UNIVERSIDADE FEDERAL DO PAMPA

Maicon Robson da Silva Alves

Uma Biblioteca Para o Desenvolvimento de Aplicações de Projeção Interativa

Alegrete
2018

Maicon Robson da Silva Alves

Uma Biblioteca Para o Desenvolvimento de Aplicações de Projeção Interativa

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Marcelo Resende Thielo

Alegrete
2018

Maicon Robson da Silva Alves

Uma Biblioteca Para o Desenvolvimento de Aplicações de Projeção Interativa

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Trabalho de Conclusão de Curso defendido e aprovado em 26 de junho... de 2018
Banca examinadora:

Prof. Dr. Marcelo Resende Thielo
Orientador
UNIPAMPA

Profª. Drª. Amanda Meincke Melo
UNIPAMPA

Prof. Me. Eliezer Soares Flores
UNIPAMPA

À Letícia Rocha, pessoa com quem amo partilhar a vida. Obrigado pelo carinho, paciência, incentivo e por sua capacidade de me trazer paz na correria de cada semestre.

AGRADECIMENTOS

À esta universidade, seu corpo docente, direção e administração que oportunizaram minha formação.

Ao meu orientador Prof. Dr. Marcelo Thielo, pela orientação e apoio na elaboração deste trabalho.

Agradeço à minha mãe Marice, que me deu apoio, incentivo nas horas difíceis, de desânimo e cansaço.

Agradeço aos meus avós maternos, que sempre estiveram presentes ao longo de minha caminhada.

Agradeço à minha família e amigos, que nos momentos de minha ausência dedicados ao estudo superior, sempre fizeram entender que o futuro é feito a partir da constante dedicação no presente.

E a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

Do not go gentle into that good night,
Old age should burn and rave at close of day;
Rage, rage against the dying of the light.
(Dylan Thomas)

RESUMO

Desde a adoção do par teclado/monitor como dispositivo padrão de entrada e saída, diversas novas formas de interação com computadores foram criadas, sendo que uma das mais promissoras atualmente é a projeção interativa. Com a projeção interativa, é possível exibir uma imagem e transmitir instruções a um computador através de reconhecimento de mãos, objetos, detecções de toques, sombras, gestos, entre outros. Com isso, torna-se possível o desenvolvimento de aplicações que interpretem entradas de uma câmera ou outro dispositivo auxiliar, bem como exibam, com o auxílio de um projetor, imagens sensíveis a estas instruções sobre superfícies planas ou curvas. Embora soluções deste tipo já existam individualmente, combiná-las para desenvolver aplicações de projeção interativa ainda é uma tarefa que demanda conhecimento profundo sobre os algoritmos envolvidos e a escrita de muitas linhas de código. O objetivo deste trabalho é facilitar o desenvolvimento de aplicações de projeção interativa, através do desenvolvimento de uma biblioteca que combine diferentes soluções de entrada e saída, reunidas em funcionalidades acessíveis e fáceis de serem utilizadas. A biblioteca desenvolvida utiliza recursos disponíveis na biblioteca de visão computacional OpenCV. Funções já existentes foram unidas a algoritmos encontrados na literatura da área para compor novas funções direcionadas para aplicações de projeção interativa. Como resultados obtidos neste trabalho, temos uma biblioteca desenvolvida em C++, baseada em OpenCV e implementações próprias, que integra os recursos desta com os recursos que implementamos, no sentido de simplificar o desenvolvimento de aplicações de projeção interativa. Por fim, avaliamos sua validade, desenvolvendo uma aplicação piloto para demonstrar o uso das suas funcionalidades. Com o uso desta biblioteca, foi possível construir uma aplicação de projeção interativa com um número menor de linhas de código e uma menor complexidade. Dessa forma, é possibilitado ao usuário uma diminuição nos esforços de programação para integrar projetor e câmera na captura de comandos e geração de imagem, respectivamente, em aplicações interativas. Como trabalhos futuros, temos em vista a otimização dos métodos desenvolvidos para detecção da interação com a projeção, bem como a criação de novos métodos para ampliar a cobertura dos requisitos descobertos durante a pesquisa bibliográfica.

Palavras-chave: Computação Visual. Projeção Interativa. OpenCV. Biblioteca.

ABSTRACT

Since the adoption of the keyboard/monitor pair as the input and output standard device, several new forms of computer interaction have been created. Currently one of the most promising is the interactive projection. With interactive projection, it is possible to display an image and transmit instructions to a computer through hands recognition, objects, touch detection, shadows, gestures, and others. Therewith, it is possible to develop applications to interpret inputs from a camera, or other auxiliary device, as well as display, with the help of a projector, images that are sensitive to these instructions on flat or curved surfaces. Although solutions of this type already exist individually, combining it in order to develop interactive projection applications is still a task that demands a deep algorithm knowledge and writing many lines of code. This paper objective is to easy the development of interactive projection applications, throughout the development of a library which combines various input and output solutions gathered in accessible and easy-to-use functionalities. The developed library is based on resources available in the OpenCV computer vision library. Existing functions have been linked to the algorithms found in the field literature to create new functions directed to interactive projection applications. As result, we obtained a library developed in C ++, which integrates own-generated resources to the OpenCV in order to simplify interactive projections application development. Finally, as a functionality quality and validity check, we had developed a pilot application. With the use of this library it was possible to build an interactive projection application with a lower amount of code lines and complexity. Therefore, we provided a reduction in programming efforts to integrate a projector to a camera, to capture and image generation commands, respectively, into interactive applications. As future work, we aim to optimize the methods developed to detect the interaction with the projection, as well as the creation of new methods to extend the coverage of the requirements discovered during the bibliographic research.

Key-words: Visual Computing. Interactive Projection. OpenCV. Library.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Sistemas computacionais e interfaces são tecnologias em rápida difusão. Para que pessoas se beneficiem dessas inovações, é preciso que *designers* possuam alto conhecimento tecnológico e entendam a necessidade humana. Quando o conceito de interface apareceu, ele era entendido como o *hardware* e o *software* com o qual o homem podia se comunicar. A evolução deste conceito levou à inserção dos aspectos cognitivos e emocionais do usuário durante a comunicação. A interface é algo dito como discreto e tangível, é uma coisa que se pode desenhar, mapear, projetar e implementar, que pode ser encaixada posteriormente em um conjunto definido de funcionalidades (ROCHA; BARANAUSKAS, 2003).

A interação humano-computador se dá de diversas formas. O usuário precisa fornecer instruções ao computador, enquanto o mesmo exibe informações ao usuário. Os dispositivos para essa interação são chamados dispositivos periféricos (STALLINGS, 2009). Esses incluem: *mouse*, teclado, câmera, microfone, *touchpad*, *touchscreen*, monitor, projetor, impressora, entre outros.

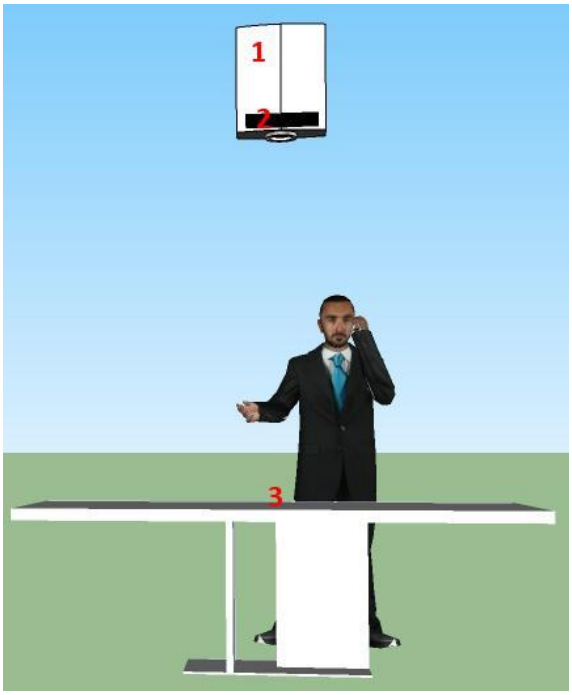
O *touchscreen* vem se tornando um importante e dominante recurso na interação com o computador, fornecendo ao usuário uma experiência a qual, através do toque na tela, permite arrastar itens, abrir arquivos e aplicações e executar diversas funcionalidades. Porém, a dimensão da superfície sensível ao toque é limitada ao tamanho da tela que, em dispositivos móveis, possui dimensões limitadas (3-6 polegadas) (HE; CHENG; TAO, 2014). Em computadores e *notebooks* essas dimensões, apesar de serem maiores, são limitadas pelo alto custo dessa tecnologia.

Projetores são cada vez mais utilizados no cotidiano das pessoas. Comparados a um *display* (dispositivo para apresentação de informações) tradicional, os projetores podem facilmente transformar superfícies planas em uma área de visualização com maiores dimensões (50 polegadas ou mais) sendo estes dispositivos pequenos e leves. Com a recente evolução tecnológica, os novos recursos de iluminação (LED, lâmpadas) proporcionam ao projetor maior performance a um menor custo. Isso acelera a popularização dos projetores no mercado (KAI et al., 2010).

Um projetor simples é um dispositivo de saída: imagens são projetadas na tela enquanto pessoas assistem passivamente o conteúdo exibido. A informação é um fluxo unidirecional do projetor para o usuário. Um sistema de projeção é chamado interativo quando não apenas exibe uma imagem, mas também recebe instruções e reage a elas em tempo real (KAI et al., 2010). Um exemplo de sistema de projeção interativa é exibido na Figura 1, o projetor (1) está apontado para superfície de projeção (3) e a interação é capturada pela câmera (2).

Alguns dos mais recentes sistemas de projeção interativa usam algoritmos de visão computacional para reconhecer sinais de entrada com câmeras ou com assistência de outros dispositivos (HE; CHENG; TAO, 2014; HA; KO, 2015; SHAH et al., 2011; LI et al., 2012; MELO et al., 2011). Alguns desses algoritmos podem ser implementados

Figura 1 – Exemplo de sistema de projeção interativa.



Fonte: Melo et al. (2011, p. 2).

com o uso do OpenCV (*Open Source Computer Vision*). OpenCV é uma biblioteca de visão computacional de código aberto[1]. A biblioteca é escrita em C e C ++ e pode ser executada em Linux, Windows e Mac OS X. Há desenvolvimento ativo em interfaces para Python, Ruby, Matlab e outras linguagens (BRADSKI; KAEHLER, 2008).

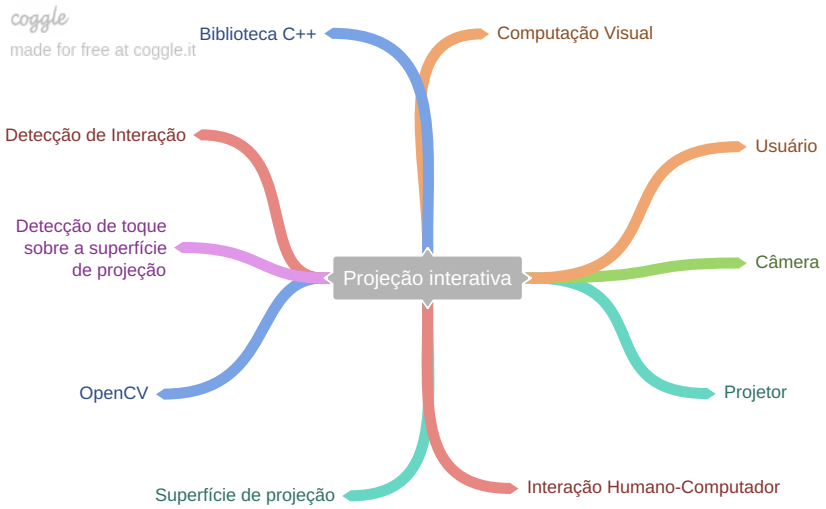
Uma biblioteca é um código, geralmente escrito por terceiros, que usamos para facilitar o desenvolvimento de uma aplicação (STROUSTRUP, 2012). Neste trabalho, nós propomos o desenvolvimento de uma biblioteca C++ baseada em OpenCV, para uso em aplicações de projeção interativa.

Na Figura 2, está exibida, em forma de mapa mental, uma visão geral dos temas e termos que foram exibidos durante o nosso trabalho. O mapa mental foi desenvolvido com uso da ferramenta Coggle[2].

Este capítulo apresenta, na seção 1.1, a motivação do desenvolvimento do trabalho. Na seção 1.2, estão os objetivos gerais e específicos deste trabalho. Na seção 1.3, é feita uma breve descrição de como o trabalho foi desenvolvido. Por fim, na seção 1.4, é apresentada uma visão geral sobre os demais capítulos.

[1] Disponível para download em https://sourceforge.net/projects/opencvlibrary/
[2] Disponível em https://coggle.it/

Figura 2 – Mapa mental: visão geral do trabalho.



Fonte: Elaborado pelo autor.

1.1 Motivação

Segundo Stroustrup (2012), uma biblioteca é simplesmente algum código – geralmente escrito por terceiros – que acessamos usando declarações encontradas em um arquivo que foi incluído. Segundo Deitel e Deitel (2011), usar funções e classes de bibliotecas em vez de escrever suas próprias versões pode melhorar o desempenho do programa, pois elas foram escritas para funcionar de modo eficiente. Essa técnica também reduz o tempo de desenvolvimento de programas. Os autores descrevem algumas vantagens e desvantagens de usar bibliotecas:

> A vantagem de criar suas próprias funções é que você saberá exatamente como elas funcionam. Você poderá examinar o código C++. A desvantagem é que isso consome muito tempo e esforço complexo, necessários para projetar, desenvolver e fazer a manutenção de novas funções e classes que sejam corretas e que operem de modo eficiente (DEITEL; DEITEL, 2011, p. 446).

> Na programação em C++, você normalmente usará os seguintes blocos de montagem: classes e funções da biblioteca padrão de C++, classes e funções que você e seus colegas criaram e classes e funções de várias bibliotecas populares de terceiros (DEITEL; DEITEL, 2011, p. 446).

Quando se desenvolve uma aplicação em alguma linguagem de programação, existe a possibilidade de incluir trechos de código que resolvem problemas específicos desenvolvidos por outros programadores. Dessa forma, é possível escrever códigos menos complexos e em menos linhas, dado que com o seu reuso, não é preciso replicar a solução que foi incluída.

Na área de pesquisa da projeção interativa, segundo averiguado nos trabalhos relacionados (Capítulo 2), existem soluções para o desenvolvimento de aplicações completas e precisas. Porém, para que essas soluções sejam aplicadas em sistemas, por desenvolvedores que não possuem conhecimento sobre o vasto campo de pesquisa, demandam tempo e esforço.

Portanto, o uso de uma biblioteca específica possui grande contribuição para o desenvolvimento de aplicações que usam a tecnologia de projeção interativa, pois permite acelerar o desenvolvimento extraindo complexidade e linhas de código escritas pelo programador.

1.2 Objetivos

O objetivo geral deste trabalho é desenvolver uma biblioteca em C++, baseada em OpenCV, para facilitar o desenvolvimento de aplicações de projeção interativa para um sistema composto por projetor, câmera CCD (ver seção 3.4), e superfície de projeção. Ver mais detalhes na seção 3.1. Como objetivos específicos, podem ser citados os seguintes:

a) utilizar funcionalidades já existentes em OpenCV para desenvolvimento das funcionalidades específicas para projeção interativa, reunindo métodos utilizados em trabalhos relacionados;

b) desenvolver uma aplicação de projeção interativa incluindo uso da biblioteca desenvolvida;

c) avaliar a complexidade e quantidade de linhas de código abstraídas do desenvolvimento com o uso da biblioteca desenvolvida.

1.3 Metodologia de Trabalho

Nesta seção, apresentamos a metodologia utilizada para atingir os objetivos do nosso trabalho, a saber:

a) **Investigar o estado da arte:** pesquisamos trabalhos relacionados com o contexto do trabalho através de um mapeamento sistemático;

b) **Avaliar trabalhos relacionados:** avaliamos trabalhos relacionados quanto a sua relevância para uso em nosso trabalho;

c) **Estudar conceitos ligados ao trabalho:** estudamos conceitos da biblioteca OpenCV, bibliotecas C++ e projeção interativa;

d) **Investigar quais funcionalidades desenvolver:** obtemos uma lista de exemplos de funcionalidades que poderiam desenvolvidas;

e) **Planejar desenvolvimento das funcionalidades:** obtemos um plano de desenvolvimento para as funcionalidades que foram desenvolvidas;

f) **Realizar etapa de desenvolvimento do trabalho:** nesta etapa, desenvolvemos as funcionalidades da biblioteca implementada;

g) **Executar e validar funcionalidades:** nesta etapa do trabalho, fizemos os testes necessários para garantir a qualidade das funcionalidades;

h) **Desenvolver aplicação exemplo:** nesta etapa, desenvolvemos a aplicação que contém as funcionalidades da biblioteca desenvolvida;

i) **Registrar e divulgar resultados do trabalho:** coletamos informações relevantes em cada uma das etapas e registramos no trabalho de conclusão de curso.

1.4 Organização do Documento

O restante do nosso trabalho está organizado da seguinte forma:

- Capítulo 2 **Trabalhos Relacionados**: relatamos a aplicação do mapeamento sistemático para investigação do estado da arte em projeção interativa e apresentamos os trabalhos relacionados selecionados no mapeamento sistemático;
- Capítulo 3 **Fundamentação Teórica**: abordamos conceitos ligados ao nosso trabalho, do ponto de vista de alguns autores e pesquisadores;
- Capítulo 4 **Planejamento do Trabalho**: apresentamos o planejamento de desenvolvimento do trabalho;
- Capítulo 5 **Desenvolvimento do Trabalho**: apresentamos o desenvolvimento do trabalho, suas funcionalidades e o modo como essas foram planejadas, bem como os resultados obtidos;
- Capítulo 6 **Considerações Finais**: apresentamos nossas considerações finais sobre o trabalho realizado e descrevemos possibilidades de trabalhos futuros.

2 TRABALHOS RELACIONADOS

Neste capítulo, apresentamos o Mapeamento Sistemático da literatura realizado para avaliar o estado da arte em sistemas de projeção interativa. Esse mapeamento foi executado conforme o guia proposto por Petersen et al. (2008). A seguir, na seção 2.1, apresentamos o protocolo de pesquisa do mapeamento. Na seção 2.2, apresentamos os resultados obtidos na pesquisa e por fim, na seção 2.3, apresentamos as considerações do capítulo.

2.1 Protocolo de Pesquisa

O propósito deste Mapeamento Sistemático é averiguar o estado da arte em sistemas de projeção interativa. Como inicio do mapeamento, definimos o escopo de pesquisa, buscando saber quais métodos de projeções interativas usam unicamente uma câmera e algoritmos de computação visual para capturar instruções. E também, quais soluções em OpenCV são utilizadas para detecção, rastreamento, e orientação de objetos, planos e formas. De acordo com essas dúvidas levantadas, definimos as seguintes questões:

- **Q1:** *Quais métodos são utilizados na projeção interativa baseada em câmera e projetor?*

- **Q2:** *Quais métodos são utilizados para detecção e rastreamento de objetos ou interação humana?*

- **Q3:** *Quais métodos são utilizados na detecção e identificação de orientação de planos?*

Iniciamos o processo de busca a partir da definição da base de dados. Utilizamos o *Scopus*[1] como base de dados por agregar um grande número de publicações, e possuir outras bases de pesquisa, como *Institute of Electrical and Electronics Engineers* (IEEE) e *Association Computing Machinery* (ACM) *Digital Library*. A seguinte chave de pesquisa foi utilizada em uma busca avançada:

TITLE-ABS-KEY (Interactive PRE/0 projection AND ("system"OR "camera"OR "vision"OR "method"OR "opencv")) OR TITLE-ABS-KEY(opencv AND ("object detection"OR "object tracking"OR "plane detection"OR "plane orientation")) AND (LIMIT-TO(SUBJAREA,"COMP"))[2]

Para filtragem dos trabalhos e eliminação de falsos-positivos, utilizamos os seguintes critérios de seleção:

[1] Disponível em www.scopus.com.
[2] Significa que a busca será realizada no título, resumo e palavras-chave dos trabalhos.

- **Critérios de inclusão:**
  - Trabalho que propõe uma solução para projeção interativa com uso de câmera e computação visual;
  - Trabalho que apresenta uso de OpenCV na resolução de problemas relacionados à detecção e rastreamento de objetos e planos.
- **Critérios de exclusão:**
  - Trabalhos que, além do uso de câmera e computação visual, utilizam dispositivos extras ou outras formas de captação de instrução;
  - Trabalhos que não possuam descrição dos algoritmos e métodos utilizados;
  - Trabalhos com menos de 4 páginas.

Realizamos a leitura dos títulos e resumos dos trabalhos encontrados pela pesquisa a fim de classificá-los e agrupá-los. Em casos em que o resumo não foi suficiente para o entendimento do trabalho, fez-se necessária uma leitura mais aprofundada. Para cada trabalho, adicionamos uma identificação e agrupamos de acordo com as questões de pesquisa respondidas.

2.2 Análise dos Resultados da Pesquisa

A pesquisa preliminar dos trabalhos, utilizando a chave de busca em uma pesquisa avançada, retornou 119 trabalhos relacionados. Após a aplicação dos critérios de filtragem, 8 trabalhos foram selecionados para compor o mapeamento sistemático. Após leitura aprofundada dos trabalhos, consideramos adicionar ao mapeamento sistemático algumas referências dos trabalhos listados pela pesquisa na base *Scopus*. Consideramos que as questões podem ser respondidas independentemente, não sendo obrigatório que se responda ambas em um mesmo trabalho.

Na Tabela 1, listamos o nome dos artigos que foram selecionados, seu(s) autor(es), ano de publicação e quais as questões de pesquisa que cada trabalho respondeu, em ordem cronológica, do mais recente ao mais antigo.

Em seguida, respondemos as questões de pesquisa com base na análise dos artigos.

Tabela 1 – Artigos selecionados no mapeamento sistemático.

Título do trabalho	Autor(es)	Ano	Q1	Q2	Q3
Comparison of tracking algorithms implemented in OpenCV	Janku et al. (2016)	2016	–	x	–
Fingertip-based interactive projector-camera system	Cheng et al. (2015)	2015	x	–	–
Visual tracking: An experimental survey	Smeulders et al. (2014)	2014	–	x	–
Touch-sensitive interactive projection system	He, Cheng e Tao (2014)	2014	x	x	x
A geometric correction method of plane image based on OpenCV	Xiaopeng et al. (2014)	2014	–	–	x
Projection center calibration for a co-located projector camera system	Amano (2014)	2014	x	–	x
Real-time continuous geometric calibration for projector-camera system under ambient illumination	Li et al. (2012)	2012	x	–	x
Hand gesture based user interface for computer using a camera and projector	Shah et al. (2011)	2011	x	–	–
Vision-based projected tabletop interface for finger interactions	Song et al. (2007)	2007	x	x	–
A Flexible New Technique for Camera Calibration (Technical Report)	Zhang (2002)	2002	–	–	x

Fonte: resultado do mapeamento sistemático.

2.2.1 Quais métodos são utilizados na projeção interativa baseada em câmera e projetor?

Contextualizado pela dificuldade de cobrir a informação 3D da ponta do dedo com uma única câmera, os autores Cheng et al. (2015) propõem um método de detecção de toque na tela de projeção com uso de algoritmos para identificação e detecção do toque na superfície de projeção. Primeiro, é identificado o primeiro plano com a extração da imagem prevista gerada pela calibração geométrica e fotométrica, depois, a ponta do dedo é identificada pela curvatura da mão e pontos de contorno, e por último, o toque é detectado pelo sistema de visão ativa. A detecção do primeiro plano inclui uma correção geométrica da imagem gerada pelo projetor e obtida pela câmera.

O objetivo do trabalho de Cheng et al. (2015) é semelhante ao nosso, pois detectar o toque na superfície de projeção é uma funcionalidade básica de um sistema de projeção interativa. Deste modo, nos baseamos em partes deste método para compor

funcionalidades da nossa biblioteca.

O método proposto por He, Cheng e Tao (2014) é semelhante ao de Cheng et al. (2015). Ambos possuem as mesmos objetivos: detectar o toque na superfície de projeção e identificar a localização deste toque. Porém, este utiliza métodos diferentes para identificação do toque, que são os algoritmos FAST (*Feature Accelerated Segment Test*) e BRISK (*binary robust invariant scalable keypoint*).

Os autores Shah et al. (2011) propõem um método de interação humano computador baseado no reconhecimento de gestos da mão com uma webcam e um projetor portátil. O projetor projeta a tela na parede ou outra superfície plana. O usuário pode interagir com a tela de projeção usando o rastreamento em tempo real das pontas dos dedos pela câmera usando o método *Camshift*, que é um método para detectar e reconhecer gestos de um único movimento dos dedos, que são traduzidos em ações. Essa proposta é de grande utilidade para aplicações de projeção interativa.

2.2.2 Quais métodos são utilizados para detecção e rastreamento de objetos ou interação humana?

Existem muitos algoritmos rastreadores propostos na literatura. O rastreamento de objetos no cenário real é um problema difícil e continua a ser estudado na computação visual. O trabalho dos Smeulders et al. (2014) analisa 19 dos principais algoritmos de rastreamento em 315 vídeos que possuem características de contexto real. Além da detecção do toque, que é proposta por Cheng et al. (2015), Song et al. (2007) e mais alguns, também existe a possibilidade de rastrear objetos sobre a projeção. Esse artigo cita e avalia algoritmos de rastreamento utilizados em muitos outros trabalhos.

No trabalho de Song et al. (2007), é proposto um método de detecção do toque na projeção a partir da distância da forma da mão à sua sombra. Ambas são identificadas e o toque do dedo na projeção ocorre quando a imagem da mão e sua sombra se unem em uma única forma. Há a necessidade de a câmera não estar situada próxima ao projetor para que seja gerada uma sombra na projeção. A detecção do toque não é totalmente precisa, mas é um método simples e durante o desenvolvimento do nosso trabalho cogitamos o seu uso.

O trabalho de Janku et al. (2016) é uma comparação dos algoritmos de rastreamento implementados em OpenCV. Os algoritmos SURF, SIFT, ORB e outros quatro algoritmos estão incluídos. Esses algoritmos possuem boa performance em rastreamento de objetos e são uns dos mais utilizados na literatura. Para rastreamento de componentes interativos (objetos) no sistema de projeção, é necessário que se tenha um bom conhecimento desses algoritmos. O trabalho de Janku et al. (2016) referencia outros artigos que possuem desenvolvimento desses algoritmos, portanto, é uma boa base de informação.

2.2.3 Quais são os métodos utilizados na detecção e identificação de orientação de planos?

O sistema de projeção interativa precisa ser preciso quanto à identificação da localização dos pontos na projeção. O artigo escrito por Amano (2014) propõe um alinhamento interativo de posições para centros de projeção. É usado o gabarito da câmera e projetor para visualizar o desalinhamento dos centros. Uma vez que o desalinhamento gera um padrão de distância entre os *pixels*, é possível obter um alinhamento altamente preciso.

No trabalho dos autores Xiaopeng et al. (2014), é proposto um método de correção geométrica do plano da imagem para posição desconhecida da câmera com uso de OpenCV. Com esse método, é possível corrigir distorções e perspectiva do plano da imagem. Na projeção interativa, a projeção da imagem pode sofrer distorções se a superfície de projeção não estiver ortogonal ao projetor. O conteúdo desse artigo nos ajudou no desenvolvimento de métodos de correção de perspectiva e distorções da imagem projetada.

No trabalho de Li et al. (2012), é proposto um método rápido e contínuo de calibração geométrica para o sistema de projetor e câmera sobre condições de iluminação ambiente. O método estima um tempo de exposição apropriado pra prevenir degradação na imagem capturada e utiliza ORB (Oriented FAST and Rotated BRIEF) para correlacionar pares de características em tempo real. A proposta desse artigo é semelhante à proposta de Xiaopeng et al. (2014).

O trabalho de Zhang (2002) foi adicionado ao mapeamento por ser citado por Cheng et al. (2015), Amano (2014) e Xiaopeng et al. (2014). Ele propõe um método de calibração de câmera que requer apenas a câmera e uma imagem plana vista de ângulos diferentes. Então, a câmera ou a imagem podem ser movidos livremente. O procedimento proposto consiste em uma solução fechada, seguida de um refinamento não linear com base no critério de máxima verossimilhança.

2.3 Considerações do Capítulo

Através do mapeamento realizado, descobrimos que o desenvolvimento de aplicações de projeção interativa baseada em visão computacional é pouco disseminado dada sua complexidade, enquanto a quantidade de aplicações baseadas em câmera de profundidade, como o Microsoft Kinect é bem maior.

Um sistema de projeção interativa pode possuir diversas funcionalidades. Uma das funcionalidades mais básicas é a correção geométrica da captura da projeção, visto em Cheng et al. (2015). A correção geométrica da captura da projeção faz com que a projeção seja exibida de forma que existam pontos correspondentes na imagem projetada e na imagem capturada pela câmera.

A detecção de toque sobre a projeção é uma funcionalidade presente nos trabalhos de He, Cheng e Tao (2014), Ghys (2013), Song et al. (2007), Shah et al. (2011), Harrison,

Benko e Wilson (2011), e outros. O sistema tem a capacidade de detectar uma mão ou dedo sobre a superfície e detectar se ocorre um toque na superfície de projeção. O sistema responde a este toque exibindo conteúdos na projeção ou executando ações sobre os conteúdos exibidos, como: clicar, pegar ou arrastar objetos, desenhar, e selecionar conteúdos.

Outras funcionalidades interessantes são exibidas no trabalho de Ziola et al. (2011), em que é possível identificar um objeto sobre a superfície de projeção, inferir uma ação a este objeto e exibir conteúdos na projeção de acordo com a função do objeto. Por exemplo, um dragão de Lego pode ativar a exibição de fogo sobre outros objetos.

Os principais aprendizados obtidos através de nossa pesquisa foram: é preciso demandar tempo e esforço para o aprendizado de algoritmos de visão computacional; uma aplicação de projeção interativa pode ser robusta e confiável sem o uso de informações de profundidade; uma ferramenta de desenvolvimento pode impulsionar a criação de novas aplicações.

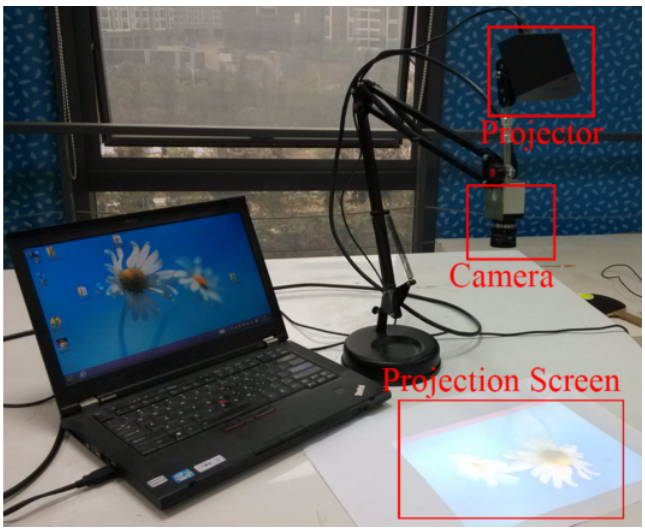
3 FUNDAMENTAÇÃO TEÓRICO-METODOLÓGICA

Neste capítulo são apresentados os fundamentos teóricos necessários para o entendimento do nosso trabalho. O capítulo está organizado da seguinte forma: na seção 3.1 nós apresentamos os conceitos ligados à projeção interativa; na seção 3.2 apresentamos os diferentes mecanismos utilizados para a captura de imagem na projeção interativa; na seção 3.3 exibimos um exemplo de sistema de projeção interativa; na seção 3.4 exibimos as principais aplicações da projeção interativa; na seção 3.5 apresentamos algumas características de bibliotecas C++; na seção 3.6 apresentamos a conceitualização da visão computacional; na seção 3.7, uma descrição do OpenCV; na seção 3.8 apresentamos alguns conceitos sobre metodologias ágeis; por fim, na seção 3.9, apresentamos as considerações do capítulo.

3.1 Projeção Interativa

A combinação do projetor e câmera proporciona não só uma experiência unidirecional exibindo uma imagem projetada para o usuário, mas também um método de entrada que permite que usuários interajam com a projeção. Os sistemas de projeção interativa proporcionam uma interface humano-computador realista e intuitiva. Sem uso de dispositivos de cliques, é possível interagir com uma projeção tocando ou pondo objetos na superfície em que a imagem é projetada. Os sistemas de projeção interativa atuais utilizam algoritmos de visão computacional para detectar a interação humana com câmeras ou outros dispositivos auxiliares.

Figura 3 – Sistema de projeção Interativa proposto por He, Cheng e Tao (2014).



Fonte: He, Cheng e Tao (2014, p. 2).

3.2 Diferentes Mecanismos de Captura Utilizados na Projeção Interativa

Durante o processo de pesquisa, encontramos diferentes mecanismos de captura utilizados na projeção interativa com custos e complexidades diferentes.

Entre os trabalhos de pesquisadores da área, são encontradas pesquisas sobre captura e detecção de interação com uso de câmeras CCD (*Charge-Coupled Device*). As câmeras CCD possuem um sensor semicondutor para captação de imagens formado por um circuito integrado que contém uma matriz de capacitores acoplados. Através de lentes, uma imagem é projetada na matriz de capacitores (a região fotoativa) e faz com que cada capacitor acumule uma carga elétrica proporcional à intensidade da luz incidente naquela localização (CID; JR; MICHEL, 2002). Para uso de câmeras CCD na captura e detecção de interação, são necessários algoritmos específicos e um processamento maior de informações, já que é uma tarefa complexa obter informação de distância a partir de uma única câmera. Porém, o baixo custo de câmeras CCD comparado a outras tecnologias é um atrativo para os pesquisadores, pois possibilita desenvolvimento de aplicações complexas a um baixo custo.

Também são encontradas pesquisas sobre projeção interativa baseadas no uso de câmeras RGB-D, que são capazes de obter informações sobre distância ou profundidade. Os dispositivos RGB-D combinam informações de cores RGB com informação de profundidade por pixel. O dispositivo projeta um feixe de infravermelho, que é capturado por uma câmera infravermelha e comparado parte a parte aos padrões de referência armazenados no dispositivo que possuem informação de profundidade já conhecidas. O sensor, então, calcula a profundidade por pixel com base em quais padrões de referência o padrão projetado corresponde melhor (LITOMISKY, 2012). Esta tecnologia torna menos complexa a tarefa de obter informações sobre distância de objetos e possibilita detectar, por exemplo, se um objeto toca a superfície de projeção ou não. Porém, o custo dessa tecnologia é mais elevado se comparado às câmeras CCD.

Outros métodos de detecção de interação foram encontrados, como sensores de toque com coordenadas 2D. Estes fogem do objetivo do nosso trabalho que é obter informação com uso de algoritmos de computação visual, e não serão descritos.

Nas seções que seguem, são descritos os mecanismos de detecção de interação separados por tipo de dispositivo: câmera CCD e sensores de profundidade.

3.2.1 Captura por câmera CCD

Entre os mecanismos de detecção de interação por câmera CCD, está o método proposto por Shah et al. (2011). Ele propõe o reconhecimento da mão baseado na cor da pele. Uma base de dados é formada com pixeis tirados de 50 imagens de pessoas diferentes, sob diferentes condições de iluminação. Um pixel é classificado como pixel de pele ou não de acordo com a similaridade aos pixeis da base de dados. Uma vez que a

mão é detectada, um contorno contendo a mão é selecionado como região de interesse e é passado para a função *Camshift* para rastrear a mão nos *frames* futuros. O algoritmo de rastreamento *Camshift*, disponível no OpenCV, permite rastrear objetos em movimento avaliando o tamanho e forma do objeto.

No trabalho de Song et al. (2007), é proposto um método de interação baseado na distância da sombra do dedo ao dedo em si. O dedo e sua sombra são detectados na imagem capturada, ambos são classificados como região de interesse. Quando as duas regiões de interesse (o dedo e sua sombra) se unem em uma só forma, é assumido que ocorreu um toque na superfície de projeção.

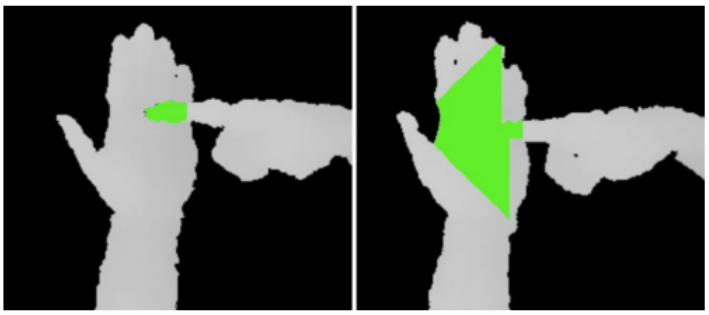
O trabalho de Ha e Ko (2015) exibe também um método de interação baseado na sombra da mão, porém, não implementa a detecção de toque na superfície de projeção. A interação é baseada em reconhecimento de gestos, que quando reconhecidos acionam uma determinada ação, como desenhar, arrastar, tocar.

Métodos semelhantes aos exibidos na seção 3.3 compõem trabalho de Cheng et al. (2015). A diferença é o método de detecção de toque na superfície de projeção, que é feito a partir de uma triangulação sobre a imagem prevista e a imagem capturada projetando um circulo próximo à ponta do dedo.

3.2.2 Captura com auxílio de sensor de profundidade

No trabalho de Harrison, Benko e Wilson (2011), é utilizada uma câmera de profundidade PrimeSense[1], que provê um mapa de profundidade 320x240 a 30 FPS. Esse sensor consegue mapear objetos à uma distância de 20cm com erro aproximado de 5mm no eixo de profundidade Z. Os autores apresentam um método de rastreio dos dedos que possibilita múltiplos toques em qualquer superfície, mesmo que irregular, sem calibração ou treino.

Figura 4 – Detecção do toque utilizando sensor de profundidade.



Fonte: Harrison, Benko e Wilson (2011, p. 3).

[1] PrimeSense Ltd. http://www.primesense.com.

É possível calcular a posição dos dedos nos eixos X, Y e Z, e saber se estão tocando ou flutuando sobre a superfície. Para detectar os dedos, primeiro é computado o ponto central do comprimento do dedo, que equivale aproximadamente a junta menor. A partir deste ponto, os *pixels* do dedo são preenchidos com cor. Esta operação é realizada no mapa de profundidade usando uma tolerância de 13mm para determinar se os *pixels* vizinhos podem ser preenchidos. Quando o dedo está pairando na superfície, o preenchimento ocorre apenas no dedo. No entanto, se o dedo entra em contato com a superfície, a operação de preenchimento também atinge o objeto de contato. Se um limite de *pixels* preenchidos é atingido, a operação é interrompida e o toque é detectado. O processo de preenchimento é exibido na Figura 4.

3.3 Um Exemplo de Sistema de Projeção Interativa

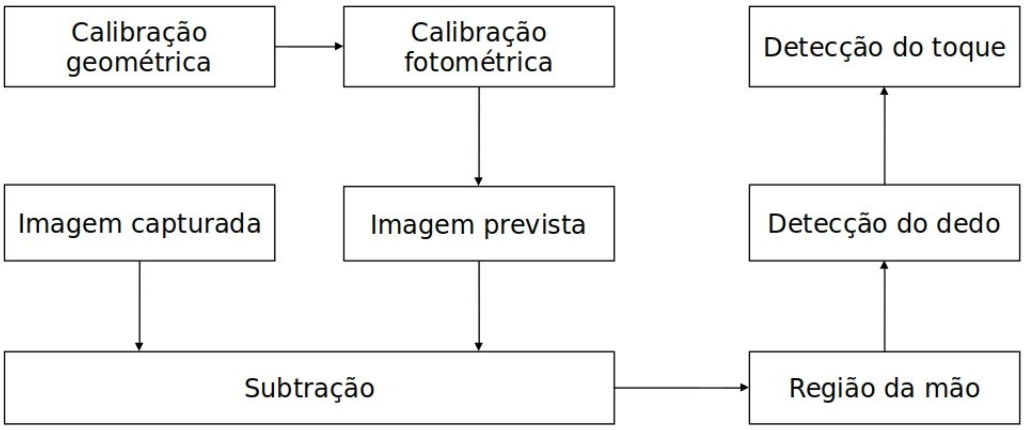
Para exemplificar o funcionamento de um sistema de projeção interativa, apresentamos o sistema proposto por He, Cheng e Tao (2014), exibido na Figura 3. O sistema provê uma interação com o computador usando o toque dos dedos na superfície projetada. É composto por um projetor portátil, uma câmera, e uma superfície plana de projeção.

O projetor e a câmera são fixados rigidamente em uma estrutura que fica sobre a mesa a uma distância aproximada de 45cm, projetando uma tela de 14", que pode ser ajustada manualmente para alterar sua dimensão. Como o projetor e a câmera estão em uma posição fixa, é possível prever o conteúdo que a câmera deveria capturar pela calibração geométrica e fotométrica. A imagem capturada e a imagem prevista terão uma grande diferença se houver uma mão sobre a superfície de projeção, esta diferença pode ser utilizada para identificar onde a mão está. Com essa informação, as pontas dos dedos são identificadas pela curvatura da forma extraída. O toque é detectado usando os algoritmos FAST e BRISK, comparando a imagem prevista com a imagem capturada.

O algoritmo FAST (*Features from accelerated segment test*) é um método de detecção de cantos que pode ser usado para extrair pontos de interesse em uma imagem, que posteriormente podem ser usados para rastrear e mapear objetos (BIADGIE; SOHN, 2014). Já o algoritmo BRISK (*Binary Robust Invariant Scalable Keypoints*) é um método para descrição e correspondência de pontos chaves em duas ou mais imagens (LEUTENEGGER; CHLI; SIEGWART, 2011).

A sequência de passos do sistema proposto por He, Cheng e Tao (2014) para detectar o toque é exibida na Figura 5. Primeiro é feita a calibração geométrica entre a imagem capturada e a imagem projetada. Em seguida, é feita uma calibração fotométrica para estimar como a imagem capturada será dada uma imagem projetada qualquer, gerando então a imagem prevista. Uma subtração entre a imagem prevista e a imagem capturada destacam os *pixels* localizados na superfície da mão. Com a região da mão identificada, a detecção do dedo é feita. Por fim, o toque é identificado com o uso dos algoritmos FAST e BRISK.

Figura 5 – Visão geral do sistema de projeção interativa.



Fonte: adaptado de He, Cheng e Tao (2014, p. 3).

3.3.1 Preparação da imagem capturada

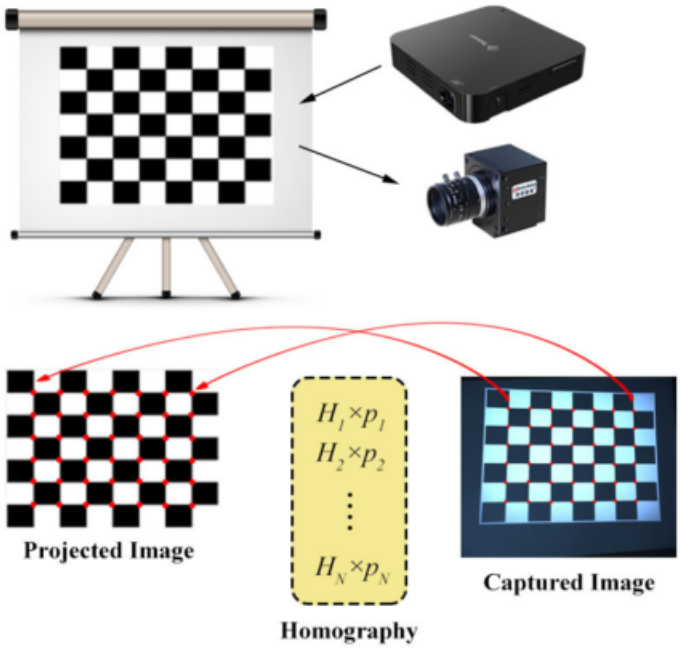
As imagens de uma superfície plana vistas de duas diferentes perspectivas são relacionadas por uma homografia. Também chamada de transformação projetiva planar, colineação ou projetividade, uma homografia descreve um mapeamento linear entre espaços planares bidimensionais (AIRES, 2009)

Para prever a imagem capturada pela câmera, é necessário saber a relação entre os *pixels* da imagem capturada e os *pixels* da imagem projetada. O processo de calibração homográfica é ilustrado na Figura 6. Essa relação pode ser descrita em uma matriz H_{3x3}, e, para computar a matriz H, é preciso conhecer alguns pontos correspondentes nas duas imagens (LIAO; YANG; ZHANG, 2008). Os principais passos são:

- projetar um tabuleiro de xadrez e capturá-lo simultaneamente utilizando a câmera fixada, como ilustrado na Figura 6;
- detectar os cantos do tabuleiro de xadrez na imagem capturada;
- usar os cantos detectados e suas posições conhecidas na imagem projetada para estimar a homografia entre a tela projetada e a imagem plana capturada pela câmera.

Também é necessário, para prever a imagem capturada, saber para cada *pixel* capturado pela câmera, a cor do *pixel* correspondente na imagem prevista, pois, uma mesma cor na imagem projetada é diferente na imagem capturada. Esta diferença é causada por muitos fatores, incluindo iluminação ambiente, cor e material da superfície de projeção, variação da intensidade do projetor. A calibração fotométrica deve ser tanto

Figura 6 – Estimativa homográfica no sistema de projeção interativa.



Fonte: He, Cheng e Tao (2014, p. 3).

de cor quanto de posição. O método de calibração fotométrica proposto por Chen et al. (2008) é utilizado no trabalho de He, Cheng e Tao (2014), e é descrito a seguir:

- P_K: brilho do projetor para o canal K;
- C_L: brilho capturado pela câmera para o canal L.

O modelo fotométrico do sistema de câmera e projetor para cada pixel pode ser escrito como:

$$C = A(VP + F), \tag{3.1}$$

onde

$$C = \begin{bmatrix} C_R \\ C_G \\ C_B \end{bmatrix}, A = \begin{bmatrix} A_R & 0 & 0 \\ 0 & A_G & 0 \\ 0 & 0 & A_B \end{bmatrix}, F = \begin{bmatrix} F_R \\ F_G \\ F_B \end{bmatrix}, V = \begin{bmatrix} V_{RR} & V_{GR} & V_{BR} \\ V_{RG} & V_{GG} & V_{BG} \\ V_{RB} & V_{GB} & V_{BB} \end{bmatrix}, P = \begin{bmatrix} P_R \\ P_G \\ P_B \end{bmatrix}$$

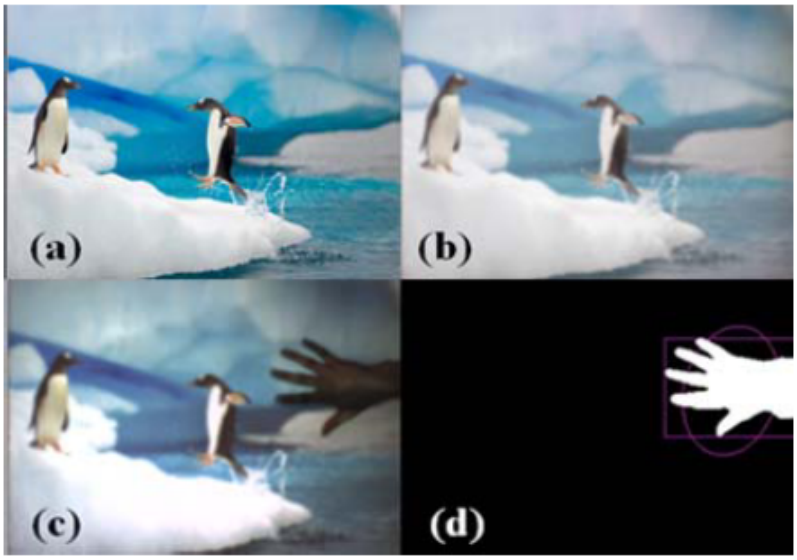
A matriz A_{3x3} é a reflectância da imagem gerada pelo projetor na superfície, a matriz V_{3x3} é a matriz de mixagem de cores, e o vetor F é a influência das condições de iluminação. O objetivo é calcular A, V e F. Os passos são:

- para cada *pixel* na imagem projetada, usar H para estabelecer a correspondência entre a imagem capturada pela câmera e a imagem que o projetor exibe na superfície;

- projetar uma imagem preta para calcular F;
- projetar vermelho puro, verde puro e azul puro para calcular a matriz V de mixagem de cor;
- projetar uma imagem branca para computar a matriz A.

Com esses passos, para uma imagem projetada é possível calcular a imagem prevista usando a Equação 3.1. Na Figura 7, (a) e (b) mostram a imagem projetada e a imagem de captura prevista. Depois de gerar a imagem prevista, é possível identificar a região da mão, ilustrado em (c) e (d).

Figura 7 – Detecção do contorno da região da mão.



Fonte: He, Cheng e Tao (2014, p. 4).

3.3.2 Detecção da mão

Quando há interação com a projeção, é alterada a imagem capturada e também o albedo da superfície de projeção. Detectar a mudança do albedo nos remete à extração da mão na imagem capturada calculando a variação do albedo na superfície de projeção e a superfície da mão. Albedo é a refletividade difusa ou poder de reflexão de uma superfície. Ela é a razão entre a radiação refletida pela superfície e a radiação incidente sobre ela.

Considerando que Q é a luz incidente, A é o albedo da superfície de projeção, T é a transformação de cor do pixel capturado pela câmera e C é o brilho capturado pela câmera, temos $C = A \times T \times Q$. Se nada estiver sobre a superfície de projeção a imagem I deve ser igual a C. Se há uma mão sobre a superfície de projeção, o albedo da superfície é alterado, denotado como A'. A imagem capturada pode então ser descrita

como $I = A' \times T \times Q$. Podemos calcular a proporção do albedo $a = A'/A$ para o pixel [x,y] no canal de cor $c \in \{R, G, B\}$ que é dado por

$$a_{[x,y,c]} = \frac{A'}{A} = \frac{I_{[x,y,c]}}{C_{[x,y,c]}}. \tag{3.2}$$

Baseado na proporção de albedo a, podemos detectar a mão. O albedo da superfície de projeção sem a presença da mão deve ser próximo de 1. Para um pixel [x,y], assumindo que a soma de relação de albedo dos três canais é $a_{[x,y,sum]}$. Para uma imagem, a média da relação total de albedo de três canais é $a_{[sum]}$.

Um pixel [x,y] está na região da mão se e somente se

$$a_{[x,y,R]} + a_{[x,y,G]} + a_{[x,y,B]} < s \times a_{[sum]} \tag{3.3}$$

onde s é uma escala tolerante da alteração do albedo, com valor típico de $0.5 \sim 0.8$. A Figura 7, exibe (c) imagem capturada pela câmera e (d) resultado da detecção da mão. Depois de conhecer a região da mão, é aplicado um método baseado na aparência da forma para detectar a localização da ponta do dedo.

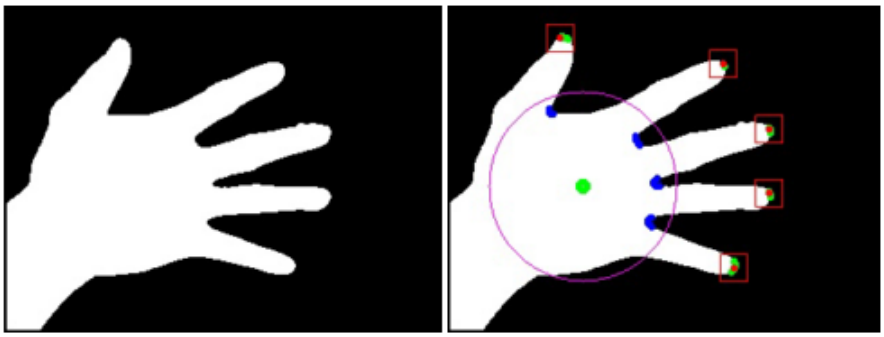
3.3.3 Detecção das pontas dos dedos

Cada curvatura do contorno da região da mão é filtrado como candidato à ponta do dedo com:

$$Cur(P_i) = \frac{P_iP_{i-\delta} \times P_iP_{i+\delta}}{\|P_iP_{i-\delta}\| \times \|P_iP_{i-\delta}\|}, \tag{3.4}$$

onde, $Cur(P_i)$ é a curvatura do ponto de contorno P_i, $P_{i-\delta}$ é o δ-ésimo ponto precedente à P_i e $P_{i+\delta}$ é o δ-ésimo ponto depois de P_i.

Figura 8 – Detecção das pontas dos dedos.



Fonte: He, Cheng e Tao (2014, p. 4).

Na Figura 8, é exibida a localização da ponta do dedo, sendo: imagem esquerda, o contorno da região da mão e imagem direita, a localização das pontas dos dedos marcadas com pontos vermelhos, que é implementada seguindo três passos:

- são marcados os pontos com curvatura maior que um limiar estático t (pontos verdes)
- são distinguidos os pontos nos dedos com maior distância do centro de gravidade do vale entre dois dedos calculando os momentos centrais da região da mão
- é calculada a localização média dos candidatos à dedo como a posição final das pontas dos dedos.

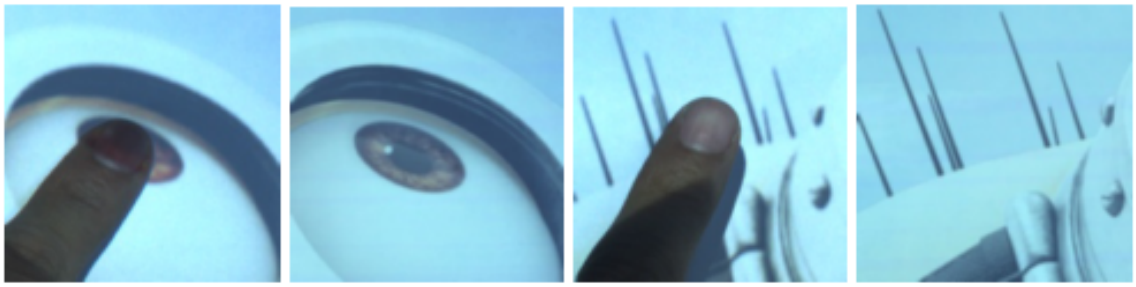
Com as pontas dos dedos localizadas, a próxima tarefa é detectar se alguma ponta do dedo está tocando a superfície de projeção. Estimar a distância das pontas dos dedos até a superfície de projeção é o ponto crítico para a detecção do toque em um sistema de projeção interativa com uso de câmera CCD. É preciso identificar os dados tridimensionais da distância do dedo à superfície de projeção com uma visão monocular.

3.3.4 Detecção do toque na superfície de projeção

O trabalho de He, Cheng e Tao (2014) propõe um método de detecção de toque através da detecção local baseada em FAST próximo às pontas dos dedos e descritores baseados em BRISK.

A Figura 9 exibe a detecção do toque na superfície, sendo a primeira e a segunda imagem respectivamente, imagem capturada e imagem prevista quando o dedo está tocando a superfície de projeção, e a terceira e quarta imagem, imagem capturada e imagem prevista quando o dedo está suspenso sobre a superfície de projeção. A aparência do conteúdo projetado, incluindo cor, forma e contorno dos cantos, é diferente quando a mão está sobre a superfície de projeção. A distorção do conteúdo projetado depende da distância entre a superfície da mão e a superfície de projeção. Os itens projetados nas mãos são quase coincidentes com a imagem prevista em (a), enquanto a distorção desses itens será óbvia se não houver contato, visualizado em (b). Com esta variação de conteúdo projetado é possível identificar se há o toque na superfície ou não, como visto a seguir.

Figura 9 – Detecção do toque.

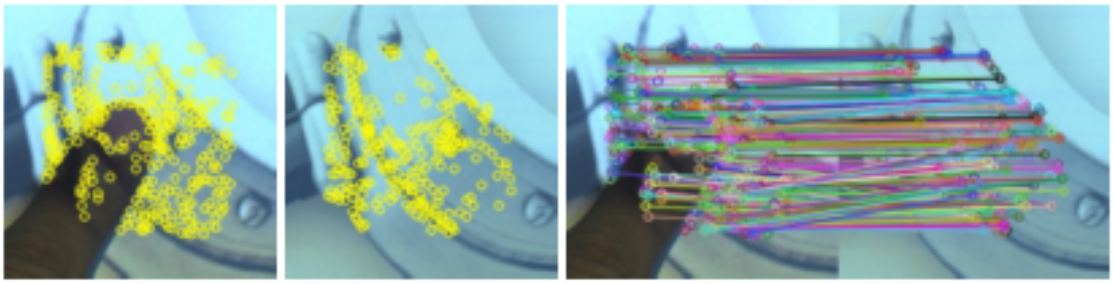


Fonte: He, Cheng e Tao (2014, p. 5).

A detecção FAST é usada em tempo real na taxa de quadros da aplicação para gerar uma alta eficiência. As imagens obtidas pela captura da câmera podem ser combinadas com as imagens previstas à *priori* depois da calibração geométrica e fotométrica. Para reduzir cálculos desnecessários, o autor delimitou a região de interesse em 200x200 *pixels* próximos à ponta do dedo. O detector baseado em FAST é usado para extrair os pontos chave do conteúdo projetado para combinar com a imagem prevista.

Para identificar todos *pixels* na imagem capturada como um ponto de interesse ou não, é preciso gerar um círculo de Bresenhan de 16 pixeis ao redor de P_i para todos *pixels* P_i na imagem capturada, em seguida comparar a intensidade dos *pixels* 1, 5, 9, e 13 do círculo com P_i. Se pelo menos três dos quatro valores de *pixel* estiverem acima ou abaixo do valor de P_i, então P_i é um ponto de interesse. O algoritmo de Bresenham é um algoritmo criado para o desenho de linhas em dispositivos matriciais. Ele permite determinar quais os pontos numa matriz de base quadriculada devem ser destacados para atender o grau de inclinação de um ângulo. Os pontos de interesse gerados pelo algoritmo FAST e a comparação com o algoritmo BRISK são exibidos na Figura 10.

Figura 10 – Identificação e comparação dos pontos de interesse.



Fonte: He, Cheng e Tao (2014, p. 5).

Os pontos de interesse na imagem capturada podem ser correlacionados aos mesmos pontos de interesse na imagem prevista. Considerando a resolução variada da câmera e do projetor, foi utilizado o BRISK como descritor para comparação. O descritor BRISK é composto por uma *string* binária (256 bits), que possui os resultados de testes de comparação de brilho simples concatenados. O descritor pode ser estimado para cada bit b correspondente à:

$$b = \begin{cases} 1 & I(P_j^\alpha, \sigma_j) > I(P_i^\alpha, \sigma_i) \\ 0 & \text{caso contrário} \end{cases}, \quad \forall (P_j^\alpha, P_i^\alpha) \in S \tag{3.5}$$

A correspondência de dois descritores BRISK é feita pelo cálculo da distância de Hamming, que pode ser calculado com uma alta eficiência em sistemas de tempo real. Depois que todos os pontos de interesse são correspondidos na imagem capturada e na

imagem prevista, a distância média entre esses pontos de interesse é estimada através do algoritmo de RANSAC, diminuindo a influência de valores atípicos causada por ruído ou descompasso. RANSAC é uma abreviatura de "*Random Sample Consensus*". É um método iterativo para estimar os parâmetros de um modelo matemático.

Se houver um dedo tocando na superfície de projeção, a distância entre os pontos de interesse da imagem capturada e a imagem prevista será inferior à um limite dinâmico t, que é definido pela distância média de todos os pontos correspondentes. Caso contrário, o dedo está suspenso sobre a superfície de projeção.

3.4 Aplicações da Projeção Interativa

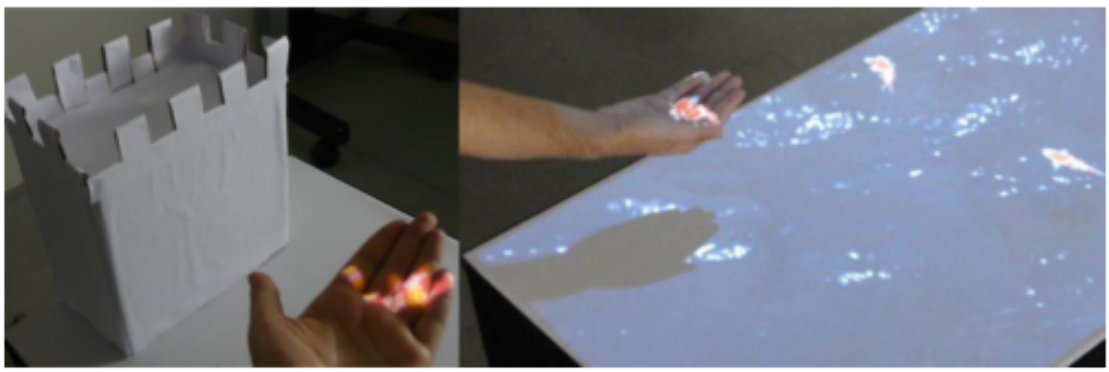
Uma vez que os sistemas de projeção interativa proporcionam um método de interação de forma vívida e atrativa a vários segmentos de mercado, eles têm sido utilizados para fins diversos, desde sistemas para auxiliar em apresentações e lecionar, até produtos de entretenimento, como jogos e brinquedos. Algumas dessas aplicações são apresentadas a seguir.

3.4.1 Educação

Com o intuito de criar um mecanismo educacional, de aplicação em espaços públicos para narrativas interativas para crianças, os autores Melo et al. (2011) propuseram um sistema interativo que pode tornar uma narrativa mais divertida e atraente. Usando câmeras com sensores de profundidade e algoritmos de visão computacional, objetos, mãos e gestos são rastreados, possibilitando arrastar um objeto ou pegá-lo.

Personagens da história são projetados na mesa e os usuários podem interagir com eles usando suas mãos para pegá-los, soltá-los ou entregá-los a outros usuários. A Figura 11 exibe o uso do sistema.

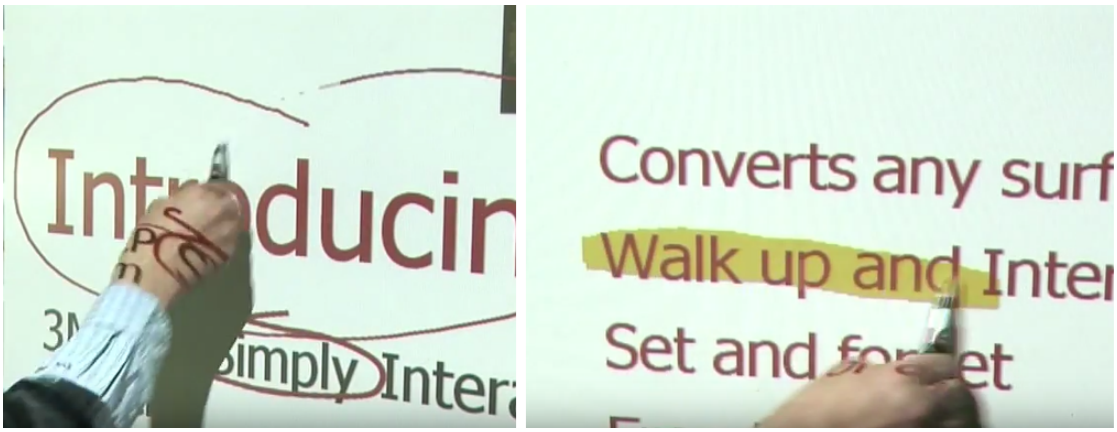
Figura 11 – *HIP-StoryTelling*: projeção interativa para narrativas.



Fonte: Melo et al. (2011, p. 4).

Outro projeto usado em ambientes educacionais é o proposto por Company (2010). O sistema é composto por um projetor, uma câmera infra vermelho e uma caneta que emite infra vermelho na superfície de projeção. A localização do clique é capturado pela câmera que identifica sua posição em uma matriz bidimensional composta pelos eixos X e Y. Com a detecção da interação é possível desenhar linhas sobre a projeção, grifar textos, fazer anotações, e controlar o *mouse* do computador. Essas funcionalidades são exibidas na Figura 12. Esse sistema pode ser utilizado para lecionar, fazer apresentações de *slides* ou também ser um *touchscreen* para controlar o computador.

Figura 12 – 3M *Simply interactive projection.*



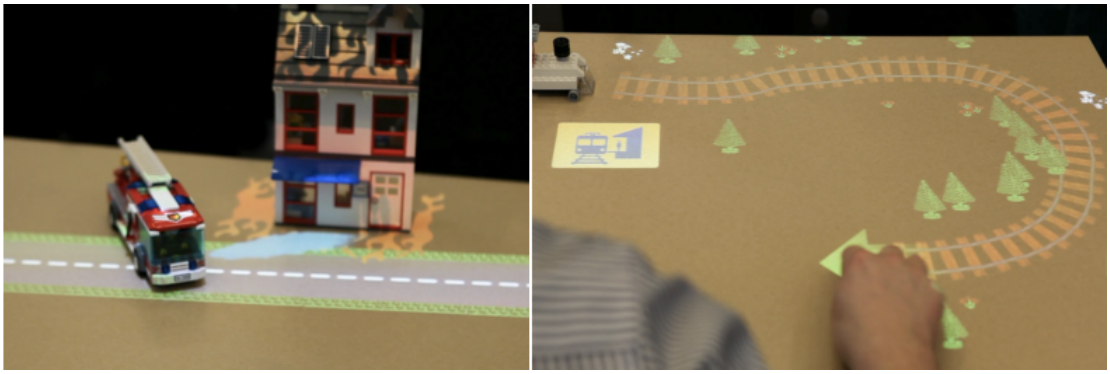
Fonte: Company (2010).

3.4.2 Entretenimento

O sistema de Ziola et al. (2011) utiliza algoritmos de computação visual e aprendizagem de máquina para reconhecer objetos em tempo real e interage com eles utilizando projeção. Por exemplo, colocando um dragão de Lego sobre a superfície de projeção, um fogo é projetado na direção em que ele aponta. E se outro objeto inflamável estiver perto, resulta no dragão incendiando este objeto. Objetos também podem gerar terreno projetado e ativar ambientes relacionados à sua identidade. Quando um trem é posto sobre a superfície de projeção, é possível desenhar trilhos com o gesto de arrastar sobre a superfície.

As entradas do sistemas são providas por uma câmera RGB-D, montada à 30 centímetros da superfície. A projeção é feita por um projetor de 170 lumens. A informação de profundidade é usada para isolar os objetos da superfície de projeção e também para detectar eventos de toque, arrastar, e sobreposição. Os objetos são reconhecidos utilizando informações de forma, cor e atributos de profundidade. O uso da aplicação é exibido na Figura 13.

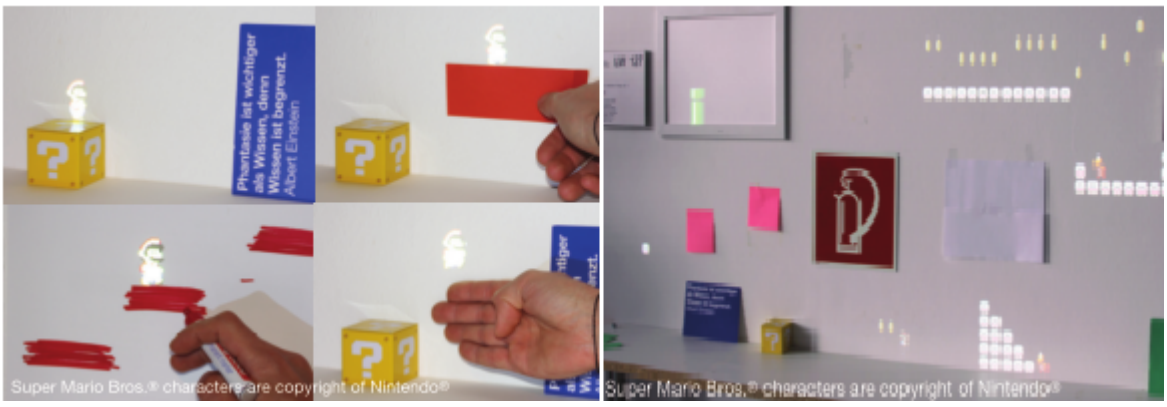
Figura 13 – Interação com objetos no sistema OASIS.



Fonte: Ziola et al. (2011).

No trabalho de Oswald et al. (2015), é proposto um Video Game interativo que permite jogar e interagir com o próprio ambiente em tempo real. A informação do ambiente é integrada ao conteúdo do jogo utilizando um *Microsoft Kinect* [2] e um projetor. O personagem do jogo também pode interagir com outros elementos digitais como oponentes, projéteis ou objetos bônus. Eles podem ser gerados detectando características dos objetos físicos anexados como cor e localização, como exibido na Figura 14.

Figura 14 – Interação de jogos com os objetos do ambiente.



Fonte: Oswald et al. (2015).

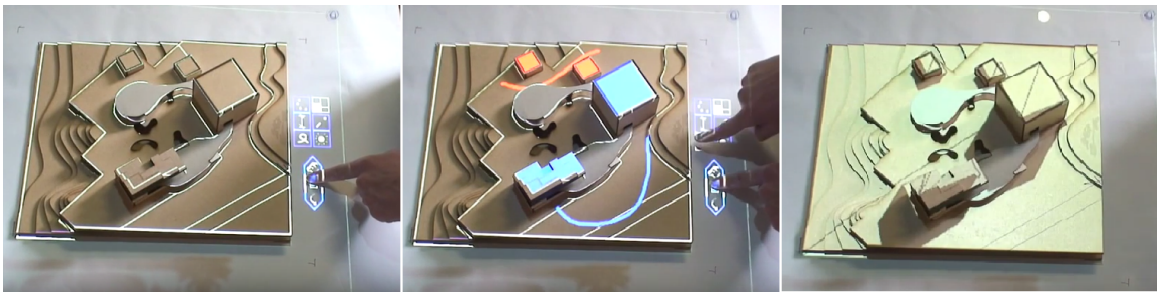
3.4.3 Outras aplicações

Outra aplicação interessante é proposta por Ghys (2013). É uma aplicação que possibilita interagir com maquetes de construções civis, como visto na Figura 15. Para

[2] http://www.xbox.com/pt-BR/xbox-one/accessories/kinect

esta interação é utilizado um *Microsoft Kinect* e um projetor, ambos suspensos sobre a maquete. O usuário pode interagir com a maquete colorindo superfícies, medindo distâncias, desenhando linhas e projetando a luz do sol pairando o dedo sobre a maquete.

Figura 15 – *Interactive Projection Mapping.*



Fonte: Ghys (2013).

3.5 Bibliotecas C++

Segundo Astrachan (2003), uma biblioteca pode ser uma coleção de classes, funções e variáveis escritas na própria linguagem para facilitar a implementação de softwares. Elas contem código e dados auxiliares, que provem instruções auxiliares a programas independentes. Uma das principais vantagens de ser armazenada em módulos independentes é que pode ser atualizada sem precisar alterar a aplicação em que é utilizada. Uma biblioteca é um uso de referência a endereços de dados ou instruções que são mantidos na forma relativa até o momento em que são necessários, ou seja, até o momento de execução da instrução que contém esta referência. As bibliotecas são classificadas pela maneira que são compartilhadas, ligadas e por quando são ligadas.

Durante o desenvolvimento do nosso trabalho, uma biblioteca de ligação dinâmica foi desenvolvida. Ela poderá ser utilizada para desenvolvimento de aplicações de projeção interativa, poupando o desenvolvedor de elaborar códigos longos e complexos.

Uma ligação dinâmica significa que os dados de uma biblioteca não são replicados em um executável em tempo de compilação, permanecendo em um arquivo separado no disco. O ligador (*linker*) carrega os dados relevantes na memória principal no momento em que eles são requisitados (tempo de execução) ou no momento em que o programa é carregado (tempo de carregamento) (ASTRACHAN, 2003).

3.6 Visão Computacional

Visão computacional é a transformação de dados armazenados ou de captura de uma câmera de vídeo em uma decisão ou uma nova representação (KAEHLER; BRADSKI,

2016). Todas transformações são feitas para um objetivo particular. O dado de entrada deve incluir alguma informação de contexto como "a câmera está montada em um carro" ou "a distância de um laser encontrado indica a distância do objeto". A decisão poderia ser "existe uma pessoa na frente do carro" ou "existe tumores nesta imagem de raio-x". Uma nova representação poderia ser transformar a imagem em uma nova com escalas de cinza ou remover movimentos da câmera em uma sequência de imagens (KAEHLER; BRADSKI, 2016).

As ações ou decisões que a computação visual tenta fazer com base em dados de câmera são realizadas no contexto de uma finalidade ou tarefa específica. Podemos querer remover ruídos ou danos de uma imagem para que nosso sistema de segurança emita um alerta se alguém tentar escalar uma cerca ou porque precisamos de um sistema de monitoramento que considere quantas pessoas entram em uma loja. O sistema de visão para carros autônomos empregará estratégias diferentes do que um sistema de visão para câmeras de segurança porque os dois sistemas têm contextos e objetivos diferentes. Como regra geral, quanto mais restrito for um contexto de visão computacional, mais podemos confiar nessas restrições para simplificar o problema e mais confiável será a solução final (KAEHLER; BRADSKI, 2016).

3.7 OpenCV

OpenCV (*Open Computer Vision*) é uma biblioteca de visão computacional de código aberto[3]. A biblioteca é escrita em C e C ++ e pode ser executada em Linux, Windows e Mac OS X. Há desenvolvimento ativo em interfaces para Python, Ruby, Matlab e outras linguagens (KAEHLER; BRADSKI, 2016).

O OpenCV foi desenvolvido para prover eficiência computacional com um forte foco voltado para aplicações de tempo real. A biblioteca possui suporte para programação em *multi-core* e otimização em interfaces Intel.

Um dos objetivos do OpenCV é prover uma infraestrutura para computação visual simples de usar, para ajudar programadores a construir aplicações visuais sofisticadas rapidamente. A biblioteca contém mais de 500 funções contidas em muitas áreas da visão computacional, incluindo tratamento de imagens médicas, segurança, interface de usuário, calibração de câmera, visão estéreo e robótica. Pelo fato de visão computacional e aprendizagem de máquina serem áreas bem próximas, a biblioteca OpenCV também contém uma biblioteca de aprendizagem de máquina de propósito geral. Essa sub-biblioteca é focada no reconhecimento de padrões estáticos e agrupamentos (*clustering).* Abaixo estão listados os objetivos iniciais do OpenCV:

- avançar a pesquisa de visão computacional com código aberto e otimizado para uma infra-estrutura básica, sem reinventar a roda;

[3] Disponível para download em https://sourceforge.net/projects/opencvlibrary/

- disseminar o conhecimento da visão computacional, fornecendo uma infraestrutura comum que os desenvolvedores possam construir e para que o código seja mais legível e transferível;

- avançar as aplicações comerciais de visão computacional por fazer um código com desempenho otimizado, portável e livre, com uma licença que não exige que as aplicações sejam abertas ou livres.

A licença de código aberto para o OpenCV foi estruturada de modo que se pode desenvolver um produto comercial usando todo ou parte das funcionalidades disponíveis na biblioteca. Não há obrigação de que o produto contenha código aberto ou que seja desenvolvido para domínio público, embora essas características sejam desejadas. Em partes, por causa desses termos liberais de licenciamento, há uma grande comunidade de usuários que inclui pessoas de grandes empresas como IBM, Microsoft, Intel, SONY, Siemens e Google, e centros de pesquisa como Stanford, MIT , CMU, Cambridge e INRIA. Existe um fórum do Yahoo Groups onde os usuários podem postar perguntas e discussões com quase 50.000 membros.

OpenCV é popular em todo o mundo, com grandes comunidades de usuários na China, Japão, Rússia, Europa e Israel. Desde a sua versão *alpha* em Janeiro de 1999, o OpenCV tem sido utilizado em muitas aplicações, produtos e esforços de investigação. Essas aplicações incluem costura de imagens em mapas satélite e *web*, alinhamento de digitalização de imagem, redução de ruído em imagens médicas, análise de objetos, sistemas de segurança e detecção de intrusão, sistemas automáticos de monitoramento e segurança, sistemas de inspeção de fabricação, calibração de câmera, aplicações militares e aéreos não tripulados, terrestres e veículos subaquáticos. Também tem sido usado no reconhecimento de som e música, onde técnicas de reconhecimento de visão são aplicadas a imagens de espectrograma de som. O OpenCV foi uma parte fundamental do sistema de visão do robô de Stanford, "Stanley", que ganhou a corrida de robô do deserto de *2M DARPA Grand Challenge* (KAEHLER; BRADSKI, 2016).

O OpenCV é estruturado em módulos, o que significa que o pacote inclui várias bibliotecas compartilhadas ou estáticas. Alguns dos módulos disponíveis são[4]:

- ***core*:** um módulo compacto que define basicamente estruturas de dados, incluindo o denso vetor Mat multidimensional e funções básicas usadas em outros módulos;

- ***imgproc*:** um modulo de processamento de imagens que inclui filtragem não linear de imagens, transformações geométricas de imagens (redimensionamento, distorção afim e perspectiva, remapeamento genérico baseado em tabelas), conversão de cores, histogramas e mais;

[4] Disponível em http://docs.opencv.org/master/

- ***video*:** um módulo de análise de vídeo que inclui estimativa de movimento, subtração de fundo e algoritmos de rastreamento de objetos;
- ***calib3d*:** algoritmos básicos de geometria de visão múltipla, calibração de câmera única e estéreo, estimativa de pose de objeto, algoritmos de correspondência estéreo e elementos de reconstrução 3D;
- ***features2d*:** detectores de características salientes, descritores e correspondentes de descritores;
- ***objdetect*:** detecção de objetos e instancias de uma classe predefinida. Por exemplo: olhos, rostos, pessoas, carros e mais;
- ***highgui*:** uma interface fácil de usar para recursos de UI (*user interfaces*) simples;
- ***videoio*:** uma interface fácil de usar para captura de video e *codecs* de vídeo;
- ***gpu*:** algoritmos acelerados por GPU de diferentes módulos do OpenCV;
- e alguns módulos auxiliares.

3.8 Metodologias Ágeis Para Desenvolvimento de Software

Os princípios da metodologia ágil auxiliam no desenvolvimento com base em poucos requisitos, prezando pela boa comunicação entra as partes interessadas e responder regularmente a mudanças, que é algo que acontece muito no decorrer do desenvolvimento do software (MOCHIO; ARAKI, 2011)

A iteração com poucos requisitos faz com que não haja necessidade de definir o escopo final do projeto para iniciar seu desenvolvimento. Existem vários métodos ágeis para suprir as necessidades da indústria, que vão desde métodos individuais ou comportamentais, como apresenta as práticas da programação extrema e estruturas de processos orientados para o gerenciamento de projeto, como o SCRUM. Muitos métodos possuem uma ótima aceitação na indústria de software como uma forma de garantir que os sistemas sejam criados dentro do prazo e do orçamento. (WOLFF, 2012). Os métodos ágeis Programação extrema e SCRUM são sucintamente descritos a seguir.

3.8.1 Programação extrema

A programação extrema (*Extreme Programming* ou XP), segundo Sommerville (2010), é talvez um dos mais conhecidos e utilizados métodos ágeis. Nesse método os requisitos são expressos como cenários (estórias de usuário) que são implementados diretamente como uma série de tarefas. Os programadores desenvolvem testes para cada tarefa antes de escreverem o código e quando o novo código é integrado ao sistema, todos os testes devem ser executados com sucesso.

O desenvolvimento incremental é sustentado por meio de pequenos *releases* do sistema. Os requisitos são baseados em cenários ou em simples estórias de clientes, usadas como base para decidir a funcionalidade que deve ser incluída em um incremento do sistema. Os cartões de estória são as principais entradas para o processo de planejamento em programação extrema. Quando são desenvolvidos, a equipe de desenvolvimento os divide em tarefas. Essas tarefas, então, são priorizadas para implementação, escolhendo aquelas que podem ser usadas imediatamente para executar o sistema.

Um problema geral com o desenvolvimento incremental é que ele tende a degradar a estrutura do software. A programação extrema aborda esse problema sugerindo que o software deve ser refatorado constantemente. Quando um membro da equipe percebe que o código pode ser melhorado, essas melhorias são feitas, mesmo quando não existe necessidade imediata destas.

O software deve ser sempre fácil de compreender e mudar à medida que novas estórias são implementadas. Na prática, muitas empresas que adotam a programação extrema não usam todas as práticas aconselhadas. Eles escolhem de acordo com o que é mais útil para sua organização.

3.8.2 Metodologia ágil para gestão de projeto SCRUM

O SCRUM não é uma técnica para desenvolver software, mas sim uma metodologia onde se pode utilizar diversos processos ou práticas para o desenvolvimento. Ele provê uma integração de estratégias para os eventos, artefatos e papéis garantindo a flexibilidade de mudança (WOLFF, 2012).

Dentro do SCRUM, os participantes podem assumir três papeis:

- ***Product Owner* (PO):** pessoa que representa o interesse do cliente e determina o objetivo de cada interação Sprint. A responsabilidade mais importante do PO é priorizar as funcionalidades e garantir que as mais importantes sejam implementadas em primeiro lugar, ou seja, quais funcionalidades devem ser feitas em cada iteração;

- ***Scrum Master*:** é a pessoa responsável por garantir que o SCRUM está entendido e sendo aplicado de forma correta. Ele tem a responsabilidade de garantir que a equipe não está distraída da tarefa em questão;

- **Equipe de Desenvolvimento:** pessoas que trabalham em conjunto para entregar software que satisfaça o cliente. Eles são estruturados, organizados e devem gerenciar seu próprio trabalho.

De acordo com (CARDOZO et al., 2010), o SCRUM utiliza um processo empírico com base em flexibilidade, adaptabilidade e produtividade. As atividades e artefatos do SCRUM podem ser descritas como:

- ***Backlog* do produto:** este artefato contém todas as especificações do sistema proposto e suas prioridades;
- ***Sprint*:** é um intervalo de tempo de 2 a 4 semanas, onde a equipe desenvolve determinadas especificações;
- **Planejamento do *Sprint*:** geralmente ocorre no primeiro dia do *Sprint*, onde a equipe se reúne para planejar quais atividades precisam ser feitas para concluir o *Sprint*;
- **Backlog do *Sprint*:** itens do *backlog* do produto que devem ser implementados no *Sprint*;
- **Reunião diária:** diariamente a equipe conversa em torno de 10 a 15 minutos e cada integrante da equipe responde três perguntas: o que fez no dia anterior; quais problemas teve para resolução de tarefas; e o que fará durante o dia;
- **Reunião de revisão do Sprint:** é onde a equipe entrega o incremento para o PO. O PO aceita ou não o incremento e define conjuntamente com a equipe o que será feito no próximo *Sprint*;
- **Reunião de retrospectiva do *Sprint*:** Acontece após a Reunião de revisão do *Sprint*, onde o foco é nos processos, técnicas e relações que estão dando certo e precisam continuar, ou que precisam ser revisadas e melhoradas.

3.9 Considerações do Capítulo

O vasto campo de aplicação da projeção interativa nos remete a uma crescente utilização durante os próximos anos. Esse fato nos motiva mais para o desenvolvimento do nosso trabalho, pois uma biblioteca de projeção interativa poderá impulsionar a sua disseminação em ambientes não só acadêmicos e corporativos, mas também para que desenvolvedores possam criar suas próprias aplicações de domínio específico.

Ao longo deste capítulo foi possível compreender algumas das tecnologias atualmente utilizadas na área da projeção interativa e conteúdos teóricos para o seu desenvolvimento. Também compreendemos processos de desenvolvimento e gestão ágil que facilitarão o processo de evolução da nossa biblioteca.

4 PLANEJAMENTO DO TRABALHO

Neste capítulo apresentamos o planejamento de desenvolvimento da biblioteca para aplicações de projeção interativa. Na seção 4.1 apresentamos o trabalho realizado durante a fase de definição, na seção 4.2 apresentamos o trabalho realizado durante a fase planejamento, incluindo uma lista de tarefas e um cronograma de desenvolvimento. Por fim, na seção 4.3 são apresentadas as considerações do capítulo.

4.1 Definição das funcionalidades

De modo geral, uma funcionalidade é uma ação que um sistema pode executar para uma determinada entrada, compondo um ciclo com início e fim. Para desenvolver estas funcionalidades são necessárias definições de requisitos do sistema, que são dados e informações sobre o sistema necessários para que o sistema seja desenvolvido. A seguir são exibidas as definições das funcionalidades do sistema.

4.1.1 Descoberta de requisitos

A descoberta de requisitos é o processo de reunir informações sobre o sistema requerido e os sistemas existentes e separar dessas informações os requisitos do sistema desenvolvido. Fontes de informação durante a fase de descoberta de requisitos incluem especificações de sistemas similares. Os requisitos são classificados como funcionais e não funcionais. Os requisitos funcionais de um sistema descrevem o que ele deve ou não fazer, já os requisitos não funcionais são requisitos que não estão diretamente relacionados com os serviços específicos oferecidos pelo sistema a seus usuários (SOMMERVILLE, 2010).

Neste processo de descoberta de requisitos, reunimos informações sobre a biblioteca desenvolvida em trabalhos já realizados nessa área, e separamos dessas informações os requisitos para que a biblioteca seja funcional e utilizável para desenvolvimento de aplicações de projeção interativa.

A definição dos requisitos levou em consideração o fato do sistema ser uma biblioteca de propósito geral para aplicações de projeção interativa, que deve atender ao vasto campo de aplicação fornecendo auxílio para desenvolvimento de aplicações específicas. De acordo com os trabalhos encontrados durante a pesquisa de conteúdos e trabalhos relacionados podemos definir os seguintes requisitos que uma biblioteca de projeção interativa deve conter:

- Requisitos funcionais:
  - identificar a orientação do plano da superfície de projeção;
  - corrigir deformações da imagem projetada de acordo com a orientação da superfície de projeção;
  - identificar interação de usuários a partir do toque na superfície de projeção;

  - aprender características de um objeto sobre a superfície de projeção;
  - identificar objetos sobre a superfície de projeção;
  - identificar interação com objetos sobre a superfície de projeção;
  - prover reações a partir de determinadas interações;
  - ser capaz de exibir conteúdos definidos pelo usuário.

- Requisitos não funcionais:

  - possuir implementação legível e simples;
  - ser capaz de responder à estímulos de interações com rapidez;
  - fornecer resultados com uma boa acurácia;
  - o uso da biblioteca deverá ser feito em sistemas Linux;

4.1.2 Backlog do produto.

O *backlog* do produto foi desenvolvido utilizando estórias de usuário, que é um artefato de desenvolvimento ágil responsável por descrever uma tarefa e sua prioridade. As estórias de usuários estão listadas na Tabela 2, ordenadas por prioridade, da mais alta à mais baixa e com suas respectivas dependências.

4.2 Fase de planejamento

Na fase de planejamento, averiguamos o que deveria ser feito para alcançar nosso objetivo, definindo tarefas e cronogramas. O planejamento do projeto é um processo iterativo que começa quando um plano inicial de projeto é criado, na fase de iniciação. E conforme mais informações sobre o sistema estiverem disponíveis, o plano poderá ser alterado para refletir as mudanças de requisitos ou cronograma.

Durante o desenvolvimento da biblioteca utilizaremos métodos de desenvolvimento ágil, combinando os métodos XP e SCRUM, vistos na seção 3.8, com duração de um ciclo *Sprint* de uma semana. Os métodos ágeis de desenvolvimento de software são abordagens iterativas nas quais o software é desenvolvido e entregue em incrementos. A funcionalidade incluída em cada incremento é planejada com antecedência, mas passível de mudanças. A decisão sobre incluir uma funcionalidade em incremento ou não depende do progresso e das prioridades da biblioteca.

Avaliamos o projeto quanto ao tempo de desenvolvimento e sua complexidade de forma empírica e então inferimos pontos de checagem (*milestones*) em que a biblioteca poderá ser utilizada com as funcionalidades até então desenvolvidas. Os detalhes do planejamento são exibidos a seguir.

Tabela 2 – Backlog do produto.

ID	Descrição	Prioridade	Dependência
T1	Como usuário desenvolvedor quero poder incluir a biblioteca em meu código para facilitar o desenvolvimento da minha aplicação	Alta	–
T2	Como usuário desenvolvedor quero ter recursos para saber se a projeção está alinhada e a imagem possui forma retangular	Alta	T1
T3	Como usuário desenvolvedor quero que a projeção se alinhe automaticamente	Baixa	T1, T2
T4	Como usuário desenvolvedor quero ter recursos para saber se ocorreu um toque na superfície de projeção	Alta	T1, T3
T5	Como usuário desenvolvedor quero ter recursos para saber se há um objeto sobre a projeção	Média	T1, T3
T6	Como usuário desenvolvedor quero ter recursos para identificar um objeto sobre a projeção	Média	T1, T5
T7	Como usuário desenvolvedor quero ter recursos para identificar uma interação de objeto	Média	T1, T6
T8	Como usuário desenvolvedor quero poder exibir conteúdos de acordo com uma determinada interação	Alta	T1, T4, T7

Fonte: resultado da definição de tarefas.

4.2.1 Cronograma de desenvolvimento

O cronograma de desenvolvimento foi estimado considerando que o tempo decorrido na fase de codificação não deve ultrapassar 120 dias. Avaliamos o tempo estimado de cada tarefa de forma pessimista, disponibilizando mais dias para cada tarefa considerando a possibilidade de contratempos. A Tabela 3 exibe a organização dos *Sprints* em relação a inclusão de cada tarefa, assim como sua duração. Os asteriscos indicam pontos de checagem que definem versões usáveis da biblioteca.

4.3 Considerações do capítulo

No decorrer deste capítulo apresentamos as principais funcionalidades que desejávamos desenvolver e o cronograma para este desenvolvimento. Percebemos que o planejamento é uma das atividades mais importantes de qualquer desenvolvimento, pois possibilita visualizar as tarefas que precisam ser executadas para atingir o objetivo.

Tabela 3 – Cronograma de desenvolvimento das tarefas.

ID	**Incluído no *Sprint***	**Início**	**Duração (dias)**
T1	1	17/mar	7
T2	2	24/mar	14
T3	4	07/abr	7
T4*	5	14/abr	21
T5	5	14/abr	35
T6	9	18/mai	14
T7	11	02/jun	14
T8*	13	16/jun	21

* *Milestones*

Fonte: resultado da definição de tarefas.

Também vimos que o cronograma mesmo não sendo um objeto imutável, apresenta grande dificuldade para ser estimado, pois, não possuímos informações detalhadas sobre quanto tempo cada atividade levará para ser realizada, e esse fato aumenta a possibilidade de haver alteração durante o desenvolvimento.

5 DESENVOLVIMENTO DO TRABALHO

Neste capítulo, descrevemos os métodos utilizados para o desenvolvimento da biblioteca e suas funcionalidades, bem como o modo que foi planejado seu funcionamento. Na seção 5.1, descrevemos os materiais utilizados durante a fase de desenvolvimento e testes da biblioteca. Na seção 5.2, mostramos uma visão geral da biblioteca desenvolvida. Da seção 5.3 até a seção 5.8 exibimos os métodos utilizados e desenvolvidos. Na seção 5.9, explicamos o protótipo desenvolvido para ilustrar o funcionamento da biblioteca. Na seção 5.10 exibimos os resultados do desenvolvimento do nosso trabalho. Por fim, na seção 5.11, apresentamos as considerações do capítulo.

5.1 Materiais e métodos

Para desenvolvimento e execução da biblioteca foi utilizada a linguagem C++, compilada em linha de comando com CMake e executado com o comando *./sample*. O CMake é um conjunto de ferramentas de plataforma aberta e de código aberto projetadas para criar, testar e empacotar software[1].

O computador utilizado possui um processador Intel core I5 dual com frequência 1.7ghz, 10GB de memória RAM DDR3, e sistema operacional Ubuntu 18.04 com arquitetura de 64 bits. Para captura de imagem foi utilizada a Web Cam Logitech C920 Pro HD 1080P. A projeção da imagem foi feita com o projetor BENQ MS502 DLP ANSI 2700 SVGA (800 X 600).

O sistema foi montado sobre uma mesa e direcionado para uma parede de tonalidade clara. A câmera foi posicionada próxima ao projetor, minimizando a criação de sombras na projeção. O projetor foi colocado a uma distância aproximada de 1,3 metros da parede, gerando uma projeção de 30 polegadas. A montagem do sistema é exibida na Figura 16.

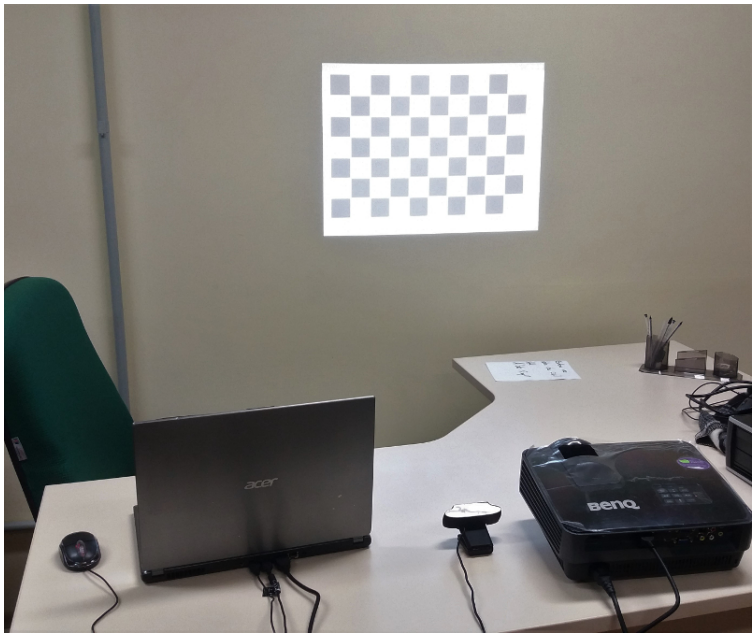
Para geração da documentação foi utilizada a ferramenta Doxygen. O Doxygen é uma ferramenta para gerar documentação a partir de códigos fonte C++ anotados, mas também suporta outras linguagens de programação populares, como C, Objective-C, C#, PHP, Java, Python entre outras[2].

5.2 Visão Geral da Biblioteca de Projeção Interativa

A biblioteca foi implementada com intuito de estimular o desenvolvimento de aplicações de projeção interativa, oferecendo funcionalidades que facilitem no desenvolvimento de tais aplicações. Para isso foram implementados mais de dez métodos que auxiliam na correção de perspectiva da imagem capturada pela câmera, identificação da região da mão, identificação da ponta do dedo, detecções de interação e atualização da imagem

[1] CMake. Disponível em: https://cmake.org/.
[2] Doxygen: Main Page. http://www.stack.nl/ dimitri/doxygen/.

Figura 16 – Montagem do sistema de projeção interativa.



Fonte: gerada pelo autor.

projetada. Na Figura 17 são exibidos os passos do processo de detecção de interação, que pode ser feito a partir da região da mão e também pela ponta do dedo.

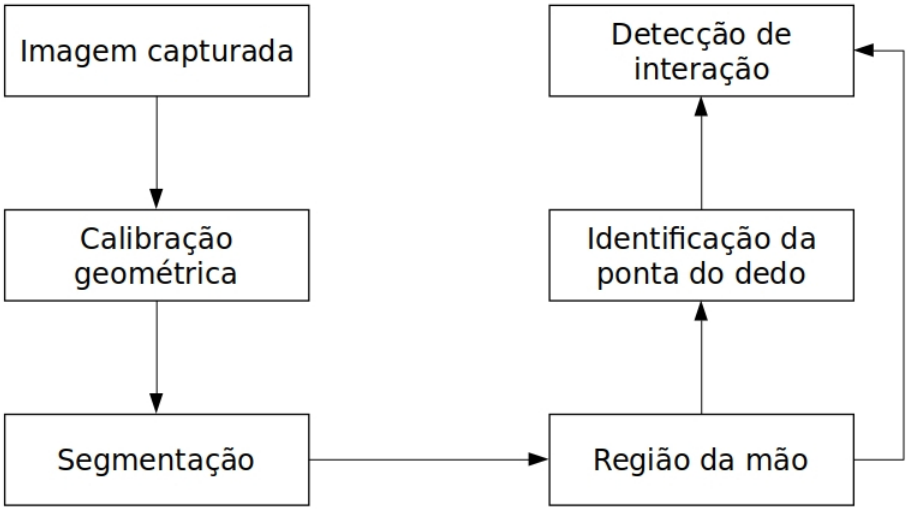
Quando iniciamos a fase de desenvolvimento da biblioteca, nossa intenção era seguir o modelo do trabalho de He, Cheng e Tao (2014), exibido na seção 3.3. Entretanto, devido algumas dificuldades em replicar os métodos utilizados, decidimos fazer algumas mudanças na maneira que a interação é identificada. Explicamos as razões na seção 5.4 e seção 5.7.

No modelo de He, Cheng e Tao (2014) é feita uma comparação entre a imagem capturada e a imagem prevista para identificar a região da mão. A imagem prevista é gerada pela aplicação da calibração geométrica e fotométrica. A identificação da ponta do dedo é feita medindo as distâncias entre o centróide da mão e os pontos externos do contorno. Por fim, a interação com a projeção é identificada usando os algoritmos FAST e BRISK.

O nosso modelo difere do modelo de He, Cheng e Tao (2014) em alguns aspectos. Nós não utilizamos uma imagem prevista para identificar a região da mão, fazendo com que a maneira que essa identificação é feita seja diferente. O toque na superfície de projeção não é implementado em nosso trabalho, sendo substituído pela detecção da colisão entre a região da mão e os objetos na projeção. A calibração geométrica e a identificação da ponta do dedo são semelhantes em ambos trabalhos.

Os métodos disponíveis na biblioteca são listados abaixo:

Figura 17 – Visão geral da biblioteca de projeção interativa.



Fonte: gerada pelo autor.

- ***InteractiveProjection()***: construtor padrão;
- ***~ InteractiveProjection()***: destrutor padrão;
- ***bool Start(int cameraId)***: inicia os parâmetros da projeção interativa. Recebe o identificador da câmera, configura a resolução, formato e inicia a captura. Carrega a imagem xadrez do disco rígido, detecta seus cantos e a exibe. O método tem um retorno boleano, sendo verdadeiro, se não existir nenhuma falha durante a inicialização, ou falso, se por exemplo, a câmera estiver indisponível ou a imagem com o padrão xadrez para calibração não for encontrada;
- ***void Apply()***: executa a atualização da projeção interativa. Recebe os dados da captura, identifica a região da mão aplicando filtros e selecionando o maior contorno e identifica a ponta do dedo;
- ***void ShowCapturedImage()***: exibe a imagem capturada;
- ***void ShowDetectedInteraction()***: exibe a região da mão com o centróide e a ponta do dedo destacados;
- ***void UpdateProjectedImage(Mat image)***: atualiza a imagem projetada dada uma imagem de entrada;
- ***void ShowProjectedImage()***: exibe a imagem projetada;

- ***bool FingertipInteraction(Rect rectangle)***: verifica se a ponta do dedo está contida em um dado retângulo;
- ***bool RegionInteraction(Rect rectangle)***: verifica se algum ponto do contorno da mão está contido em um dado retângulo;
- ***bool FingertipInteraction(Point center, double radius)***: verifica se a ponta do dedo está contida em um dado círculo;
- ***bool RegionInteraction(Point center, double radius)***: verifica se algum ponto do contorno da mão está contido em um dado círculo;
- ***void KeyListener()***: recebe as interações de teclado e executa suas ações, como iniciar a calibração geométrica, distorcer a imagem capturada, iniciar ou pausar a detecção de interação, sair da aplicação e outros.

A biblioteca está disponível sob a licença de software MIT, ou seja, qualquer pessoa que obtém uma cópia da biblioteca e seus arquivos de documentação associados pode lidar com eles sem restrição, incluindo sem limitar os direitos a usar, copiar, modificar, mesclar, publicar, distribuir, vender cópias do software. As condições impostas para tanto são apenas manter o aviso de *copyright* e uma cópia da licença em todas as cópias da biblioteca.

O código fonte da biblioteca, o exemplo desenvolvido e sua documentação estão disponíveis no repositório público *Github* através do link <https://github.com/alvesmaicon/InteractiveProjectionLib> . O *GitHub* é uma plataforma de hospedagem de código-fonte com controle de versão usando o *Git*. Ele permite que programadores, utilitários ou qualquer usuário cadastrado na plataforma contribuam em projetos privados e/ou *Open Source* de qualquer lugar do mundo. *GitHub* é amplamente utilizado por programadores para divulgação de seus trabalhos ou para que outros programadores contribuam com o projeto, além de promover fácil comunicação através de recursos que relatam problemas ou mesclam repositórios remotos (*issues, pull request*).

5.3 Estimativa Homográfica

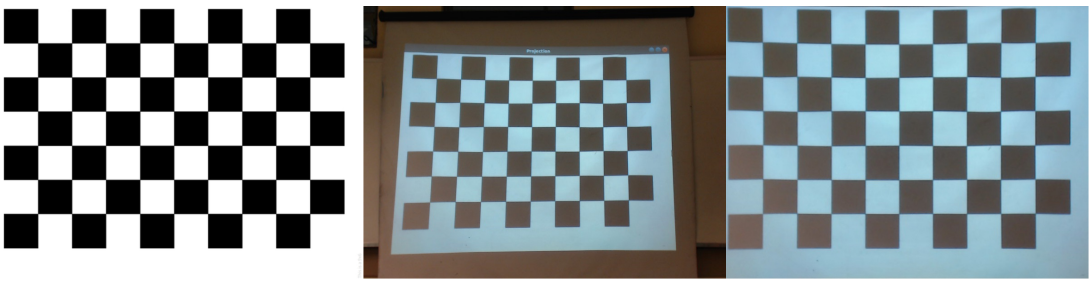
O objetivo da estimativa homográfica é identificar pontos de correspondência na imagem capturada e na imagem projetada. A partir dessa correspondência de *pixels* é possível saber com precisão qual a localização da interação na projeção a partir da localização na imagem capturada.

Durante o processo de pesquisa aos trabalhos relacionados, identificamos meios para obter a homografia entre a imagem projetada e a imagem capturada. Os métodos mais usados são projetar padrões na superfície de projeção e identifica-los na imagem capturada. No trabalho de Xiaopeng et al. (2014) visualizamos um exemplo de estimativa

homográfica, descrito na subseção 3.3.1. Um tabuleiro de Xadrez é projetado e capturado. Na captura da projeção são identificados os cantos do tabuleiro e suas localizações. Com isso é possível relacionar os *pixels* das duas imagens.

No desenvolvimento do nosso trabalho utilizamos um método semelhante ao de Xiaopeng et al. (2014). Primeiro é carregado do disco rígido a imagem do tabuleiro de Xadrez com dimensões 6x9 casas e encontrados seus cantos. Utilizando um método para interagir com o usuário recuperamos teclas digitadas enquanto a aplicação está em execução e, caso o usuário digite "c", identificamos os cantos do tabuleiro capturado pela câmera. Logo após encontrar os cantos, o tabuleiro é projetado na superfície de projeção. A interação por teclado é feita pelo método *Keylistener* implementado em nossa biblioteca. Se a câmera está visualizando por completo a projeção e os cantos do tabuleiro de Xadrez são identificados, a estimativa homográfica é concluída. Caso contrário, uma mensagem de erro é exibida ao usuário. A implementação dessa funcionalidade é exibida na linha 234 da seção A.1.

Figura 18 – Estimativa homográfica e correção de perspectiva.



Fonte: gerada pelo autor.

Na Figura 18 o processo de correção de perspectiva é exibido. Da esquerda para direita temos: imagem projetada, imagem capturada, e imagem capturada com perspectiva corrigida.

Durante a execução da aplicação o usuário tem a opção de corrigir a perspectiva da imagem capturada para enquadrar a área de projeção em que o tabuleiro foi identificado. Enquanto a imagem capturada está com a perspectiva corrigida, ela possui pontos exatos de correspondência na imagem projetada, sendo possível estimar a posição exata da interação do usuário com a projeção.

Além de identificar a relação de *pixels* entre as duas imagens, tínhamos também o interesse em corrigir a projeção e deixá-la com formato retangular na superfície de projeção caso estivesse distorcida. Porém, com apenas informações 2D de uma câmera, esse método seria dependente da posição da câmera, que deveria estar exatamente ortogonal ao plano de projeção para que a imagem projetada fosse corrigida de forma eficaz.

Devido ao tempo gasto para tentar corrigir a orientação do plano de projeção,

decidimos deixar essa funcionalidade para trabalhos futuros, em que podemos pesquisar outras formas para corrigir o plano de projeção e garantir que a projeção tenha um formato retangular. Por enquanto assumimos que o usuário tenha posto o projetor em uma posição ortogonal à superfície de projeção ou tenha corrigido a projeção utilizando as opções do projetor.

5.4 Correção Fotométrica

A correção fotométrica tem por objetivo gerar uma imagem prevista da imagem capturada pela câmera a partir da imagem projetada. No trabalho de He, Cheng e Tao (2014), a imagem prevista é utilizada para identificar a região da mão durante a interação do usuário subtraindo as duas imagens, dado que na imagem capturada existe uma grande diferença da imagem prevista caso exista, por exemplo, uma mão sobre a superfície de projeção.

Durante a implementação do nosso trabalho utilizamos este mesmo método, a imagem prevista foi gerada a partir da imagem projetada utilizando o modelo fotométrico visto no trabalho de He, Cheng e Tao (2014), descrito na subseção 3.3.1.

A imagem prevista gerada a partir do modelo fotométrico não teve total semelhança à imagem capturada, apresentando grandes diferenças de cor nos *pixels* próximos às extremidades da imagem. A imagem capturada apresenta um decaimento de brilho do centro aos cantos, e a imagem prevista apresenta um brilho homogêneo em toda sua superfície. O resultado da estimativa fotométrica feito em duas imagens é exibido na Figura 18, sendo as imagens esquerdas as imagens projetadas, as imagens centrais são as imagens previstas, e as imagens direitas as imagem capturadas pela câmera.

E por consequência das diferença na imagem projetada e imagem prevista, a identificação da região da mão, vista no trabalho de He, Cheng e Tao (2014) e descrito na subseção 3.3.1, também não obteve um resultado satisfatório, não sendo possível sua utilização. As tentativas de corrigir este problema estavam atrasando o andamento do projeto, então decidimos tentar outras maneiras de identificar a região da mão.

5.5 Identificação da Região da Mão

A identificação da região da mão tem por objetivo separar a região da mão da imagem capturada pela câmera para identificar onde está ocorrendo a interação do usuário com a projeção. Neste ponto do nosso trabalho, tentamos desenvolver o método visto no trabalho de He, Cheng e Tao (2014) sem alcançar os resultados esperados, então decidimos utilizar os métodos implementados na biblioteca OpenCV para efetuar a segmentação do plano de fundo. O método utilizado no nosso trabalho foi o *BackgroundSubtracotrMOG2*, que é baseado em *Gaussian Mixture Model* (GMM).

Figura 19 – Estimativa fotométrica.

Fonte: gerada pelo autor.

O GMM é uma função de densidade de probabilidade paramétrica representada como uma soma ponderada de densidades de componentes Gaussianas. Os parâmetros GMM são estimados a partir de dados de treinamento usando o algoritmo iterativo *Expectation-Maximization* (EM) ou *Maximum A Posteriori* (MAP) de um modelo anterior treinado (ALI; GOYAL; SINGHAI, 2017).

A implementação da identificação da região da mão é executada após corrigir a perspectiva da imagem capturada. O usuário tem a opção de iniciar a captura de interação a partir de um comando de teclado. Quando a tecla "d" é digitada, a funcionalidade de identificação de interação é ativada. A imagem capturada é passada para o método *BackgroundSubtracotrMOG2*, que por sua vez identifica os *pixels* que são diferentes em um histórico de imagens capturadas (OPENCV, 2018b).

O método possui um parâmetro que aciona ou não o treinamento recursivo. Esse parâmetro é acessado por meio da interação do usuário com o teclado. Caso o usuário tecle "*space*", o treinamento recursivo é ativado ou desativado. Se o treinamento recursivo estiver ativo, um certo número de capturas irá influenciar na identificação da interação, fazendo com que uma interação que se comporte de forma estática deixe de ser identificada ao passar do tempo. Como por exemplo, se a mão pairar sobre a projeção em um ponto fixo sem alteração de posição, ela deixará de ser identificada em um dado número de iterações, sendo considerada parte do plano de fundo. Por outro lado, se o treinamento recursivo estiver desativado, a primeira imagem capturada pelo método será utilizada como plano de fundo, e toda interação será capturada, seja ela estática ou dinâmica.

Quando o treinamento recursivo está desativado, é necessário reativá-lo em cada mudança total da imagem projetada. Dado que se a imagem projetada for alterada ela

será identificada como uma alteração nos *pixels* do plano de fundo.

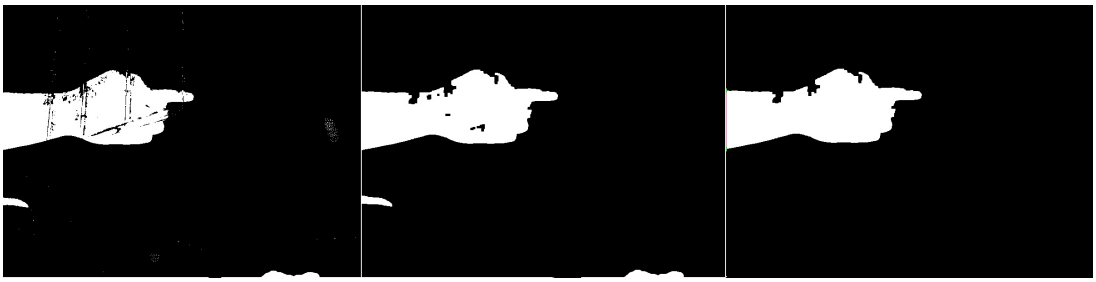
O método *BackgroundSubtracotrMOG2* identifica a região da mão colorindo os *pixels* em que há uma diferença de cor entre um histórico de capturas na cor branca, e os demais *pixels* na cor preta. Esse processo gera um ruído devido à variação de iluminação e cor na imagem capturada, sendo necessário uso de filtros para que esse ruído seja minimizado.

Os filtros utilizados são variações do método *morphologyEx*. Primeiro é aplicado o filtro de abertura de *pixels*, que é uma aplicação da operação erosão seguida da operação dilatação. Em seguida é aplicado o filtro de fechamento, que é semelhante ao de abertura, porém com a ordem das operações invertidas. Desse modo os *pixels* pequenos de ruído são minimizados, e as aberturas na áreas de *pixel* branco são fechadas.

O próximo passo é a identificação da região da mão. Os pontos dos contornos da imagem filtrada são obtidos com o uso do método *findContours*. Na maioria das vezes é identificado mais de um contorno devido à aplicação de filtros não eliminar regiões grandes de ruído, então buscamos o contorno de maior área. Essa seleção restringe a identificação de interação a uma única região, ou seja, apenas a região de maior área é identificada.

A aplicação dos filtros e seleção do maior contorno são exibidos na Figura 20. A imagem da esquerda é a imagem obtida pelo método *BackgroundSubtracotrMOG2*, a imagem central é a mesma imagem após a minimização do ruído, e a imagem da direita é a seleção da região com a maior área de contorno.

Figura 20 – Aplicação de filtros para minimização de ruído e seleção de maior contorno.



Fonte: gerada pelo autor.

A Figura 21 exibe o processo de identificação da região da mão. A imagem da esquerda é a imagem projetada, a imagem central é a imagem capturada e a imagem direita exibe a região da mão após a aplicação dos filtros e seleção do maior contorno.

5.6 Identificação da Ponta do Dedo

A identificação da ponta do dedo é feita a partir do maior contorno selecionado durante a identificação da região da mão. Primeiro encontramos o centro de gravidade do

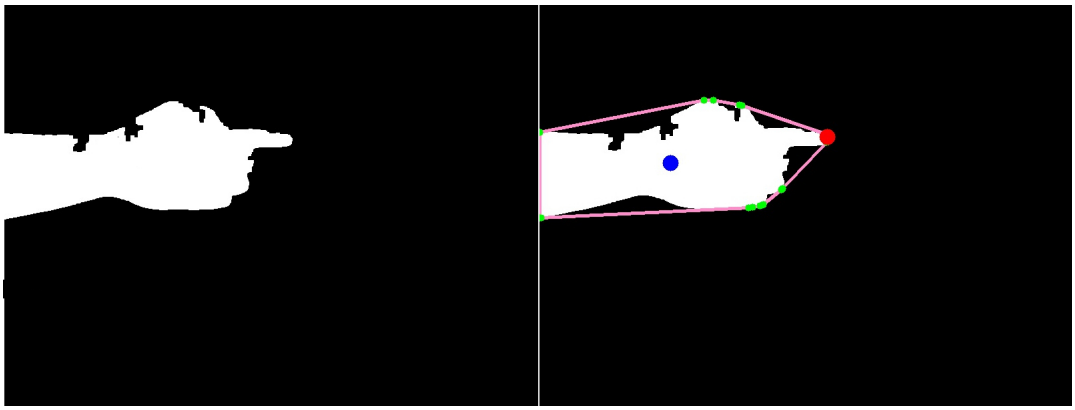
Figura 21 – Identficação da região da mão.

Fonte: gerada pelo autor.

contorno utilizando o método *moments*. Então encontramos os pontos de contorno que formam uma figura plana com o método *convexHull*, que percorre os pontos de contorno da região da mão e seleciona os pontos sem defeitos de convexidade (pontos externos). De um modo geral, as curvas convexas são as curvas que são salientes ou pelo menos planas. E, se um ponto estiver abaixo de uma linha traçada por dois pontos externos, é chamado de defeito de convexidade (OPENCV, 2018a).

Após encontrar os pontos externos da imagem, percorremos a lista de pontos buscando o ponto com a maior distância até o centróide do contorno, como visto em He, Cheng e Tao (2014). Nesse método é necessário que um dos dedos esteja esticado enquanto os outros ficam dobrados, tomando uma forma indicativa. Na Figura 22 é exibida a identificação da ponta do dedo. Na imagem esquerda é exibido o contorno da região da mão e na imagem direita, o ponto azul é o centroide do contorno da mão, os pontos verdes são os pontos externos, e o ponto vermelho é a ponta do dedo.

Figura 22 – Identificação da ponta do dedo.



Fonte: gerada pelo autor.

Na Figura 23 é exibido o resultado da identificação da ponta do dedo com diferentes

condições de iluminação. Nas imagens esquerdas são exibidas as imagens projetadas, nas centrais as imagens capturadas e nas imagens direitas são exibidas as pontas dos dedos marcadas com o ponto vermelho.

A identificação da ponta do dedo se mostrou eficiente em projeções claras com condições de iluminação ambiente claras. Porém, em projeções escuras e baixa iluminação ambiente, a região da mão não é identificada com precisão. Fazendo com que a região da mão não seja reconhecida em sua totalidade e influenciando na localização da ponta do dedo detectada.

Figura 23 – Resultado das identificações de ponta do dedo em diferentes condições de iluminação.

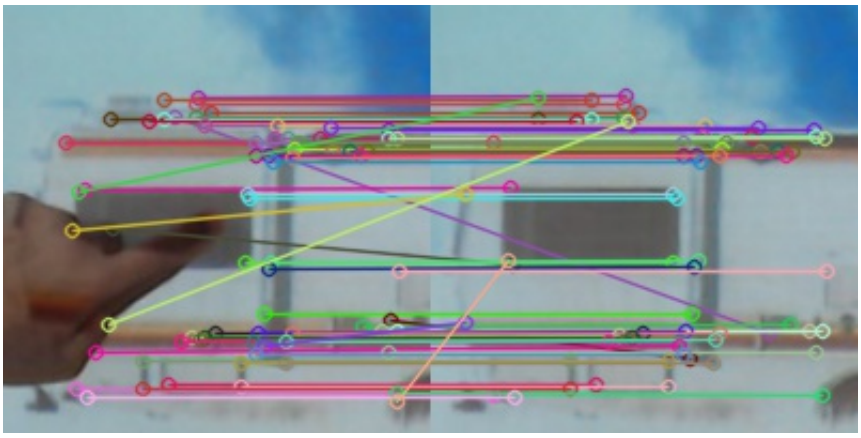
Fonte: gerada pelo autor.

5.7 Detecção do Toque na Superfície de Projeção

Durante a implementação da detecção de toque, testamos dois métodos distintos para sua execução. Primeiro tentamos utilizar o método descrito em (HE; CHENG; TAO, 2014). Que é baseado nos algoritmos FAST e BRISK para detecção do toque na projeção, visto na subseção 3.3.4. Este modelo busca pontos de interesse na imagem capturada e imagem prevista em uma região de 200x200 *pixels* utilizando o algoritmo FAST.

Logo após a aplicação do algoritmo FAST é feita a correspondência dos pontos de interesse encontrados em ambas imagens com o algoritmo BRISK. A detecção do toque é feita a partir da distância média entre os pontos de interesse correspondentes. Quando o dedo está pairando a projeção, a imagem projetada fica distorcida, fazendo com que a distância média seja alta. Por outro lado, se o dedo estiver tocando a superfície de projeção, a imagem projetada sobre o dedo se aproxima da forma da imagem projetada, fazendo com que a distância média dos pontos de interesse seja baixa. Caso a distância média for menor que um limiar, o toque é identificado. Esse processo é exibido nas Figura 24 e Figura 25, as imagens exibem os cortes 200x200 *pixels* na imagem capturada (esquerda) e imagem capturada antes da interação (direita).

Figura 24 – Reconhecimento de toque baseado em FAST e BRISK em uma imagem clara.



Fonte: gerada pelo autor.

A nossa implementação não obteve êxito na detecção do toque. Não são identificados os pontos de interesse sobre o dedo, não sendo possível estimar a distância entre os pontos de interesse que indicariam o toque ou não. Acreditamos que esse resultado obtido, tanto em imagens projetadas claras como escuras, é devido às configurações de hardware utilizados, diferentes dos utilizados no trabalho de He, Cheng e Tao (2014). A câmera teve que ser posicionada a uma certa distância da superfície de projeção para que pudesse lidar com a alta iluminação gerada pelo projetor, não captando com detalhes a superfície do dedo.

Figura 25 – Reconhecimento de toque baseado em FAST e BRISK em uma imagem escura.

Fonte: gerada pelo autor.

Quanto mais próxima ficava a câmera da superfície de projeção, mais clara ficava a captura, chegando a ficar completamente branca em algumas distâncias utilizadas. Tentamos configurar o projetor para que emitisse menos luz, porém a qualidade da imagem projetada era afetada.

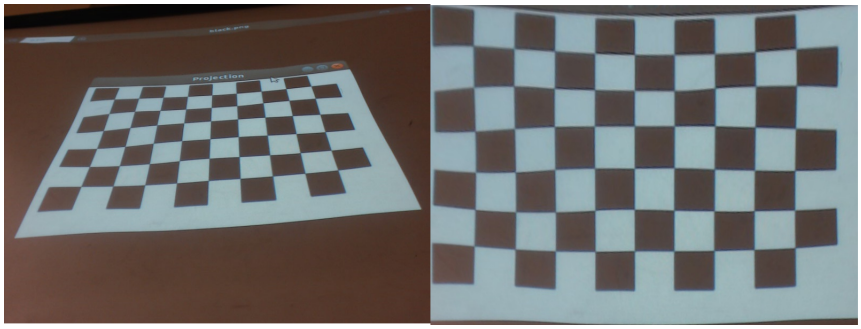
Após as tentativas sem resultados na detecção do toque pelo método citado a cima, optamos por mudar o modo que o toque era identificado. No trabalho de Song et al. (2007) é proposto um modo de capturar a interação do usuário a partir da sombra gerada pela mão na superfície de projeção. A ponta do dedo e a ponta da sombra são identificados e a distância entre elas definirá se ocorre ou não um toque.

Quando o dedo está pairando sobre a superfície de projeção dois dedos são identificados (o dedo e sua sombra). Quando o dedo toca a superfície de projeção, a ponta do dedo e a ponta da sombra se unem formando apenas um dedo largo. Quando esse único dedo é identificado, é por que ocorreu o toque.

Para que a sombra e a mão sejam identificados é necessário que a câmera seja apontada para superfície de projeção de modo que gere um certo ângulo. Na nossa montagem do projetor e câmera, o projetor está fixado ao teto e apontado para parede, e a câmera está apontando para a parede fixada próxima ao chão, como visto na Figura 26. A imagem esquerda é a imagem capturada pela câmera antes da calibração geométrica, e a imagem da direita é a captura após a correção de perspectiva.

Depois de tentar reproduzir o método como visto no trabalho de Song et al. (2007) sem obter sucesso devido a falta de informação de como o método foi desenvolvido, decidimos desenvolver o nosso próprio método baseado nos mesmos conceitos. Primeiro a ponta do dedo na região da mão e a ponta do dedo na sombra são identificados. Depois é traçada uma linha que indica a distância entre ambas pontas do dedo. Caso essa distância

Figura 26 – Calibração para reconhecimento de toque baseado em sombra.



Fonte: gerada pelo autor.

seja menor que um limiar especificado, o toque na superfície de projeção é identificado. A utilização desse algoritmo gera várias restrições sobre o modo como o sistema deve ser montado e utilizado. Se o ângulo entre o projetor e a câmera não forem suficientes para gerar uma sombra bem definida, o toque não será identificado. O modo que a mão paira sobre a superfície de projeção influencia na forma da sombra gerada. A projeção de uma imagem escura gera pouca sombra dado que a cor preta é a ausência de luz naquele ponto, dificultando a identificação da sombra e posteriormente a detecção do toque.

Essa última restrição é exibida nas Figura 27 e Figura 28, que mostra as imagens capturadas e região da mão quando a mão está pairando a superfície de projeção no lado esquerdo, e do lado direito, quando ocorre um toque na superfície de projeção. A Figura 27 exibe o resultado positivo em uma imagem clara, enquanto a Figura 28 exibe um resultado negativo quando a imagem projetada é escura.

Figura 27 – Detecção de toque baseado em sombra em uma imagem clara.

Fonte: gerada pelo autor.

Devido a restrição dos pontos escuros da imagem projetada não definir de forma exata a sombra da mão sobre a projeção, os resultados não foram precisos em sua totalidade. Pois quando a sombra não está bem definida, a ponta do dedo na sombra não é identificada corretamente, podendo gerar um toque em uma posição diferente da posição onde ocorreu o toque ou simplesmente não ser reconhecido.

Figura 28 – Detecção de toque baseado em sombra em uma imagem escura.

Fonte: gerada pelo autor.

Decidimos não utilizar a detecção de toque por sombra, embora tenha sido positivo em alguns casos. A dificuldade de reproduzir a configuração necessária para um bom funcionamento e a restrição de como interagir com a projeção, tornaram a detecção do toque difícil de ser identificado, fazendo com que a complexidade do uso da biblioteca para desenvolver uma aplicação aumentasse.

5.8 Detecção de Interação

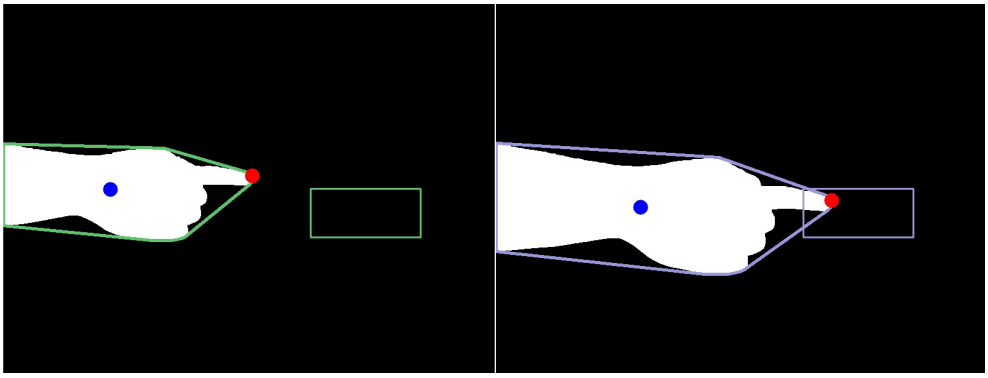
Após tentativas em reconhecer o toque do dedo na superfície de projeção sem obter resultados aceitáveis, decidimos utilizar a interação a partir da colisão da região da mão com um dado objeto e suas coordenadas definidas pelo usuário. O método de colisão é tratado de forma 2D, ou seja, não leva em consideração a distância da mão à superfície de projeção.

Desenvolvemos quatro métodos para identificar a interação com a projeção, dois métodos para objetos retangulares e dois para objetos circulares. A identificação da interação é feita utilizando uma detecção de intersecção entre os objetos. Essa verificação diz se a região da mão possui algum ponto de contorno dentro da área do objeto. Se o resultado for positivo, a interação com o objeto é identificada (KOCKARA et al., 2007).

São identificadas interações sobre objetos quadrados e circulares, verificados com a ponta do dedo ou qualquer ponto do contorno da região da mão. Um exemplo de interação é exibido na Figura 29, onde a imagem esquerda é a região da mão afastada do objeto e a imagem direita é a região da mão sobre o objeto.

O resultado desse método de detecção foi efetivo em todos casos que testamos, se mostrando utilizável em uma aplicação de projeção interativa. Esse tipo de detecção possui uma restrição em seu uso, pois o primeiro objeto colidido é “tocado”, fazendo com que ocorra interação com objetos que estejam no trajeto do objeto desejado.

Figura 29 – Detecção de Interação.



Fonte: gerada pelo autor.

5.9 Protótipo

O protótipo desenvolvido para exemplificar o uso da biblioteca é um jogo interativo. O jogo é composto por bolhas coloridas que surgem da lateral direita da projeção e se movem com velocidades e posições verticais aleatórias em direção ao lado oposto. O jogador deve tocar as bolhas antes que elas sumam ao encostar na lateral esquerda. Um contador de sucesso indica a quantidade de acertos, enquanto um contador de falhas indica quantas bolhas alcançaram o final da projeção. A execução do jogo é exibida na Figura 30.

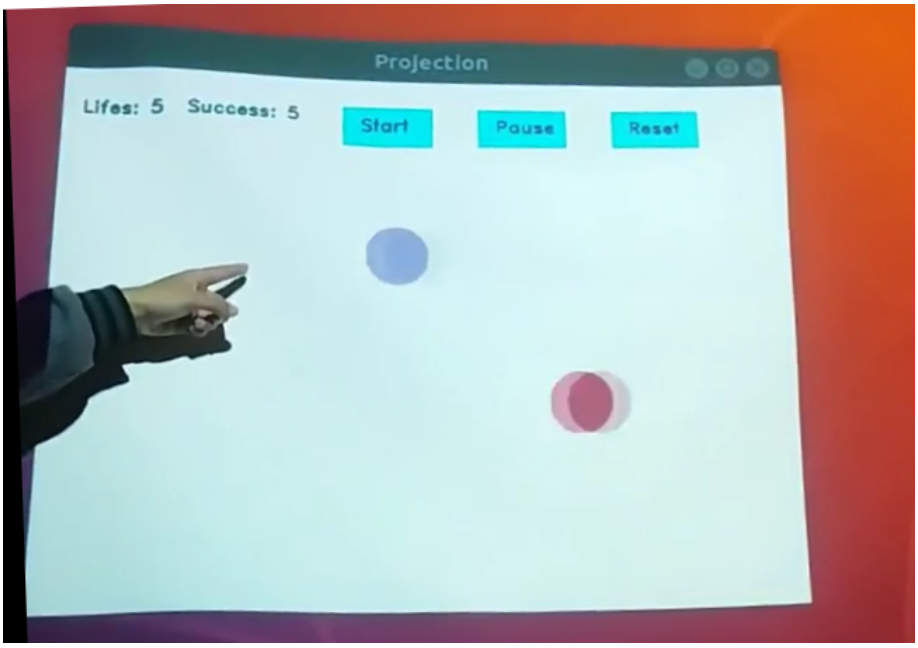
Quando é detectada uma interação com o botão "*Start*", uma contagem regressiva é iniciada para indicar o inicio da partida. Se o botão "*Pause*" for colidido, o jogo é pausado. Caso seja detectada uma interação com o botão "*Reset*", o jogo é reiniciado. Se o contador de interações bem sucedidas chegar a vinte, é exibido o estado do jogo em forma de texto no centro da projeção indicando que o jogador venceu a partida. Caso contrário, se o contador de falhas chegar a cinco, uma mensagem é exibida informando ao jogador que o jogo terminou sem obter êxito. O código fonte do protótipo é exibido na seção A.3.

5.10 Resultados

Durante o desenvolvimento do nosso trabalho enfrentamos algumas dificuldades em replicar os resultados obtidos por outros autores, em alguns casos, devido a falta de informação de como seus métodos foram desenvolvidos. Alguns desses métodos tiveram que ser descartados e outros sofreram pequenas adaptações para que funcionassem de maneira aceitável para o nosso caso específico.

A calibração geométrica, vista em vários dos trabalhos relacionados, foi implementada em nosso trabalho. A calibração se mostrou eficaz, corrigindo a imagem capturada

Figura 30 – Protótipo.



Fonte: gerada pelo autor.

de diversos ângulos e a transformando em uma imagem plana sem perda de qualidade ou forma.

A calibração fotométrica vista no trabalho de He, Cheng e Tao (2014) não foi utilizada para detectar a região da mão em nosso trabalho devido a não conseguirmos os mesmos resultados do autor. Entretanto, conseguimos identificar a região da mão utilizando um método diferente, obtendo resultados semelhantes aos vistos em seu trabalho. A região da mão foi identificada com precisão em projeções claras submetidas à condições de iluminação ambiente. Embora os resultados não tenham sido tão bons com imagens escuras em ambientes de pouca iluminação, chegaram a níveis aceitáveis e não impediram o funcionamento da aplicação.

A ponta do dedo foi identificada apresentando resultados positivos, sendo resultados dependentes da forma da região da mão identificada. Em uma identificação da região da mão em que a forma da mão é bem definida, a ponta do dedo é identificada com precisão. Essa identificação depende também da forma com que a interação é feita, sendo necessário que o um dos dedos esteja esticado enquanto os outros estejam dobrados, formando uma região aguda além da palma da mão.

O desenvolvimento da funcionalidade de reconhecer toques na superfície de projeção não foi concluído. O método implementado baseado em FAST e BRISK não obteve um resultado positivo. O método de detecção de toque baseado em sombra não obteve um resultado expressivo, embora tenha sido efetivo em alguns casos, os requisitos para

que a funcionalidade seja eficaz dificultam sua utilização. Para que fosse possível interagir com a projeção mudamos a forma como a detecção da interação deveria ocorrer, gerando resultados precisos durante os testes. Dessa forma, mesmo que o toque na projeção não seja identificado, a interação pode ocorrer baseada na distância da mão ao objeto que se deseja interagir.

O objetivo geral do trabalho era desenvolver uma biblioteca para desenvolvimento de aplicações de projeção interativa, já os objetivos específicos eram: utilizar funcionalidades vistas na literatura e combinar com métodos existentes da biblioteca OpenCV; desenvolver uma aplicação de projeção interativa utilizando a nossa biblioteca; e por fim, avaliar a complexidade e quantidade de linhas de código abstraídas do desenvolvimento de uma aplicação com o uso de nossa biblioteca.

A biblioteca foi desenvolvida com funcionalidades básicas que atendem o desenvolvimento da aplicação de projeção interativa. Combinamos funcionalidades vistas na literatura com os métodos existentes na biblioteca OpenCV, gerando nossos próprios métodos com funções específicas para o nosso domínio.

A complexidade extraída do desenvolvimento da aplicação foi bastante expressiva. Não é necessário que o usuário desenvolvedor se preocupe com a implementação da calibração geométrica, identificação da região da mão, identificação da ponta do dedo e também a detecção da interação. O código fonte da aplicação exemplo possui 20% da quantidade de linhas existentes no código fonte da biblioteca desenvolvida. Avaliamos como atingidos os objetivos desejados. Mesmo sem atender todas funcionalidades planejadas, nosso trabalho contempla os propósitos iniciais almejados.

5.11 Considerações do Capítulo

Durante o processo de pesquisa dos trabalhos relacionados já tínhamos reparado que alguns dos trabalhos não explicavam com detalhes a maneira que deveriam ser implementados para que fossem obtidos os mesmos resultados. Dessa forma já esperávamos enfrentar alguma dificuldade durante o desenvolvimento dos nossos métodos. Por este motivo pesquisamos diferentes maneiras para identificar uma interação.

O atraso do desenvolvimento dos nossos métodos devido a não conseguir replicar métodos de outros autores quase compromete o andamento do desenvolvimento. Foi importante saber o momento em que era preciso descobrir outros meios de atingir nossos objetivos.

Algumas funcionalidades vistas durante a descoberta de requisitos não foram desenvolvidas. Elas são funcionalidades interessantes e úteis para o funcionamento de um sistema de projeção interativa, embora não sejam essenciais. Conseguimos desenvolver as funcionalidades básicas para detectar uma interação em uma projeção e os resultados foram positivos.

6 CONSIDERAÇÕES FINAIS

Considerando que o principal objetivo deste trabalho era desenvolver uma biblioteca para uso em aplicações de projeção interativa, composta por funcionalidades das bibliotecas OpenCV, OpenGL e algumas implementações de soluções propostas por pesquisadores da área, durante o processo de revisão bibliográfica e análise dos trabalhos relacionados constatamos que ainda existe espaço para estudo mais profundo em diversos tópicos. Dessa forma, o desenvolvimento de aplicações de projeção interativa é uma tarefa que demanda estudo e esforço.

A maioria das aplicações disponíveis já desenvolvidas são fornecidas por empresas – que não divulgam seu modo de funcionamento de maneira aberta – e possuem um custo associado para a sua obtenção. Também vimos que algumas das soluções já utilizadas são de difícil replicação, pois alguns trabalhos apresentam explicações pouco detalhadas de seu funcionamento.

Analisamos os recentes métodos desenvolvidos por pesquisadores da área, apropriando-nos do conhecimento acerca do estado da arte em sistemas de projeção interativa e métodos utilizados na detecção e rastreamento de objetos e planos, o qual forneceu os subsídios necessários para o desenvolvimento da biblioteca proposta.

A partir do trabalho realizado, percebemos que as principais dificuldades enfrentadas durante o seu desenvolvimento foram: obter pleno conhecimento dos métodos utilizados para que fossem replicados ou ajustados ao que necessitávamos, desenvolver métodos fáceis de serem utilizados por outros pesquisadores e desenvolvedores.

Atingimos os objetivos traçados inicialmente, conduzindo o desenvolvimento de forma que alcançássemos os propósitos desejados. Desenvolvemos uma biblioteca de fácil acesso e de fácil utilização facilitando, assim, o desenvolvimento de aplicações de projeção interativa e incentivando desenvolvedores a construírem suas próprias aplicações.

6.1 Trabalhos futuros

Como visto na seção 5.10, deixamos de utilizar alguns métodos propostos por autores da literatura. Tivemos que descartar o uso de alguns modelos e adaptar outros para que se encaixassem em nosso ambiente. Como consequência desses acontecimentos identificamos alguns pontos a serem incluídos e melhorados no nosso trabalho.

A detecção do toque sobre a superfície de projeção é uma funcionalidade desejável em uma aplicação de projeção interativa. Por isso, ela se torna uma boa área de melhorias a serem implementadas. Da mesma forma que a identificação da região da mão pode ser feita de outras maneiras, dado que os resultados obtidos em imagens escuras pode ser otimizado.

Vimos nos trabalhos relacionados formas para corrigir o plano da projeção. Essa funcionalidade também faz parte do contexto da projeção interativa. No entanto, não conseguimos desenvolvê-la em tempo hábil para que fosse incluída em nossa biblioteca.

Mais pesquisa e tempo podem ser empregados em seu desenvolvimento.

Durante a descoberta de requisitos identificamos algumas funcionalidades que não foram desenvolvidas em nosso trabalho. Como identificar objetos e realizar interações com suas características, visto em Ziola et al. (2011). Essa funcionalidade foi desenvolvida com informações de sensores de profundidade combinadas aos formatos dos objetos, mas que também pode ser desenvolvida utilizando algoritmos de visão computacional.

REFERÊNCIAS

AIRES, K. **Segmentação de Planos Baseada em Homografia Afim, Fluxo Óptico e Reconstrução Métrica**. 120 p. Tese (Doutorado) — Universidade Federal do Rio Grande do Norte, 2009. Citado na página 37.

ALI, S. T.; GOYAL, K.; SINGHAI, J. Moving object detection using self adaptive gaussian mixture model for real time applications. In: **2017 International Conference on Recent Innovations in Signal processing and Embedded Systems (RISE)**. [S.l.: s.n.], 2017. p. 153–156. Citado na página 63.

AMANO, T. Projection center calibration for a co-located projector camera system. **IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops**, p. 449–454, 2014. Citado 2 vezes nas páginas 29 e 31.

ASTRACHAN, O. L. **Creating Unix Libraries**. 2003. Último accesso em abril de 2017. Disponível em: <https://users.cs.duke.edu/~ola/courses/programming/libraries.html>. Citado na página 46.

BIADGIE, Y.; SOHN, K. A. Feature detector using adaptive accelerated segment test. In: **2014 International Conference on Information Science Applications (ICISA)**. [S.l.: s.n.], 2014. p. 1–4. ISSN 2162-9048. Citado na página 36.

BRADSKI, G.; KAEHLER, A. **Learning OpenCV - Computer Vision with OpenCV Library**. 1. ed. United States of America: O'Reilly, 2008. 577 p. Citado na página 22.

CARDOZO, E. S. F. et al. Scrum and productivity in software projects: A systematic literature review. In: **Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering**. Swindon, UK: BCS Learning & Development Ltd., 2010. (EASE'10), p. 131–134. Citado na página 50.

CHEN, X. et al. Color mixing property of a projector-camera system. **Proceedings of the 5th ACM/IEEE International Workshop on Projector camera systems - PROCAMS '08**, v. 1, n. 212, p. 1, 2008. Citado na página 38.

CHENG, J. et al. Fingertip-based interactive projector-camera system. **Signal Processing**, v. 110, p. 54–66, 2015. Citado 4 vezes nas páginas 29, 30, 31 e 35.

CID, R.; JR, F.; MICHEL, J. As ferramentas do Astrônomo. p. 1–118, 2002. Último acesso em agosto de 2017. Disponível em: <http://www.telescopiosnaescola.pro.br/ferramentas.pdf>. Citado na página 34.

COMPANY, M. **3M Simply Interactive Projection System**. 2010. Último accesso em abril de 2017. Disponível em: <https://www.youtube.com/watch?v=LyArPuqrvwI>. Citado na página 44.

DEITEL, P.; DEITEL, H. **C: Como Programar**. 6. ed. São Paulo: Pearson Prentice Hall, 2011. 818 p. Citado na página 23.

GHYS, D. **Interactive Projection Mapping - Final Beta**. 2013. Disponível em: <https://www.youtube.com/watch?v=pnfIk1zIlao>. Citado 3 vezes nas páginas 31, 45 e 46.

HA, H.; KO, K. A method for image-based shadow interaction with virtual objects. **Journal of Computational Design and Engineering**, Elsevier, v. 2, n. 1, p. 26–37, 2015. Citado 2 vezes nas páginas 21 e 35.

HARRISON, C.; BENKO, H.; WILSON, A. D. OmniTouch: Wearable Multitouch Interaction Everywhere. **Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11**, p. 441, 2011. ISSN 10974199. Citado 2 vezes nas páginas 32 e 35.

HE, M.; CHENG, J.; TAO, D. Touch-sensitive interactive projection system. In: . [S.l.: s.n.], 2014. p. 436–441. Citado 18 vezes nas páginas 15, 21, 29, 30, 31, 33, 36, 37, 38, 39, 40, 41, 42, 58, 62, 65, 67 e 72.

JANKU, P. et al. Comparison of tracking algorithms implemented in opencv. In: . [S.l.: s.n.], 2016. v. 76. Citado 2 vezes nas páginas 29 e 30.

KAEHLER, A.; BRADSKI, G. **Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library**. 1st. ed. [S.l.]: O'Reilly Media, Inc., 2016. ISBN 1491937998, 9781491937990. Citado 2 vezes nas páginas 47 e 48.

KAI, N. et al. Interactive projection system based on a two- dimensional position sensitive detector. **IEEE Latin America Transactions**, v. 7850, n. 755, p. 1–7, 2010. Citado na página 21.

KOCKARA, S. et al. Collision detection: A survey. In: **2007 IEEE International Conference on Systems, Man and Cybernetics**. [S.l.: s.n.], 2007. p. 4046–4051. ISSN 1062-922X. Citado na página 70.

LEUTENEGGER, S.; CHLI, M.; SIEGWART, R. Y. Brisk: Binary robust invariant scalable keypoints. In: **2011 International Conference on Computer Vision (ICCV)**. [S.l.: s.n.], 2011. p. 2548–2555. ISSN 1550-5499. Citado na página 36.

LI, Y. et al. Real-time continuous geometric calibration for projector-camera system under ambient illumination. In: . [S.l.: s.n.], 2012. p. 7–12. Citado 3 vezes nas páginas 21, 29 e 31.

LIAO, M.; YANG, R.; ZHANG, Z. Robust and accurate visual echo cancelation in a full-duplex projector-camera system. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 30, n. 10, p. 1831–1840, 2008. Citado na página 37.

LITOMISKY, K. Consumer rgb-d cameras and their applications. **Rapport technique**, 2012. Citado na página 34.

MELO, N. et al. HIP-Storytelling: Hand Interactive Projection for Storytelling. **Interactive Storytelling - Fourth International Conference on Interactive Digital Storytelling**, v. 1, n. 1, 2011. Citado 3 vezes nas páginas 21, 22 e 43.

MOCHIO, H.; ARAKI, K. VDM ++ as a Basis of Scalable Agile Formal Software Development. **9th Overture Workshop on VDM**, 2011. ISSN 2245-2087. Citado na página 49.

OPENCV, O. S. C. V. **Contour Features**. 2018. Último acesso em outubro de 2017. Disponível em: <https://docs.opencv.org/3.4.0/dd/d49/tutorial_py_contour_features.html>. Citado na página 65.

OPENCV, O. S. C. V. **cv::BackgroundSubtractor Class Reference**. 2018. Último acesso em outubro de 2017. Disponível em: <https://docs.opencv.org/3.2.0/d7/df6/classcv_1_1BackgroundSubtractor.html>. Citado na página 63.

OSWALD, P. et al. i . Ge : Exploring New Game Interaction Metaphors with Interactive Projection. **Proceedings of the 9th International Conference on Tangible, Embedded, and Embodied Interaction - TEI '15**, p. 733–738, 2015. Citado na página 45.

PETERSEN, K. et al. Systematic mapping studies in software engineering. **12th international conference on evaluation and assessment in software engineering**, n. 1, p. 1–10, 2008. Citado na página 27.

ROCHA, H. da; BARANAUSKAS, M. **Design e avaliação de interfaces humano-computador**. [S.l.]: Unicamp, 2003. ISBN 9788588833043. Citado na página 21.

SHAH, S. et al. Hand gesture based user interface for computer using a camera and projector. In: . [S.l.: s.n.], 2011. p. 168–173. Citado 5 vezes nas páginas 21, 29, 30, 31 e 34.

SMEULDERS, A. et al. Visual tracking: An experimental survey. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 36, n. 7, p. 1442–1468, 2014. Citado 2 vezes nas páginas 29 e 30.

SOMMERVILLE, I. **Software Engineering**. 9th. ed. USA: Addison-Wesley Publishing Company, 2010. ISBN 0137035152, 9780137035151. Citado 2 vezes nas páginas 49 e 53.

SONG, P. et al. Vision-based projected tabletop interface for finger interactions. **Lecture Notes in Computer Science**, v. 4796, p. 49, 2007. ISSN 03029743. Citado 5 vezes nas páginas 29, 30, 31, 35 e 68.

STALLINGS, W. **Computer Organization and architeture: designing for performance**. 8. ed. Upper Saddle River: Pearson, 2009. 775 p. Citado na página 21.

STROUSTRUP, B. **Princípios e práticas de programação com C++**. 1. ed. Porto Alegre: Bookman, 2012. 1216 p. Citado 2 vezes nas páginas 22 e 23.

WOLFF, S. Scrum goes formal: Agile methods for safety-critical systems. In: **Proceedings of the First International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches**. Piscataway, NJ, USA: IEEE Press, 2012. (FormSERA '12), p. 23–29. ISBN 978-1-4673-1906-5. Citado 2 vezes nas páginas 49 e 50.

XIAOPENG, L. et al. A geometric correction method of plane image based on opencv. **Sensors and Transducers**, v. 165, n. 2, p. 234–238, 2014. Citado 4 vezes nas páginas 29, 31, 60 e 61.

ZHANG, Z. A Flexible New Technique for Camera Calibration (Technical Report). **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 22, n. 11, p. 1330–1334, 2002. ISSN 01628828. Citado 2 vezes nas páginas 29 e 31.

ZIOLA, R. et al. Examining interaction with general-purpose object recognition in LEGO OASIS. **Proceedings - 2011 IEEE Symposium on Visual Languages and Human Centric Computing, VL/HCC 2011**, p. 65–68, 2011. ISSN 1943-6092. Citado 4 vezes nas páginas 32, 44, 45 e 76.

Apêndices

APÊNDICE A – CÓDIGO FONTE DA BIBLIOTECA DE PROJEÇÃO INTERATIVA.

A.1 InteractiveProjection.cpp.

```
1  #include "InteractiveProjection.hpp"
2
3
4  // the constructor
5  InteractiveProjection::InteractiveProjection()
6  {
7
8  }
9
10 // the destructor
11 InteractiveProjection::~InteractiveProjection()
12 {
13
14 }
15
16 // inicializating the system
17 void InteractiveProjection::Start(int cameraId)
18 {
19     RNG rng(12345);
20
21     cap.open(cameraId);
22     if( !cap.isOpened())
23     {
24         cout << "Could not initialize video (" << cameraId << ") capture" << endl;
25         exit(0);
26     }
27
28
29     cap >> capturedImage;
30
31     // setting the camera resolution and video format
32     cap.set(CV_CAP_PROP_FOURCC,CV_FOURCC('M','J','P','G'));
33     cap.set(CV_CAP_PROP_FRAME_WIDTH, 800);
34     cap.set(CV_CAP_PROP_FRAME_HEIGHT, 600);
35
36
37     // reading chessboard image
38     chessboardMatrix = imread("images/pattern-800x600.png", ↩
           ↪ CV_LOAD_IMAGE_COLOR);
39     if (chessboardMatrix.empty())
40     {
41         cout << "Failed imread(): chesboard image not found" << endl;
42         exit(0);
```

```
43      }
44
45      boardSize.width = 9;
46      boardSize.height = 6;
47
48
49      // finding chessboard corners in chesboard image
50      foundCornersPattern = findChessboardCorners( chessboardMatrix, boardSize, ↩
            ↪ cornersPattern, CALIB_CB_ADAPTIVE_THRESH | ↩
            ↪ CALIB_CB_FAST_CHECK | CALIB_CB_NORMALIZE_IMAGE);
51
52
53      projectedImage = chessboardMatrix;
54
55      namedWindow("Projection", WINDOW_NORMAL);
56      imshow("Projection", projectedImage);
57
58      namedWindow("Detcted Interaction", WINDOW_NORMAL);
59
60      namedWindow("Captured Image", WINDOW_NORMAL);
61
62
63      // creating background subtractor model
64      model = createBackgroundSubtractorMOG2();
65      // setting to not detect shadows
66      model->setDetectShadows(true);
67      // setting tolerance matching and shadow value
68      model->setShadowValue(0);
69      model->setVarThreshold(40);
70
71      drawing = Mat::zeros( projectedImage.size(), CV_8UC3 );
72  }
73
74
75  // showing capturede image
76  void InteractiveProjection::ShowCapturedImage()
77  {
78
79      imshow("Captured Image", capturedImage);
80  }
81
82  // showing detected interaction with projection
83  void InteractiveProjection::ShowDetectedInteraction()
84  {
85
86      imshow("Detcted Interaction", drawing);
87  }
```

```
88
89  // appling background subtraction model and undistorting captured image if solicited
90  void InteractiveProjection::Apply()
91  {
92      cap >> capturedImage;
93
94
95      if(undistortImage)
96      {
97          DoUndistortCapture();
98      }
99
100     if(detectingTouch)
101     {
102
103         cap.set(CAP_PROP_AUTO_EXPOSURE, 0.25); // setting exposure to manual
104         cap.set(CAP_PROP_AUTOFOCUS, 0); // turn the autofocus off
105
106         // pass the frame to background subtraction model
107         model->apply(capturedImage, foregroundMask, doUpdateModel ? -1 : 0);
108
109
110         Mat kernel = getStructuringElement(0, Size(5,5), Point(-1,-1) );
111
112         // removing noise in mask using morphology filter
113         morphologyEx(foregroundMask, foregroundMask, MORPH_OPEN, kernel, Point↩
                ↪ (-1,-1), 1, BORDER_CONSTANT, morphologyDefaultBorderValue() );
114         morphologyEx(foregroundMask, foregroundMask, MORPH_CLOSE, kernel, Point↩
                ↪ (-1,-1), 1, BORDER_CONSTANT, morphologyDefaultBorderValue() );
115
116         /// Finding contours in mask
117         findContours( foregroundMask, contours, hierarchy, RETR_TREE, ↩
                ↪ CHAIN_APPROX_SIMPLE, Point(0, 0) );
118
119         /*
120             Find max contour area then find convex hull to this contour.
121             - find center point to this contour
122             - find convex points to this contour.
123             - find bigger distance between center and convex point.
124             The result will be a fingertip point
125         */
126         double max_area = 0;
127
128         vector<vector<Point>> hull(contours.size());
129
130         // cleaning drawing image
131         drawing = Mat::zeros( foregroundMask.size(), CV_8UC3 );
```

```
132
133         // drawing contour, fingertip and lines in hand region image
134         if(!contours.empty())
135         {
136             // finding the larger contour
137             for(int i = 0; i < contours.size(); i++)
138             {
139                 double a = contourArea( contours[i],false);
140                 if(a > max_area)
141                 {
142                     max_area = a;
143                     max_contour = i;
144                 }
145             }
146
147
148             drawContours( drawing, contours, max_contour, Scalar(255,255,255), -1, 8, ↩
                    ↪ vector<Vec4i>(), 0, Point() );
149
150             Moments mu = moments(contours[max_contour]);
151             Point centroid = Point (mu.m10/mu.m00, mu.m01/mu.m00);
152
153
154
155             Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255)↩
                    ↪ );
156
157             // drawing circle in centroid point
158             circle(drawing, centroid, 12, Scalar(255,0,0), -1, 8, 0);
159
160             convexHull(contours[max_contour], hull[max_contour], false, false);
161             // drawing lines in contour
162             drawContours( drawing, hull, max_contour, color, 4, 8, vector<Vec4i>(), 0, Point↩
                    ↪ () );
163
164
165             double max_dist = 0;
166             for(int k = 0; k< hull[max_contour].size(); k++)
167             {
168                 double dist = norm(centroid - hull[max_contour][k]);
169                 if(dist > max_dist)
170                 {
171                     max_dist = dist;
172                     fingertip = hull[max_contour][k];
173                 }
174             }
175
```

```
176                 // drawing a circle in fingertip point
177                 circle(drawing, fingertip, 12, Scalar(0,0,255), −1, 8, 0);
178
179
180             }
181
182         }
183 }
184
185 // verifying if exists a fingertip interaction in a rectangular region
186 bool InteractiveProjection::FingertipInteraction(Rect rectangle){
187     if(!contours.empty()){
188         if(rectangle.contains(fingertip))
189             return true;
190     }
191
192     return false;
193 }
194
195 // verifying if exists a interaction in a rectangular region
196 bool InteractiveProjection::RegionInteraction(Rect rectangle){
197     if(!contours.empty()){
198         for(int i = 0; i < contours[max_contour].size(); i++)
199         {
200             if(rectangle.contains(contours[max_contour][i]))
201                 return true;
202         }
203     }
204
205     return false;
206 }
207
208 // verifying if exists a fingertip interaction in a circular region
209 bool InteractiveProjection::FingertipInteraction(Point center, double radius){
210
211     double dist = norm(fingertip − center);
212     if (dist < radius)
213         return true;
214     return false;
215 }
216
217 // verifying if exists a interaction in a circular region
218 bool InteractiveProjection::RegionInteraction(Point center, double radius){
219
220
221     for(int i = 0; i < contours[max_contour].size(); i++)
222     {
```

```
223             double dist = norm(contours[max_contour][i] - center);
224             if (dist < radius)
225             return true;
226         }
227 
228         return false;
229     }
230 
231     // updating projected image with a input image
232     void InteractiveProjection::UpdateProjectedImage(Mat image)
233     {
234         projectedImage = image;
235     }
236 
237     // showing projected image
238     void InteractiveProjection::ShowProjectedImage()
239     {
240         imshow("Projection", projectedImage);
241     }
242 
243     // getting homography H
244     void InteractiveProjection::GetHomography()
245     {
246         // finding chessborad corners in captured image
247         bool foundCornersView = findChessboardCorners( capturedImage, boardSize, ↩
                ↪ cornersCameraView,
248                                     CALIB_CB_ADAPTIVE_THRESH | ↩
                                        ↪ CALIB_CB_FAST_CHECK | ↩
                                        ↪ CALIB_CB_NORMALIZE_IMAGE);
249 
250 
251         if(foundCornersView && foundCornersPattern)
252         {
253             // drawing chessboard corners in captured image
254             drawChessboardCorners( capturedImage, boardSize, Mat(cornersCameraView), ↩
                    ↪ foundCornersView );
255 
256             // finding homography relation between projected image and captured image
257             H = findHomography(cornersCameraView, cornersPattern);
258             mode = CALIBRATED;
259             cout << "Calibrated" << endl;
260 
261         }
262         // if occurs a calibration failure, the system will detect it
263         else
264         {
265             mode = DETECTION;
```

```
266             cout << "Calibration failed" << endl;
267         }
268 
269 }
270 
271 // undistorting captured image
272 void InteractiveProjection::DoUndistortCapture()
273 {
274     Mat temp = capturedImage.clone();
275 
276     /*undistorting camera image view to top center perspective*/
277     warpPerspective(temp, capturedImage, H, temp.size());
278 }
279 
280 
281 // keyboard interaction with system
282 void InteractiveProjection::KeyListener()
283 {
284     const char key = (char)waitKey(30);
285 
286     // key for quit
287     if (key == 27 || key == 'q')
288     {
289         cout << "Exit requested" << endl;
290         exit(0);
291     }
292 
293     // key for calibrate homography
294     if( cap.isOpened() && key == 'c' && mode == DETECTION)
295     {
296         mode = CAPTURING;
297         projectedImage = chessboardMatrix;
298         imshow("Projection", projectedImage);
299         GetHomography();
300         return;
301     }
302 
303     // key for calibrate homography if system is already calibrated
304     if( cap.isOpened() && key == 'c' && mode == CALIBRATED && !undistortImage)
305     {
306         mode = CAPTURING;
307         projectedImage = chessboardMatrix;
308         imshow("Projection", projectedImage);
309         GetHomography();
310     }
311 
312     // key for undistort captured image
```

```
313     if( key == 'u' && mode == CALIBRATED )
314     {
315         undistortImage = !undistortImage;
316         cout << "Toggle undistort image: " << (undistortImage ? "ON" : "OFF") << endl;
317     }
318 
319     // key for start interaction detection
320     if(key == 'd' )
321     {
322         detectingTouch = !detectingTouch;
323         cout << "Toggle detecting touch: " << (detectingTouch ? "ON" : "OFF") << endl;
324     }
325 
326     // key for change background subtractor learn factor
327     if (key == ' ')
328     {
329         doUpdateModel = !doUpdateModel;
330         cout << "Toggle background update: " << (doUpdateModel ? "ON" : "OFF") << ↩
                ↪ endl;
331     }
332 
333     // key for clear background subtractor model
334     if (key == 'l')
335     {
336         model->clear();
337         cout << "Cleaned background subtractor history." << endl;
338     }
339 }
```

A.2 InteractiveProjection.hpp

```
1 
2 #ifndef INTERACTIVE_PROJECTION_HPP_
3 #define INTERACTIVE_PROJECTION_HPP_
4 
5 #include <iostream>
6 #include "opencv2/core.hpp"
7 #include "opencv2/features2d/features2d.hpp"
8 #include "opencv2/video.hpp"
9 #include "opencv2/core.hpp"
10 #include <opencv2/core/utility.hpp>
11 #include "opencv2/imgproc.hpp"
12 #include "opencv2/calib3d.hpp"
13 #include "opencv2/imgcodecs.hpp"
14 #include "opencv2/videoio.hpp"
15 #include "opencv2/highgui.hpp"
```

```
16
17
18  #define DETECTION 0
19  #define CAPTURING 1
20  #define CALIBRATED 2
21
22
23
24  using namespace std;
25  using namespace cv;
26
27  class InteractiveProjection
28  {
29  public:
30      InteractiveProjection();
31      ~InteractiveProjection();
32      void Start(int cameraId);
33      void ShowCapturedImage();
34      void ShowDetectedInteraction();
35      void UpdateProjectedImage(Mat image);
36      void ShowProjectedImage();
37      void Apply();
38      bool FingertipInteraction(Rect rectangle);
39      bool RegionInteraction(Rect rectangle);
40      bool FingertipInteraction(Point center, double radius);
41      bool RegionInteraction(Point center, double radius);
42      void KeyListener();
43
44
45      Mat capturedImage, projectedImage, H, foregroundMask, chessboardMatrix, drawing;
46      Size boardSize;
47      vector<Point2f> cornersPattern, cornersCameraView;
48
49      int mode = DETECTION;
50
51      VideoCapture cap;
52
53      bool foundCornersPattern;
54      bool doUpdateModel = false, undistortImage = false, detectingTouch = false;
55
56      Ptr<BackgroundSubtractorMOG2> model;
57
58      RNG rng;
59
60      vector<vector<Point> > contours;
61      vector<Vec4i> hierarchy;
62
```

```
63      Point fingertip;
64      int max_contour;
65  protected:
66
67  private:
68      void GetHomography();
69      void DoUndistortCapture();
70
71
72
73  };
74
75  #endif //INTERACTIVE_PROJECTION_HPP_
```

A.3 sample.cpp

```
1
2   #include "InteractiveProjection.hpp"
3   #include <iostream>
4   #include <time.h>
5
6   using namespace std;
7   using namespace cv;
8
9   int main(int argc, const char** argv)
10  {
11      // fps
12      int frameCounter = 0, fps = 0;
13
14      RNG rng;
15
16      int time_count = 3;
17      int msec = 0, trigger = 1000; /* 1000ms */
18      clock_t before, difference;
19
20      // creating and starting interactive projection
21      InteractiveProjection interProj;
22      interProj.Start(1);
23
24      // reading projected image
25      Mat img(600, 800, CV_8UC3, Scalar(255, 255, 255));
26
27      int num_count_fail = 5, num_count_success = 0;
28
29      bool running = false, win = false, loose = false;
30
```

```
31      Point obj1(2000, 270);
32
33      Point obj2(1600, 500);
34
35      Rect count_fail(10, 50, 100,50);
36      Rect count_success(120, 50, 100,50);
37      Rect count_start(400,300, 60, 60);
38      Rect count_fps(700, 550, 60, 30);
39
40      Rect start(300, 30, 100, 50);
41      Rect pause(450, 30, 100, 50);
42      Rect reset(600, 30, 100, 50);
43
44      Scalar color1 = Scalar(120, 255, 80);
45      Scalar color2 = Scalar(10, 100, 200);
46
47      int vel1 = 18, vel2 = 18;
48
49      int radius = 35;
50
51      for(;;){
52          Mat white(600, 800, CV_8UC3, Scalar(255, 255, 255));
53
54          // updating model
55          interProj.Apply();
56
57          // listening keyboard interaction
58          interProj.KeyListener();
59
60          // showing images
61          interProj.ShowDetectedInteraction();
62          interProj.ShowCapturedImage();
63
64          if(interProj.mode == CALIBRATED){
65              interProj.UpdateProjectedImage(white);
66              interProj.ShowProjectedImage();
67          }
68
69          // verifing if detecting touch is active
70          if(interProj.detectingTouch && interProj.mode != DETECTION){
71
72
73              if(num_count_fail <= 0 && win == false){
74                  loose = true;
75              }
76
77              if(num_count_success >= 20 && loose == false){
```

```
78              win = true;
79          }
80
81          if(win){
82              putText(interProj.projectedImage, "You Win!", Point2f(350,300), ↩
                    ↪ FONT_HERSHEY_DUPLEX, 0.7, Scalar(50, 50, 50), 2, 8, false );
83          }
84
85          if(loose){
86              putText(interProj.projectedImage, "Game Over!", Point2f(350,300), ↩
                    ↪ FONT_HERSHEY_DUPLEX, 0.7, Scalar(50, 50, 50), 2, 8, false );
87          }
88
89          rectangle(interProj.projectedImage,start,Scalar(250,200,100),-1,8,0);
90          rectangle(interProj.projectedImage,pause,Scalar(250,200,100),-1,8,0);
91          rectangle(interProj.projectedImage,reset,Scalar(250,200,100),-1,8,0);
92
93          putText(interProj.projectedImage, "Start", Point2f(320,60), ↩
                ↪ FONT_HERSHEY_DUPLEX, 0.7, Scalar(50, 50, 50), 2, 8, false );
94          putText(interProj.projectedImage, "Pause", Point2f(470,60), ↩
                ↪ FONT_HERSHEY_DUPLEX, 0.7, Scalar(50, 50, 50), 2, 8, false );
95          putText(interProj.projectedImage, "Reset", Point2f(620,60), ↩
                ↪ FONT_HERSHEY_DUPLEX, 0.7, Scalar(50, 50, 50), 2, 8, false );
96
97          std::ostringstream strs1;
98          strs1 << "Lifes: " << num_count_fail;
99          std::string str1 = strs1.str();
100
101         std::ostringstream strs2;
102         strs2 << "Success: " << num_count_success;
103         std::string str2 = strs2.str();
104
105         std::ostringstream strs3;
106         strs3 << "FPS: " << fps;
107         std::string str3 = strs3.str();
108
109         rectangle(interProj.projectedImage,count_fail,Scalar(250,250,255),-1,8,0);
110         putText(interProj.projectedImage, str1, Point2f(20,70), ↩
                ↪ FONT_HERSHEY_DUPLEX, 0.7, Scalar(50, 50, 50), 2, 8, false );
111
112         rectangle(interProj.projectedImage,count_fps,Scalar(250,250,255),-1,8,0);
113         putText(interProj.projectedImage, str3, Point2f(700,570), ↩
                ↪ FONT_HERSHEY_DUPLEX, 0.6, Scalar(50, 50, 50), 1, 8, false );
114
115         rectangle(interProj.projectedImage,count_success,Scalar(250,250,255),-1,8,0);
116         putText(interProj.projectedImage, str2, Point2f(130,70), ↩
                ↪ FONT_HERSHEY_DUPLEX, 0.7, Scalar(50, 50, 50), 2, 8, false );
```

```
117
118             if(interProj.FingertipInteraction(start)){
119                 running = true;
120                 before = clock();
121             }
122
123             if(interProj.FingertipInteraction(pause)){
124                 running = false;
125             }
126
127             if(interProj.FingertipInteraction(reset)){
128                 num_count_fail = 5;
129                 num_count_success = 0;
130                 win = false;
131                 loose = false;
132                 running = false;
133                 time_count = 3;
134                 obj1.x=2000;
135                 obj1.y=270;
136                 obj2.x=1600;
137                 obj2.y=500;
138                 vel1 = 18;
139                 vel2 = 18;
140                 Point obj2(1400, 500);
141             }
142
143             if(!win && !loose){
144                 circle(interProj.projectedImage,Point(obj1.x, obj1.y), radius, color1,-1,8,0);
145                 circle(interProj.projectedImage,Point(obj2.x, obj2.y), radius ,color2,-1,8,0);
146             }
147
148             frameCounter++;
149             difference = clock() - before;
150             msec = difference * 1000 / CLOCKS_PER_SEC;
151
152             if ( msec >= trigger){
153                 before = clock();
154                 fps = frameCounter;
155                 frameCounter = 0;
156
157                 if(time_count >= 0 && running && !win && !loose)
158                     time_count--;
159             }
160
161             if(running && !win && !loose){
162                 if(time_count >=0){
```

```
163                     rectangle(interProj.projectedImage,count_start,Scalar(250,250,255)↩
                            ↪ ,-1,8,0);
164
165                     std::ostringstream strs3;
166                     strs3 << time_count;
167                     std::string str3 = strs3.str();
168                     if(time_count == 0){
169                         putText(interProj.projectedImage, "Start!", Point2f(400,300), ↩
                                ↪ FONT_HERSHEY_DUPLEX, 0.7, Scalar(50, 50, 50), 2, 8, ↩
                                ↪ false );
170                     }
171                     else{
172                         putText(interProj.projectedImage, str3, Point2f(420,300), ↩
                                ↪ FONT_HERSHEY_DUPLEX, 0.7, Scalar(50, 50, 50), 2, 8, ↩
                                ↪ false );
173                     }
174                 }
175
176                 if(obj1.x <= 0){
177                     num_count_fail--;
178                     obj1.x = 800;
179                     color1 = Scalar( rng.uniform(0, 200), rng.uniform(0,200), rng.uniform↩
                            ↪ (0,200) );
180                     obj1.y = rng.uniform(150,500);
181                     vel1 = rng.uniform(20,40);
182                 }
183                 else{
184                     obj1.x-=vel1;
185                 }
186
187                 if(obj2.x <= 0){
188                     num_count_fail--;
189                     obj2.x = 800;
190                     color2 = Scalar( rng.uniform(0, 200), rng.uniform(0,200), rng.uniform↩
                            ↪ (0,200) );
191                     obj2.y = rng.uniform(150,500);
192                     vel2 = rng.uniform(20,40);
193                 }
194                 else{
195                     obj2.x-=vel2;
196                 }
197
198                 if(interProj.RegionInteraction(Point(obj1.x, obj1.y), radius)){
199                     obj1.x = 800;
200                     color1 = Scalar( rng.uniform(0, 200), rng.uniform(0,200), rng.uniform↩
                            ↪ (0,200) );
201                     obj1.y = rng.uniform(150,500);
```

```
202                     num_count_success++;
203                 }
204
205                 if(interProj.RegionInteraction(Point(obj2.x, obj2.y), radius)){
206                     obj2.x = 800;
207                     color2 = Scalar( rng.uniform(0, 200), rng.uniform(0,200), rng.uniform↩
                            ↪ (0,200) );
208                     obj2.y = rng.uniform(150,500);
209                     num_count_success++;
210                 }
211             }
212
213             // showing projected image
214             interProj.ShowProjectedImage();
215             interProj.UpdateProjectedImage(white);
216         }
217     }
218     return 0;
219 }
```

A.4 CMakeLists.txt

```
1  cmake_minimum_required(VERSION 2.8.4)
2  project(interactive_projection)
3
4  set(EXEC_NAME sample)
5
6  find_package(OpenCV REQUIRED) #OpenCV
7
8  message(STATUS "OpenCV library status:")
9  message(STATUS " version: ${OpenCV_VERSION}")
10 message(STATUS " libraries: ${OpenCV_LIBS}")
11 message(STATUS " include path: ${OpenCV_INCLUDE_DIRS}")
12
13 set(INCLUDE_DIR ./include)
14
15 include_directories(${INCLUDE_DIR} ${OpenCV_INCLUDE_DIRS})
16
17 set(SOURCE_FILES
18   sample.cpp
19   src/InteractiveProjection.cpp)
20
21 add_executable(${EXEC_NAME} ${SOURCE_FILES})
22 target_link_libraries(${EXEC_NAME} ${OpenCV_LIBS})
```