leva certo tempo até o usuário ter noção de como usar o software. Devido a interface do software estar relacionada com a imagem projetada, o usuário em certos momentos tem dificuldade de realizar as ações do software, como a configuração inicial (Seção 3.1). Neste passo, o usuário deve realizar as funções e comandos do software pressionando certas teclas no teclado (Seção 3.8). Para tentar contornar este problema sugerimos como trabalho futuro a criação de uma interface gráfica baseada em janelas e botões, na qual o usuário poderia realizar as configurações iniciais do software.

Outra abordagem que poderia resolver estes problemas seria associar algumas das funções, tais como a de realizar anotação e parar de realizar anotação, ou as ações de mudança de slide, a uma API de Speech Recognition (reconhecimento de fala), onde poderíamos usar comandos de voz para ativar e desativar a caneta, por exemplo. Desta forma o desenho só seria realizado no momento em que o usuário realizasse o comando de voz.

Outra idéia que sugerimos é realizar a reconstrução 3D do usuário utilizando duas câmeras que pode ser posicionadas uma ao lado da outra, de forma a simular os dois olhos de uma pessoa. Existem algoritmos prontos na biblioteca OpenCV para este propósito. Com a reconstrução 3D pode-se ter a noção de profundidade da mão do usuário. De posse disso podemos identificar quando o usuário toca na superfície da lousa e associar movimentos e ações para este toque. Tocar na tela e manter pressionado por um tempo poderia significar iniciar uma anotação. Um toque rápido na tela pode significar uma mudança de slide. Caso o usuário toque na tela mais à esquerda, realiza a ação de voltar um slide, e caso o usuário toque na tela em uma posição mais à direita, realiza uma ação de avançar o slide. Estes são alguns exemplos de análises que podem ser realizadas utilizando o recurso de reconstrução 3D.

5 CONSIDERAÇÕES FINAIS

A proposta geral deste trabalho foi construir uma lousa digital de baixo custo, utilizando técnicas de visão computacional. A lousa foi construída e consiste em um software que transforma um computador comum com webcam em uma lousa digital. Ao fim do desenvolvimento podemos perceber que o objetivo principal foi atingido. Realizamos o desenvolvimento de uma lousa digital que tem algumas das funções mais comuns das lousas digitais presentes no mercado.

Parte dos trabalhos presente na literatura aborda o problema das lousas digitais de forma a compartilhar o que é escrito com o pincel comum na superfície do quadro. No entanto, nos dias de hoje os Smartfones com suas câmeras também realizam esta tarefa. Na nossa abordagem focamos em dar auxílio à fluidez de uma apresentação de slides, fazendo com que seja mais fácil realizar as transições de slide e realizar destaques.

Deixamos como propostas para trabalhos futuros as sugestões de resolução de problemas ainda existentes no nosso software, descritas na Seção Figura 22, tais como o desenvolvimento de uma interface gráfica que possa auxiliar o usuário a realizar as configurações iniciais do software (Seção 3.1) de forma mais intuitiva. Não se priorizou modelar uma interface amigável que compreendesse os passos iniciais de configuração da lousa, e devido a isso algumas ações do software estão atreladas a comandos no teclado, como explicado na Seção 3.8.

Consideramos que também atingimos os objetivos secundários do projeto, que são a movimentação do mouse, embora com um atraso significativo, e o objetivo de facilitar o compartilhamento das anotações realizadas usando a lousa. Para este último criamos a função de salvar captura de tela (Seção 3.7) onde realizamos um Screenshot da tela projetada e armazenamos no disco, possibilitando o usuário compartilhá-la entre os interessados.

REFERÊNCIAS

AMCHAM BRASIL BELO HORIZONTE. **O olho, o cérebro e visão - uma visão forense**. Panorama Saúde, Belo Horizonte, [200-?]. Disponível em <http://www.holhos.com/artigos/visao_forense.pdf>. Acesso em: 26 nov. 2013.

BASU, Anup; LI, Xiaobo. **Computer Vision:** Systems, Theory and Applications. Singapore: World Scientific, 1993.

BÉRARD, François. The magic table**:** Computer-vision based augmentation of a whiteboard for creative meetings. In: IEEE INTERNATIONAL CONFERENCE IN COMPUTER VISION, 2003, Nice. **Anais**… Nice: IEEE, 2003. Disponível em: < http://iihm.imag.fr/publs/2003/procam03_magictable_berard.pdf >. Acesso em: 25 set. 2013.

CONCI, Aura; AZEVEDO, Eduardo; LETA, Fabiana R. **Computação gráfica**: teoria e prática. Rio de Janeiro: Elsevier, 2008.
GAMMA, ERICH et al. **Padrões de Projetos:** Soluções Reutilizáveis. São Paulo: Bookman, 2007.

HARRIS, Chris; STEPHENS, Mike. A combined corner and edge detector. In: ALVEY VISION CONFERENCE. 1988. Manchester, UK. **Anais…** Manchester: Plessey, 1988. Disponível em: <http://csce.uark.edu/~jgauch/library/Features/Harris.1988.pdf>. Acesso em 20 nov. 2013.

KAEWTRAKULPONG, Pakorn; BOWDEN, Richard. An improved adaptive background mixture model for real-time tracking with shadow detection. In: EUROPEAN WORKSHOP ON ADVANCED VIDEO BASED SURVEILLANCE SYSTEMS, 2., 2001. **Anais…** Kluwer Academic Publishers, 2001. Disponível em: <http://info.ee.surrey.ac.uk/Research/VSSP/Publications/papers/KaewTraKulPong-AVBS01.pdf>. Acesso em: 13 ago. 2013.

KUNZ, Andreas; NESCHER, Thomas; KUCHLER, Martin. Collaboard: a novel interactive electronic whiteboard for remote collaboration with people on content. In: INTERNATIONAL CONFERENCE ON CYBERWORLDS. 2010. Singapore. **Anais...** Singapore: IEEE, 2010. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5656157&tag=1>. Acesso em: 13 ago. 2013.

LAGANIÈRE, Robert. **OpenCV 2 computer vision application programming cookbook**. Birmingham, UK: Packt Publishing, 2011. Disponível em: <http://books.google.com.br/books?id=OC7jc8zWjlkC> Acessado em: 30 ago. 2013.
MARQUES FILHO, OGÊ; VIEIRA NETO, Hugo. **Processamento digital de imagens.** Rio de Janeiro: Brasport, 1999. Disponível em: <http://pessoal.utfpr.edu.br/hvieir/download/pdi99.pdf>. Acesso em: 03 set. 2013.

MICROSOFT, **CreateDC function**, 2015 a. Disponivel em: <https://msdn.microsoft.com/pt-br/library/windows/desktop/dd183490(v=vs.85).aspx>. Acesso em: 14 mai. 2015

MICROSOFT, **EnumDisplayDevices function**. 2015 b. Disponível em: <https://msdn.microsoft.com/en-us/library/windows/desktop/dd162609(v=vs.85).aspx>. Acesso em: 14 mai. 2015.
MICROSOFT, **GDI+**, 2015 c. Disponivel em: https://msdn.microsoft.com/en-us/library/windows/desktop/ms533798(v=vs.85).aspx. Acesso em: 13 mai. 2015

MICROSOFT, **GetSystemMetrics function**, 2015 d. Disponivel em: <https://msdn.microsoft.com/en-us/library/windows/desktop/ms724385(v=vs.85).aspx>. Acesso em:13 mai. 2015.

MICROSOFT, **The Virtual Screen**, 2015 e. Disponivel em: <https://msdn.microsoft.com/en-us/library/windows/desktop/dd145136(v=vs.85).aspx>. Acesso em: 14 mai. 2015

NASCIMENTO, André Hermenegildo do. **Rastreamento de Objetos por Similaridade de Características:** uma Aplicação para Compressão e Indexação de Vídeos. 2011. 60 f. Monografia (Graduação em Engenharia da Computação) – Universidade de Pernambuco, Recife, 2011. Disponível em: < http://tcc.ecomp.poli.br/20111/Andre%20Hermenegildo_Versao_Final.pdf >. Acessado em: 25 nov. 2013.

OPENCV. Disponível em: < http://opencv.org >. Acesso em: 28 out. 2013.

REKIMOTO, Jun. SmartSkin: an infrastructure for freehand manipulation on interactive surfaces. **Letters CHI**, Minnesota, v. 4, n.1, p. 113-120, 2002. Disponível em: < http://dl.acm.org/citation.cfm?id=503397> Acesso em: 05 set. 2013.

RIOS, Luiz Romário Santana. **Visão Computacional**. Salvador: Departamento de Ciência da computação - Universidade Federal da Bahia, 2010. Disponível em: <http://homes.dcc.ufba.br/~luizromario/Apresentação%20de%20IA/Artigo%20(final).pdf> acesso em: 19 nov. 2013.

ROSTEN, Edward; DRUMMOND, Tom. **Machine learning for high-speed corner detection**. Cambridge: Departamento de engenharia, Universidade de Cambridge, 2006. Disponível em: < http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.60.3991 >. Acesso em: 17 out. 2013.

SHARP. PN-L602B - Lousa Interativa Digital. Disponível em: <http://www.sharpbr.net/produtos-comerciais/lousa-digital.aspx?produto=99>. Acesso em: 25 out. 2013.

SOARES, Christophe Pinto de Almenda. **LoCoBoard**: Low-Cost Interactive Whiteboard Using Computer Vision Algorithms. Fernando Pessoa: Unicersidade Fernando Pessoa, 2009. Disponível em: <http://bdigital.ufp.pt/bitstream/10284/1228/1/DM_ChristopheSoares.pdf>. Acesso em: 28 set. 2013.

WILSON, Andrew D. **PlayAnywhere: a compact interactive tabletop projection-vision system**. In: User interface software and technology, 18. ACM, 2005. Disponível em: <http://dl.acm.org/citation.cfm?id=1095047>. Acesso em: 28 set. 2013.

ZHANG, Zhengyou. **Computer vision technologies for remote collaboration using physical whiteboards, projectors and cameras**. In: COMPUTER VISION FOR INTERACTIVE AND INTELLIGENT ENVIRONMENT, 5. **Anais…** 2005. IEEE, 2005. Disponível em: < http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01623773 >. Acesso em: 16 ago. 2013.

APENDICES

APÊNDICE A – Documento de requisitos

Neste apêndice será apresentado o documento de requisitos utilizado no desenvolvimento do projeto.

ESPECIFICAÇÃO DE REQUISITOS

1 Introdução

1.1 Objetivos

Este documento especifica os requisitos da lousa digital desenvolvida no projeto de TCC do aluno Paulo Filipe dos Santos Dantas, fornecendo as informações necessárias do projeto.

Glossário

Verbete	Definição
Frame	Um vídeo é composto de varias imagens, para cada imagem dar-se o nome de "Frame"
Tela Projetada	Área da lousa que contém apenas o que está sendo projetado do computador através do projetor
Anotação	Desenhos realizados pelo usuário na tela projetada.

2. Descrição geral do sistema

A lousa digital é uma ferramenta de auxilio em palestras e apresentações em geral. Utilizando esta ferramenta o usuário – em geral professores e palestrantes- poderá realizar anotações sobre o slide, bem como salvar capturas destes slides com ou sem anotações, realizar as ações básicas de mudança de slides (próximo e anterior). Todas estas ações serão realizadas apenas com analise dos movimentos do usuário na apresentação, descartando assim o uso de ferramentas adicionais e da interação física do usuário com o computador.

3. Premissas e restrições

Nesta seção descrevemos algumas premissas importantes para o uso do software. Caso alguma destas premissas forem ignoradas poderá ocorrer mau funcionamento do software.

Premissa 01: Na etapa de configuração o usuário deve verificar se o mesmo não está no vídeo capturado, caso esteja deve se mover o posicionar a câmera de forma que esta capture apenas a superfície onde é realizada a projeção.

Premissa 02: Após a configuração do software o usuário deverá trazer o software de apresentação (a janela no Windows que contem a apresentação) para o primeiro plano (foco do Windows nesta janela) de forma que as teclas de seta para direita e esquerda funcionem para realizar a mudança de slides. Isso pode

ser feito apenas clicando no item da barra de tarefas que corresponde ao software de apresentação.

Premissa 03: O software de apresentação deve realizar as mudanças de slide através das teclas de seta para direita e para esquerda, ou seja, caso o usuário não estivesse usando a lousa digital o software de apresentação de sua preferência deve realizar as mudanças de slide apenas clicando nas teclas de seta para direita e esquerda.

Premissa 04: O software funciona exclusivamente no sistema operacional Windows. Sendo este testado no Windows 7, 8, 8.1 e desenvolvido no Windows 7.

Premissa 05: O usuário não deve posicionar a mão na tela projetada nos momentos em que o software está armazenando a imagem de fundo.

Premissa 06: Os slides não devem conter cores escuras.

4. Requisitos do sistema

4.1 Requisitos Funcionais

[RF001] Mover o mouse

Descrição do caso de uso: O usuário deve ser capaz de movimentar o mouse apenas posicionando a mão sobre a tela projetada. O mouse deve seguir a mão do usuário de forma que o mesmo possa ver o mouse.

Prioridade: ■ ■ Essencial ☐ ☐ Importante ☐ ☐ Desejável

Entradas e pré-condições: O software estar devidamente configurado conforme as premissas descritas no item 3 (Premissas e restrições).

Saídas e pós-condição: O ponteiro do mouse é movimentado para a posição correspondente na tela

[RF002] Passar Slide

Descrição do caso de uso: O usuário deve ser capaz de avançar o slide da apresentação usada interagindo com a lousa de forma a retirar a sua mão da tela projetada .pela região especificada para esta função

Prioridade: ■ ■ Essencial ☐ ☐ Importante ☐ ☐ Desejável

Entradas e pré-condições: O software estar devidamente configurado conforme as premissas descritas no item 3 (Premissas e restrições).

Saídas e pós-condição: A apresentação avança para o próximo slide.

[RF003] Voltar Slide

Descrição do caso de uso: O usuário deve ser capaz de retroceder um slide da apresentação usada interagindo com a lousa de forma a retirar a sua mão da tela projetada .pela região especificada para esta função

Prioridade: ■ ■ Essencial ☐ ☐ Importante ☐ ☐ Desejável

Entradas e pré-condições: O software estar devidamente configurado conforme as premissas descritas no item 3 (Premissas e restrições).

Saídas e pós-condição: A apresentação volta para o slide anterior, caso o mesmo exista.

[RF004] Realizar uma anotação na tela

Descrição do caso de uso: O usuário deve ser capaz de realizar anotações na tela. Para isso o mesmo deve indicar que deseja iniciar uma anotação posicionando sua mão na região da tela projetada reservada para esta ação.

Prioridade: ■ ■ Essencial ☐ ☐ Importante ☐ ☐ Desejável

Entradas e pré-condições: O software estar devidamente configurado conforme as premissas descritas no item 3 (Premissas e restrições).

Saídas e pós-condição: O software solicita o ponto inicial de desenho

[RF005] Selecionar ponto inicial de anotação

Descrição do caso de uso: Após o usuário indicar que deseja realizar uma anotação na tela projetada o mesmo deve indicar o ponto inicial. Para isso o mesmo deve manter a mão parada por um intervalo de tempo até que o software indique que o mesmo pode iniciar o desenho.

Prioridade: ■ ■ Essencial ☐ ☐ Importante ☐ ☐ Desejável

Entradas e pré-condições: O usuário deve antes ter indicado que deseja iniciar uma anotação na tela. O software estar devidamente configurado conforme as premissas descritas na no item 3 (Premissas e restrições).

Saídas e pós-condição: O software indica que o desenho está sendo realizado e inicia faz as anotações com base na posição da mão do usuário.

[RF006] Parar anotação

Descrição do caso de uso: O usuário deve ser capaz de parar de realizar anotações na tela. Para isso o mesmo deve indicar que deseja parar uma anotação mantendo sua mão parada por um intervalo de tempo.

Prioridade: ■ ■ Essencial ☐ ☐ Importante ☐ ☐ Desejável

Entradas e pré-condições: O usuário deve antes estar realizando uma anotação. O software estar devidamente configurado conforme as premissas descritas no item 3 (Premissas e restrições).

Saídas e pós-condição: O software para de realizar anotações

[RF007] Apagar todas as anotação

Descrição do caso de uso: O usuário deve ser capaz de apagar todas as anotações realizadas na tela. Para isso o mesmo deve indicar que deseja realizar esta ação posicionando sua mão na região da tela projetada reservada para esta ação.

Prioridade: ■ ■ Essencial ☐ ☐ Importante ☐ ☐ Desejável

Entradas e pré-condições: O software estar devidamente configurado conforme as premissas descritas no item 3 (Premissas e restrições)..

Saídas e pós-condição: O software apaga todas as anotações realizadas na tela, caso existam.

4.2 Requisitos não Funcionais

[RNF001] Armazenar imagem de fundo da tela

Descrição do caso de uso: O software deve ser capaz de identificar e armazenar uma imagem que represente o fundo da cena.

Prioridade: ■ ■ Essencial ☐ ☐ Importante ☐ ☐ Desejável

Entradas e pré-condições: Premissa 05: .

Saídas e pós-condição: O software armazena uma imagem que represente o fundo da cena.