FACULDADE DE TECNOLOGIA DE AMERICANA
Curso Superior de Tecnologia em Jogos Digitais

Rafael de Andrade

INTERFACES NATURAIS BASEADAS EM VISÃO COMPUTACIONAL APLICADAS A JOGOS DIGITAIS

Americana, SP
2016

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Jogos Digitais

Rafael de Andrade

INTERFACES NATURAIS BASEADAS EM VISÃO COMPUTACIONAL APLICADAS A JOGOS DIGITAIS

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Jogos Digitais, sob a orientação do Prof. Me. Kleber de Oliveira Andrade

Área de concentração: Visão Computacional.

Americana, SP

2016

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

Andrade, Rafael de
A57i Interfaces naturais baseadas em visão computacional aplicadas a jogos digitais. / Rafael de Andrade. – Americana: 2016.
134f.

Monografia (Graduação em Tecnologia em Jogos Digitais). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza.
Orientador: Prof. Me. Kléber de Oliveira Andrade

1. Visão computacional I. Andrade, Kleber de Oliveira II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU: 681.6

Rafael de Andrade

INTERFACES NATURAIS BASEADAS EM VISÃO COMPUTACIONAL APLICADAS A JOGOS DIGITAIS

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Jogos Digitais pelo CEETEPS/Faculdade de Tecnologia – FATEC/ Americana.
Área de concentração: Visão Computacional.

Americana, 24 de junho de 2016.

Banca Examinadora:

Kleber de Oliveira Andrade
Mestre
Faculdade de Tecnologia de Americana (FATEC-AM)

Cleberson Eugênio Forte
Doutor
Faculdade de Tecnologia de Americana (FATEC-AM)

Diógenes de Oliveira
Mestre
Faculdade de Tecnologia de Americana (FATEC-AM)

AGRADECIMENTOS

Agradeço primeiramente a Deus pela oportunidade de estar realizando este trabalho;

Ao professor Kleber de Oliveira Andrade, pelas valiosas orientações e incentivo que me foram passados;

Ao professor Marcelo Ferreira, por sua atenção, incentivo e esclarecimento de dúvidas;

Ao professor Cleberson Eugenio Forte, pelo incentivo, por me apresentar a Visão Computacional e contribuir na delimitação do tema da pesquisa.

Por fim, agradeço a todos que, de algum modo, contribuíram para o desenvolvimento deste trabalho.

RESUMO

A interação com jogos eletrônicos por meio de interfaces naturais vem alcançando atualmente grande aceitação e popularidade devido ao fato de estas proporcionarem ao jogador experiências mais intensas, imersivas e livres de frustrações. As constantes pesquisas e o desenvolvimento de novas tecnologias tem possibilitado o surgimento de produtos que permitem ao jogador interagir com os *games* através de movimentos, gestos e voz, dispensando a utilização de dispositivos artificiais como mouses, teclados ou joysticks. Tais tecnologias permitem a ampliação dos sentidos dos computadores, permitindo a eles uma melhor compreensão dos desejos dos jogadores. A capacidade de processar e extrair informações relevantes de imagens do ambiente ao seu redor possibilita aos computadores uma poderosa habilidade de interação. Este trabalho tem como objetivo a utilização do algoritmo de visão computacional Camshift no desenvolvimento de jogos digitais com interfaces naturais, fazendo uso de uma webcam. Através de um projeto prático, visa demonstrar como é possível a integração dos diversos conceitos e tecnologias envolvidas neste desenvolvimento. Os experimentos realizados com o jogo desenvolvido demonstraram que a interação com o mesmo é simples e intuitiva e ocorre de maneira razoavelmente satisfatória.

Palavras Chave: Interfaces Naturais; Visão Computacional; Jogos Digitais.

ABSTRACT

The interaction with electronic games through natural interfaces has achieved wide acceptance and popularity nowadays due to the fact they provide to the player more intense, immersive, and free of frustration experiences. The constant research and development of new technologies has enabled the emergence of products that allow players to interact with games through movements, gestures and voice, without the use of artificial devices such as computer mouses, keyboards or joysticks. Such technologies allow the expansion of the senses of computers, allowing them a better understanding of the wishes of the players. The ability to process and extract relevant information from images of the surrounding environment enables computers a powerful ability to interact. This work aims to use the computer vision algorithm Camshift to develop digital games with natural interfaces, making use of a webcam. Through a practical project, aims to demonstrate how the integration of the various concepts and technologies involved in this development is possible. The experiments with the developed game demonstrated that the interaction with the same is simple, intuitive and occurs in a reasonably satisfactory manner.

Keywords: *Natural Interfaces; Computer Vision; Digital Games.*

SUMÁRIO

LISTA DE FIGURAS

1 INTRODUÇÃO

Na indústria de *games* se tornou comum a busca por novas formas de interação que enriqueçam a experiência do usuário. As constantes pesquisas e o avanço tecnológico vêm contribuindo para o surgimento de diversas melhorias em gráficos, roteiros e modos de interação, na preocupação de maximizar a eficiência e a satisfação dos jogadores, proporcionando experiências mais intensas, imersivas e livres de frustrações (NETO, 2011).

A fim de possibilitar uma nova forma de interação com considerável aplicação nos consoles atuais (Xbox One, Playstation 4, Nintendo Wii U), as interfaces naturais permitem ao jogador a utilização do seu próprio corpo ou de objetos do ambiente real à sua volta na interação com o jogo, livre da necessidade de utilização dos tradicionais controles, teclados, mouses ou joysticks.

Para atingir tal nível de interação vem sendo amplamente utilizadas técnicas de visão computacional que possibilitam ao sistema compreender o mundo a sua volta, extraindo as informações necessárias para o sucesso da comunicação com os usuários (SZELISKI, 2010; ROCHA; SOUZA, 2014; GARBIN, 2010; PILLON, 2015; MEDEIROS, 2012; FILHO, VIEIRA, 2012).

O objetivo deste trabalho é a utilização do algoritmo de visão computacional Camshift no desenvolvimento de jogos digitais com interfaces naturais, fazendo uso de uma webcam. Para isso, visa demonstrar através de um projeto prático, como é possível a integração dos diversos conceitos e tecnologias envolvidas neste desenvolvimento. Deste modo, se espera obter respostas no ambiente virtual do jogo correspondentes aos movimentos do jogador ou de algum objeto do ambiente real captado pela webcam.

Além da introdução, este trabalho apresenta 4 capítulos: revisão bibliográfica, projeto do jogo, implementação do jogo e considerações finais. O segundo capítulo apresenta o conceito de visão computacional, com a descrição de seus principais componentes e etapas, uma introdução à área de Interação Humano-Computador (IHC) e o conceito de interfaces naturais, para, por fim, abordar a junção destes conceitos na criação de jogos digitais. No terceiro capítulo é descrito o projeto do *game* implementado neste trabalho, o que é normalmente conhecido como

documento de projeto de jogo (do inglês, *Game Design Document* – GDD). O quarto capítulo aborda o desenvolvimento do *game* proposto, com a descrição da arquitetura básica e das técnicas de visão computacional utilizadas para o funcionamento do *game*, a demonstração do código fonte destas técnicas na linguagem C#, os resultados obtidos e a discussão sobre os mesmos com base na fundamentação teórica apresentada. O quinto capítulo fecha o trabalho com as considerações finais e aponta para possíveis trabalhos futuros a partir da base obtida nesta pesquisa.

2 REVISÃO BIBLIOGRÁFICA

Esse capítulo se destina à contextualização e apresentação de conceitos fundamentais ao desenvolvimento do tema. A seção 2.1 trata do conceito de visão computacional, inicialmente vista de maneira mais ampla e geral. A seção 2.2 apresenta o conceito de sistema de visão computacional, seguindo na seção 2.3 com a descrição dos seus principais componentes de arquitetura e na seção 2.4 com as suas principais etapas. As seções 2.5, 2.6 e 2.7 abordam assuntos complementares ao entendimento do processo de visão computacional empregado no projeto prático desenvolvido no capítulo 4, sendo eles respectivamente os conceitos de espaços de cores, histogramas e distribuição de probabilidade de cor. A seção 2.8 faz uma introdução à área de Interação Humano-Computador (IHC) e a seção 2.8.1 trata do conceito de interfaces naturais na interação entre homem e computador. Finalmente a seção 2.8.2 aborda o uso da visão computacional como parte integrante do funcionamento de interfaces naturais utilizadas em jogos digitais.

2.1 VISÃO COMPUTACIONAL

Visão Computacional é a área que tem como principal objetivo o uso de computadores na emulação da visão humana, com a possibilidade de aprendizado, tomada de decisões e a realização de ações, a partir das imagens de entrada (GONZALEZ; WOODS, 2002). Seus esforços são somados a uma área mais abrangente, a de Inteligência Artificial (IA), que tem como objetivo a difícil tarefa de emular a inteligência humana (GONZALEZ; WOODS, 2002). Desta forma, a Inteligência Artificial almeja desde sua origem permitir aos computadores a capacidade de processar informações sensoriais como a visão, de maneira semelhante à capacidade humana (BALLARD; BROWN, 1982).

A percepção visual pode ser considerada como sendo a relação entre as imagens de entrada do sistema comparadas com modelos prévios do mundo, e que existe uma lacuna representacional entre o primeiro e o segundo, que precisa ser preenchida para o entendimento necessário (BALLARD; BROWN, 1982).

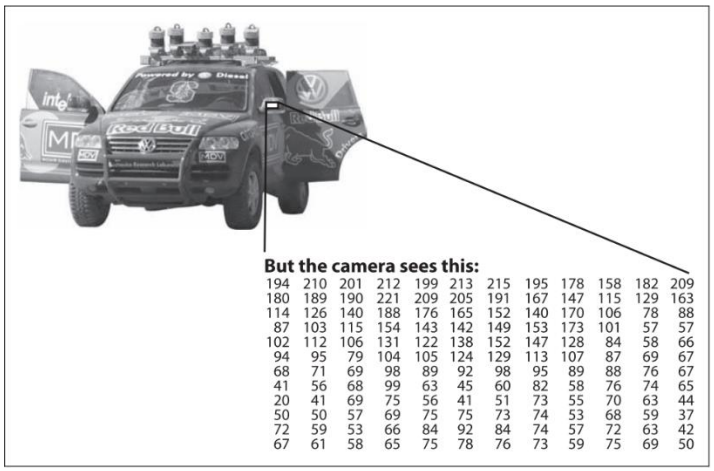
O sistema visual humano consegue entender, ou ordenar, a caótica quantidade de informações visuais que recebe como entrada. Para isso, no decorrer do seu processamento, da retina dos olhos até níveis cognitivos, faz uso de informações intrínsecas extraídas das entradas, premissas e conhecimento que são aplicados em diferentes etapas até o entendimento necessário (BALLARD; BROWN, 1982). Nesse mesmo sentido, a visão computacional tenta preencher essa lacuna, fazendo uso de um conjunto de representações intermediárias que, ligadas umas às outras, vão tecendo o entendimento das imagens de entrada, permitindo ao sistema o reconhecimento e manipulação dos objetos presentes nessas imagens (BALLARD; BROWN, 1982). Sendo possível, desse modo, que se consiga realizar a interpretação automatizada de imagens (SOLEM, 2012).

Os dispositivos de captura de informações que servem de entrada para o processamento da visão computacional podem ser de variados tipos, como câmeras de vídeo, sensores, scanners, entre outros (BALLARD; BROWN, 1982 apud MILANO; HONORATO, 2010, p. 1). Após o devido processamento se obtém informações significativas a partir destas imagens de entrada como, por exemplo, dados de modelos 3D, posição de câmera, detecção e reconhecimento de objetos presentes na imagem (SOLEM, 2012). Variados tipos de objetos ou padrões podem ser identificados, como o número de pessoas presentes em uma cena (BRADSKI; KAEHLER, 2008), o número de células em imagens médicas (FORTE *et al.*, 2012) ou os tipos de terrenos presentes em imagens captadas por satélite. Algumas realizações humanas, como o controle robótico de uma sonda em Marte, também se tornaram possíveis através da Visão Computacional. O tipo de interpretação de imagens que se faz necessário em cada caso depende dos fins específicos em que as técnicas serão utilizadas (VELLOSO; BRITO; PEREIRA, 2009).

Os humanos tendem a subestimar as dificuldades dos desafios da visão computacional durante a tarefa de interpretar o mundo a sua volta (BRADSKI; KAEHLER, 2008). Isto ocorre em virtude de serem seres visuais (BRADSKI; KAEHLER, 2008) e perceberem a estrutura tridimensional do mundo que os rodeia como uma tarefa descomplicada. Podem por exemplo, sem muita dificuldade, diferenciar os elementos presentes em uma fotografia (SZELISKI, 2010). Porém, diferentemente de um humano, uma máquina recebe no seu sistema visual somente uma matriz de números como entrada, como representado na Figura 1 onde é

possível observar a matriz de números considerada pelo computador para uma pequena região de uma imagem. Na maioria das vezes, não há facilitadores como o reconhecimento de padrões embutidos, disponibilidade de informações obtidas a partir de vivências prévias para relacionamento com os dados obtidos no momento presente e controle automático de foco e abertura (BRADSKI; KAEHLER, 2008). O tratamento adequado dos ruídos e distorções advindos de várias situações, como imperfeições das lentes dos equipamentos utilizados na captura de informações do mundo real ou compressões posteriores das informações com perda parcial dos dados, estão entre os desafios da visão computacional na percepção do mundo real (BRADSKI; KAEHLER, 2008).

Figura 1 – Informações de entrada da visão computacional.



Fonte: Bradski e Kaehler (2008, p. 2)

Além da estreita relação com IA já citada, a Visão Computacional tem laços com várias outras disciplinas. Dentre as mais próximas que também trabalham com imagens, pode-se citar o Processamento de Imagens, a Análise de Imagens e a Computação Gráfica. A Visão Computacional, por vezes acaba se confundindo com algumas delas, fazendo com que os próprios pesquisadores da área não cheguem a um consenso a respeito dos limites de cada uma, como o que acontece entre a Visão Computacional e o Processamento de Imagens (MARENGONI; STRINGHINI, 2009).

2.2 SISTEMA DE VISÃO COMPUTACIONAL

Previamente à definição do que é um sistema de visão computacional, se faz necessário considerar de maneira mais ampla, o que é um sistema de informação. De acordo com Gonçalves (2016) este pode ser genericamente considerado como sendo “todo sistema que manipula dados e gera informação, usando ou não recursos de tecnologia da informação”.

Neste trabalho, um sistema de visão computacional será considerado, por sua vez, na definição de Marques Filho e Neto (1999, p. 9), em que um sistema de visão computacional é “um sistema computadorizado capaz de adquirir, processar e interpretar imagens correspondentes a cenas reais”. A definição de sua estrutura tem como finalidade, segundo Siqueira (2014, p. 19):

> “[…] obter um conjunto de técnicas e metodologias que possam dar suporte ao desenvolvimento de teorias e produtos suficientemente eficientes e confiáveis para aplicações práticas, auxiliados pelo conhecimento de diversas áreas como a biologia, medicina, comunicação visual, eletrônica, matemática, mecânica fina.”

O autor ainda cita algumas aplicações deste tipo de sistema:

> “Como exemplo de aplicação pode-se citar a automatização dos processos de controle de qualidade, identificação e classificação de produtos e exploração de diversos ambientes” (SIQUEIRA, 2014, p. 19).

Um sistema de visão computacional apresenta, em geral, os componentes e etapas descritas nas seções a seguir.

2.3 COMPONENTES DE UM SISTEMA DE VISÃO COMPUTACIONAL

A organização de um sistema depende muito de como está sendo planejado o sistema em questão, ou seja, como ela está sendo aplicada. Questões como a rigidez do sistema quanto à possibilidade de aprendizado são importantes (CAETANO, 2009). Partindo do entendimento dessa funcionalidade, o sistema pode

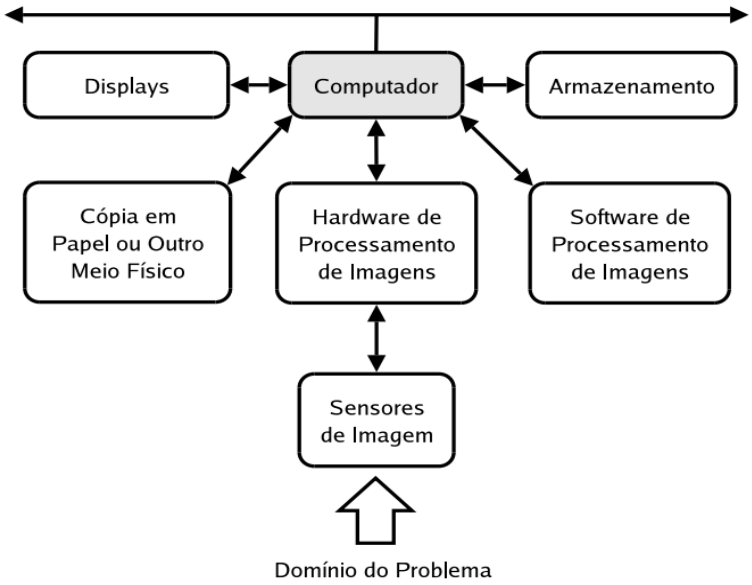
então ser definido em relação à estrutura do projeto e aos equipamentos necessários.

Existem diversos tipos de abordagens para a definição dos componentes em um sistema de visão computacional. Jähne (2009) compara os componentes aos de um sistema biológico e aponta os seguintes elementos, divididos em módulos:

- **Fonte de Radiação:** iluminação necessária para que se tornem visíveis ao sistema os objetos do ambiente em questão, principalmente aqueles que não emitem radiação por si mesmos.
- **Câmera:** guia a radiação vinda dos objetos do ambiente para o sensor. Ela é composta normalmente por componentes ópticos.
- **Sensor:** realiza o processo de transdução optoeletrônica, ou seja, converte a energia luminosa vinda dos objetos em sinal elétrico.
- **Unidade de Processamento**: converte o sinal elétrico em digital e extrai as informações de interesse que influenciam na tomada de decisões pelo sistema.
- **Atuadores:** reagem aos resultados processados pelo sistema, por exemplo, em um sistema onde um robô tem que desviar de obstáculos observados à sua frente.

Considerando um modelo de sistema com componentes mais integrados, Gonzalez e Woods (2002) colocam o computador como o centro do processo. Ele deve gerenciar as diversas atividades do sistema, desde a ativação da aquisição das imagens, passando pelo tratamento realizado pelo software de processamento de imagens, armazenamento dos dados (imagens ou informações processadas), até possibilitar a visualização dos resultados em dispositivos apropriados como monitores e outros tipos de sinalizações. A Figura 2 representa este modelo.

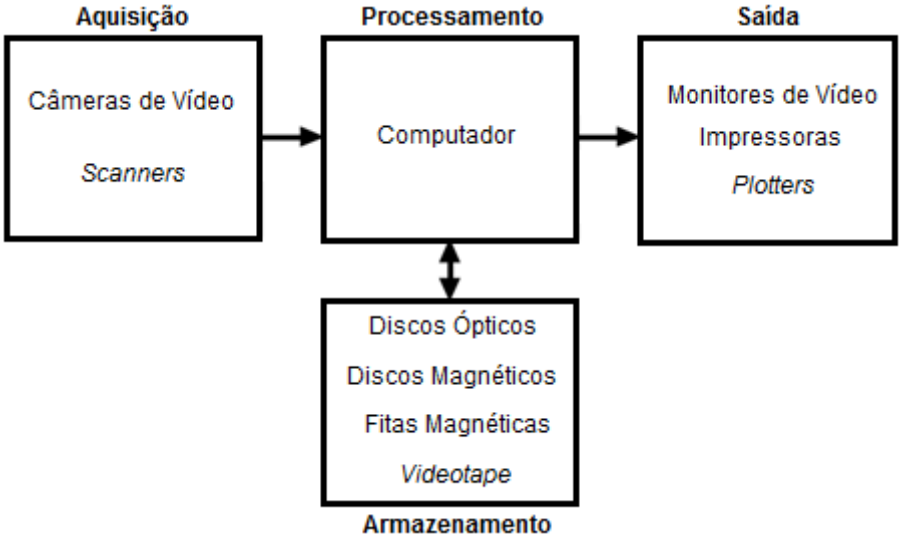
Figura 2 - Componentes de um sistema de visão computacional (1).



Fonte: Gonzalez e Woods apud Deschamps (2004, p. 22)

Marques Filho e Neto (1999, p. 2) destacam, de maneira semelhante, os componentes na Figura 3.

Figura 3 – Componentes de um sistema de visão computacional (2).



Fonte: Marques Filho e Neto (1999, p. 2)

2.4 PRINCIPAIS ETAPAS DE UM SISTEMA DE VISÃO COMPUTACIONAL

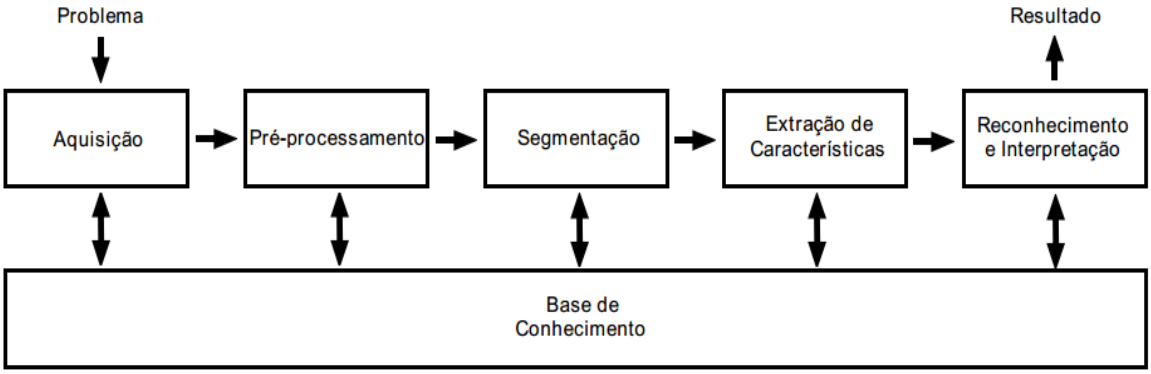
Como citado na seção 2.1, a Visão Computacional possui estreita relação com outras disciplinas, especialmente a de Processamento de Imagens, causando até a falta de consenso entre especialistas a respeito de quais seriam os limites de cada uma delas. Isto se revela quando se pesquisa sobre o assunto e se verifica que cada autor tem sua maneira de descrever e organizar as etapas do processo, desde a captura das imagens até a extração das informações necessárias.

Para Ballard e Brown (1982), a visão computacional teria o papel de preencher a lacuna existente entre a realidade e modelos já armazenados do mundo em um sistema, fazendo uso de um conjunto de representações intermediárias que, interligadas, permitem o entendimento das imagens de entrada. O autor divide essas representações em diferentes grupos, partindo de sinais mais primitivos (baixo nível) até símbolos cognitivos (alto-nível), com um aumento de abstração. Cada parte ainda pode contar com subníveis de representações e ainda cooperação entre eles. O fluxo de informações, porém, não é unidirecional e nem todos os níveis precisam ser utilizados em todos os sistemas de visão computacional. Segundo Jähne (2009), o processamento de imagens também não é um processo de uma única etapa, mas constituído de várias etapas executadas uma após a outra para que somente no final se possam extrair os dados necessários das imagens em questão. Assim como Ballard e Brown (1982), Jähne (2009) afirma que nem todos os tratamentos disponíveis precisam ser utilizados em todos os sistemas de visão computacional.

Para facilitar o entendimento a respeito do processamento realizado em visão computacional, Gonzalez e Woods (2002) sugerem a existência de três níveis de processamento. Esses níveis abrangem desde processos básicos de pré-processamento de imagens como redução de ruídos e ajuste de contraste, passando por processos como segmentação e classificação de objetos até tarefas de mais alto nível, onde os objetos reconhecidos são compreendidos e passam a “fazer sentido” para o sistema, sendo intimamente relacionadas à cognição humana ligada à visão.

Os autores Marques Filho e Neto (1999, p. 10) apontam na Figura 4 as etapas de um sistema de visão computacional. As próximas seções detalham cada etapa da Figura.

Figura 4 – Principais etapas de um sistema de visão computacional.



Fonte: Marques Filho e Neto (1999, p. 9)

2.4.1 AQUISIÇÃO

Uma imagem digital pode ser entendida como uma função bidimensional $f(x,y)$ obtida após um processo de discretização espacial e em amplitude (MARQUES FILHO; NETO, 1999), onde x e y dizem respeito as coordenadas espaciais e o valor ou amplitude de f em qualquer ponto de (x,y) se mostra proporcional à intensidade luminosa (como brilho ou nível de cinza) da imagem naquele ponto. A imagem digital é processada pelo computador como uma matriz. Seus índices de linhas e colunas apontam para coordenadas ou pontos da imagem e o valor deste elemento indicado da matriz corresponde a sua intensidade. Cada um destes pontos da imagem é conhecido como elemento da imagem ou *pixel* (abreviação de *picture element*) (TAKEMURA; DRUCKER, 2014).

A função $f(x,y)$ que representa uma imagem digital pode ser caracterizada por dois elementos:

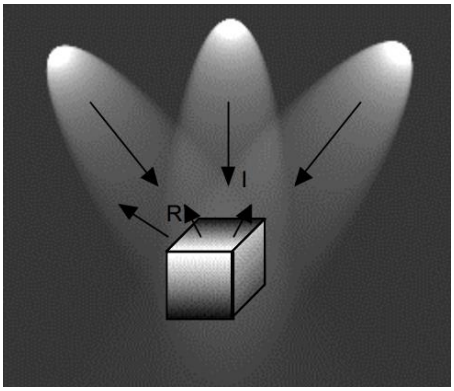
- **Iluminância** $i(x,y)$**:** exprime a quantidade de luz incidente sobre o objeto.
- **Refletância** $r(x,y)$**:** exprime a quantidade de luz refletida pelo objeto.

O produto da interação entre estes dois elementos resulta na fração de luz que chega ao sensor no ponto (x,y) e se encontra representado na função $f(x,y)$ na equação (MARQUES FILHO; NETO, 1999):

$$f(x,y) = i(x,y) \cdot r(x,y)$$

A Figura 5 ilustra os componentes iluminância (I) e refletância (R) de uma imagem. No caso de o objeto iluminado absorver toda a luz incidente o valor de $r(x,y)$ é nulo e no caso de o objeto refletir toda a luz incidente $r(x,y)$ vale 1. Outro ponto importante é que a natureza da iluminância é determinada pela fonte de luz, enquanto que a natureza da refletância é determinada pelos objetos que recebem a incidência da luz (GONZALEZ; WOODS, 2002).

Figura 5 – Os componentes iluminância (I) e refletância (R) de uma imagem.



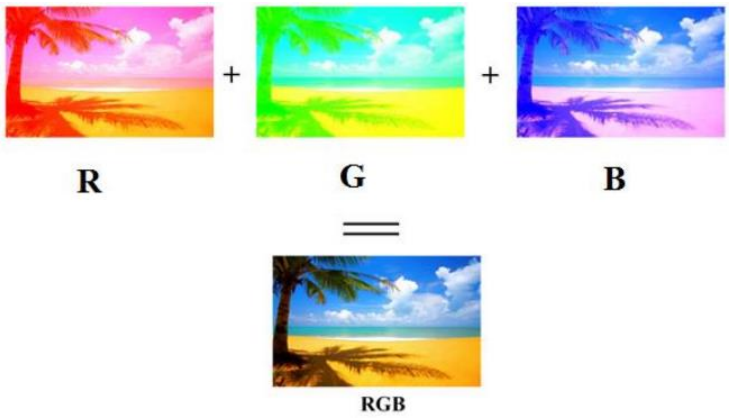
Fonte: Marques Filho e Neto (1999, p. 20)

Nas imagens coloridas no espaço de cor RGB, um dos diversos sistemas existentes para representação digital de cores (mais detalhes na seção 2.5), um *pixel* representa um vetor e cada canal deste contém as intensidades de vermelho (R – do inglês, *Red*), verde (G – do inglês, *Green*) e azul (B – do inglês, *Blue*), com uma função $f(x,y)$ própria (VILAR, 2014). Deste modo, a definição de cor de cada *pixel* é encontrada com a combinação destas três cores primárias e pode ser representada por:

$$f(x,y) = fR(x,y) + fG(x,y) + fB(x,y)$$

A Figura 6 ilustra a combinação dos canais R, G e B na formação de uma imagem colorida no espaço de cores RGB. Segundo Ballard e Brown (1982), aquisição de imagem é o processo de registrar a radiação refletida dos objetos físicos presentes no ambiente. Normalmente essa radiação é convertida para sinais elétricos em um sensor e posteriormente é digitalizada, para que seja possível o seu manuseio por computadores no formato de imagem digital.

Figura 6 – Canais R, G e B de uma imagem, separadamente, e a imagem colorida RGB resultante da composição destes três canais.

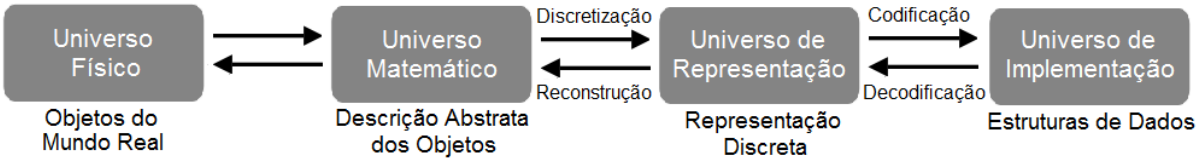


Fonte: Vilar (2014, p. 17)

O Paradigma dos Quatro Universos, representado na Figura 7, pode ser utilizado na compreensão do processo de aquisição de imagens. Nele, como o próprio nome diz, existem quatro universos representados:

- **Universo Físico:** contém os objetos do mundo real.
- **Universo Matemático:** contém as descrições abstratas dos objetos.
- **Universo de Representação:** torna possível o mundo digital a partir das descrições abstratas, através da discretização dos sinais contínuos.
- **Universo de Implementação:** onde é feita a codificação do sinal discretizado na memória do computador através de estruturas de dados (SCURI, 2002).

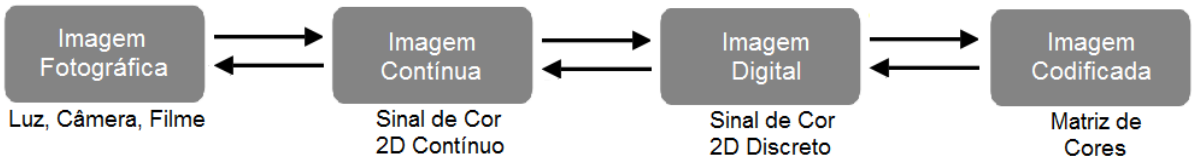
Figura 7 – O Paradigma dos Quatro Universos.



Fonte: Scuri (2002, p. 17)

A Figura 8 ilustra o funcionamento da aquisição de imagens do ponto de vista do Paradigma dos Quatro Universos.

Figura 8 – Aquisição de imagens de acordo com o Paradigma dos Quatro Universos.



Fonte: Scuri (2002, p. 18)

Por meio de câmeras ou sensores são captadas faixas de energia do espectro eletromagnético, como raios-X, ultravioleta, espectro visível ou raios infravermelhos (JÄHNE; HAUBECKER, 2000). Esta energia luminosa é então convertida em sinais elétricos proporcionais à energia luminosa recebida em um processo conhecido como transdução optoeletrônica, onde ocorre uma redução de dimensionalidade. O resultado é uma imagem analógica, onde uma cena real de três dimensões (3D) é representada por uma imagem de duas dimensões (2D) (MARQUES FILHO; NETO, 1999), constituindo então um sinal de cor 2D contínuo (SCURI, 2002).

Estas imagens no formato analógico, porém, não podem ainda ser processadas por computadores, devendo para isso serem transformadas em imagens digitais. Com este propósito ocorre então o processo de digitalização de imagem, quando através de um digitalizador, o sinal elétrico analógico é transformado em uma imagem digital (MARQUES FILHO; NETO, 1999). Esta digitalização se caracteriza exatamente pela discretização dos sinais contínuos vindos do mundo real (SCURI, 2002), neste caso do sinal de cor 2D contínuo em um sinal de cor 2D discreto, assim como explicam Young, Gerbrands e van Vliet (1998, p. 2):

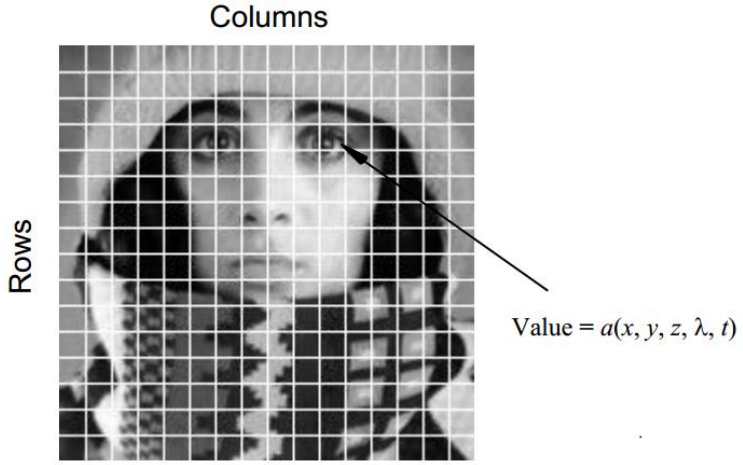
> “A digital image a[m,n] described in a 2D discrete space is derived from an analog image a(x,y) in a 2D continuous space through a sampling process that is frequently referred to as digitization”

O processo de digitalização de imagens é realizado por meio de dois passos:

- **Amostragem:** discretização das coordenadas espaciais (x, y) de uma imagem, onde x e y são números inteiros (TAKEMURA; DRUCKER, 2014). A

partir da imagem analógica se obtém uma matriz de M por N pontos (amostras), como se colocássemos uma grade sobre o plano da imagem contínua original, dividindo-a em linhas e colunas e formando pequenos pontos, os *pixels* (MARQUES FILHO; NETO, 1999). O processo de amostragem é ilustrado na Figura 9.

Figura 9 – Processo de amostragem.



Fonte: Young e Gerbrands e van Vliet (1998, p. 3)

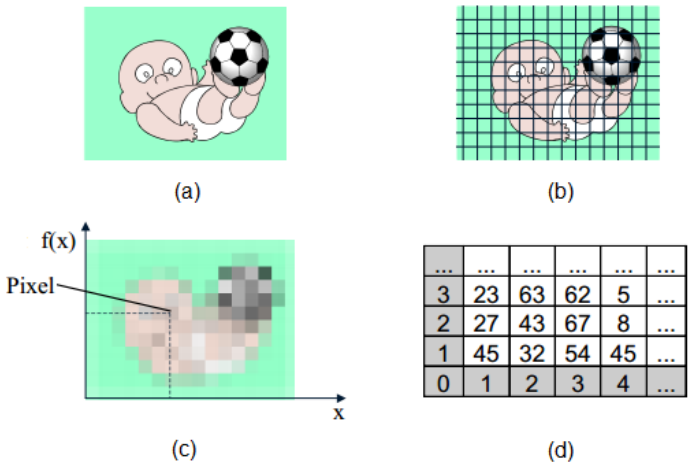
- **Quantização:** discretização da amplitude (também entendida como discretização da cor) que define o valor de cada *pixel* da imagem (TAKEMURA; DRUCKER, 2014). Em uma imagem monocromática esses valores correspondem ao brilho ou nível de cinza de cada ponto (MARQUES FILHO; NETO, 1999).

Na Figura 10 é demonstrado um exemplo completo do processo de discretização de imagens.

Marques Filho e Neto (1999, p. 9) apontam como decisões de projeto importantes relacionadas à etapa de aquisição de imagens:

> "[…] a escolha do tipo de sensor, o conjunto de lentes a utilizar, as condições de iluminação da cena, os requisitos de velocidade de aquisição […], a resolução e o número de níveis de cinza da imagem digitalizada, dentre outros."

Figura 10 – Processo de Discretização de uma imagem. (a) Imagem Contínua, (b) Amostragem, (c) Quantização, (d) Codificação.



Fonte: Scuri (2002, p. 20)

2.4.2 PRÉ-PROCESSAMENTO

Segundo Marengoni e Stringhini (2009), o pré-processamento tem a função de facilitadora das etapas posteriores, permitindo uma melhor extração das informações das imagens no processo de visão computacional. Marques Filho e Neto (1999, p. 9-10) também apontam a importância dessa fase:

> "A função da etapa de pré-processamento é aprimorar a qualidade da imagem para as etapas subsequentes." […] "A imagem resultante desta etapa é uma imagem digitalizada de melhor qualidade que a original."

Szeliski (2010, p. 101) coloca o cuidado com o design destes momentos iniciais do processamento de imagens como fundamental para a obtenção de resultados aceitáveis na maioria das aplicações de visão computacional:

> "While some may consider image processing to be outside the purview of computer vision, most computer vision applications, such as computational photography and even recognition, require care in designing the image processing stages in order to achieve acceptable results".

Essa facilitação proporcionada pelo pré-processamento às etapas seguintes pode ocorrer, por exemplo, através da conversão de formato, tamanho ou filtragem das imagens para a remoção de ruídos provenientes do processo de aquisição das mesmas. Por ruídos pode-se entender não somente interferências no sinal de captura de imagem, mas também outras situações que acabam sendo um problema na interpretação ou reconhecimento de objetos na imagem. Sua origem é variada, podendo ser o tipo de sensor utilizado, a iluminação do ambiente, as condições climáticas ou posição relativa entre a câmera e o objeto em questão (MARENGONI; STRINGHINI, 2009), como demonstrado na Figura 11.

Figura 11 – Imagens de árvores obtidas em condições diferentes.

Fonte: Marengoni e Stringhini (2009, p. 128)

Observa-se no topo da Figura à esquerda uma imagem “normal”, no topo à direita uma imagem com interferência de iluminação, embaixo à esquerda a interferência do período do ano e embaixo à direita a mudança do tipo de sensor.

O pré-processamento é normalmente classificado como uma etapa de baixo-nível, como explica Marques Filho e Neto (1999), pois as operações efetuadas nesta etapa trabalham sem levar em conta qual é o objeto de interesse, o que o compõe ou o que está ao fundo, mas sim trabalham diretamente com os valores de intensidade dos *pixels* da imagem, geralmente com a utilização de filtros.

As filtragens utilizadas são normalmente classificadas em dois grupos:

- **Filtragem Espacial:** atua diretamente na matriz de *pixels* da imagem, normalmente com operações de convolução com máscaras, que funcionam como uma região de influência aos *pixels* da imagem (MARQUES FILHO;

NETO, 1999). Nas operações de convolução, as máscaras ou elementos estruturantes (*kernels*), possuem normalmente alguns poucos *pixels* de tamanho. Esta máscara percorre a imagem *pixel* a *pixel*, partindo do canto superior esquerdo até o canto inferior direito, fixando novos valores aos *pixels* que estão no seu centro a cada mudança de posição. O resultado deste processo é a formação de uma nova imagem, com características mais propícias ao sucesso das fases posteriores da visão computacional (LAGANIÈRE, 2011).

- **Filtragem no Domínio da Frequência:** a imagem primeiramente é convertida ao domínio de frequência com o uso da transformada de Fourier, filtrada neste domínio e finalmente convertida de volta para o domínio de espaço (MARENGONI; STRINGHINI, 2009).

2.4.3 SEGMENTAÇÃO

Após basicamente a eliminação de ruídos e melhoria no contraste das imagens ocorridos na etapa anterior, chega o momento da segmentação das imagens. Esse é o momento onde a imagem é particionada em suas regiões ou objetos constituintes (GONZALEZ; WOODS, 2002), quando ocorre a divisão da imagem em suas unidades significativas, os objetos de interesse. A segmentação é considerada uma das etapas mais difíceis em Processamento de Imagens (MARQUES FILHO; NETO, 1999)

O processo é geralmente determinado pelas características do objeto ou região, como a cor ou a proximidade dos elementos na imagem. O nível de detalhamento das imagens depende do caso em questão, por exemplo, se é procurada uma casa em uma foto tirada de uma rua, estão sendo buscadas grandes regiões em uma imagem e se por outro lado é procurada uma casa em uma imagem de satélite, estão sendo buscadas pequenas regiões, visto que a resolução da imagem é diferente em cada caso. Deste modo, os tratamentos de segmentação necessários também variam de acordo com o caso (MARENGONI; STRINGHINI, 2009).

Pode-se considerar basicamente 3 tipos de operações de segmentação:

- **Por Detecção de Borda:** particiona a imagem baseada em mudanças abruptas no nível de intensidade dos *pixels*.
- **Por Corte:** particiona a imagem baseada nos valores de intensidades ou propriedades destes valores, identificados a partir do histograma da imagem, onde são identificadas as suas regiões e as suas diferentes características.
- **Por Crescimento de Região:** nesse procedimento são agrupadas regiões da imagem de acordo com a sua similaridade. A partir de um conjunto de pontos (sementes), são avaliadas as condições de crescimento pré-determinadas e realizados os agrupamentos (MARENGONI; STRINGHINI, 2009).

De maneira geral, é possível dizer que quanto melhor for a segmentação, melhor será o processo posterior de reconhecimento (GONZALEZ; WOODS, 2002).

2.4.4 EXTRAÇÃO DE CARACTERÍSTICAS

O objetivo dessa etapa é extrair características das imagens resultantes da segmentação com o uso dos descritores que permitam caracterizar os elementos desejados da imagem. Estes descritores devem possuir um alto nível de discriminação entre elementos parecidos e sua estrutura de dados deve ser compatível com o algoritmo de reconhecimento (MARQUES FILHO; NETO, 1999).

Importantes informações podem ser obtidas nesse momento como: número total de objetos na cena, propriedades geométricas como a área, o perímetro, o centro de gravidade e largura, características sobre a forma como a circularidade e a concavidade, nível de cinza e desvio padrão do nível de cinza de cada região, a textura, etc.

2.4.5 RECONHECIMENTO E INTERPRETAÇÃO

Reconhecimento é o processo em que se atribui um rótulo a um objeto a partir dos descritores das suas características, enquanto que a interpretação é responsável por atribuir significado a conjuntos de objetos reconhecidos (MARQUES FILHO; NETO, 1999).

O reconhecimento de um objeto pode se realizar a partir do reconhecimento de alguns de seus padrões como: textura, forma, cor, dimensões, etc. Um exemplo seria o reconhecimento de padrões em imagens baseando-se nos valores dos canais R, G e B dos *pixels* que as compõe (FORTE, 2015).

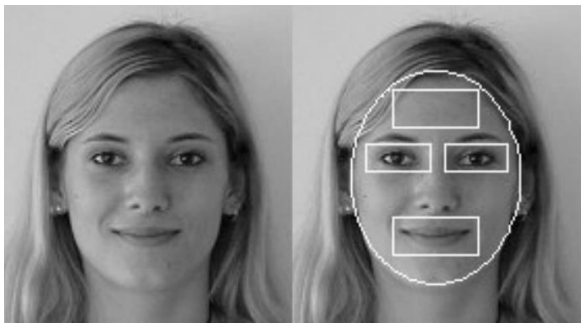
Como o próprio nome indica, reconhecer demanda um conhecimento prévio do objeto a ser reconhecido, significa conhecer de novo através de uma base de conhecimento. O trabalho com esta base relaciona a Visão Computacional com a área de Inteligência Artificial (IA). Este suporte de conhecimento pode ser implementado diretamente como regras no código ou aprendido de um conjunto de amostras dos objetos de interesse, a partir de técnicas de aprendizado de máquina (MARENGONI; STRINGHINI, 2009). As informações presentes na base podem ser desde simples orientações de regiões mais prováveis da imagem para ocorrência dos objetos buscados, até informações mais complexas como imagens em alta resolução de satélites em conexão com aplicações de detecção de mudança (GONZALEZ; WOODS, 2002).

Pode-se dividir as técnicas de reconhecimento de padrões em dois grandes grupos segundo Marengoni e Stringhini (2009):

- **Estruturais:** possuem padrões descritos simbolicamente com estruturação baseada em como os padrões se relacionam.
- **Baseadas em Propriedades Quantitativas:** possuem padrões descritos pelas propriedades quantitativas e utilização de técnicas de avaliação da existência no objeto destas propriedades.

Como exemplo de reconhecimento de elementos em uma imagem, a Figura 12 apresenta o reconhecimento de um rosto, assim como de algumas de suas partes.

Figura 12 – Reconhecimento de um rosto e suas partes.



Fonte: Marengoni e Stringhini (2009, p. 144)

2.5 ESPAÇOS DE CORES

Um modelo de cor, também conhecido como espaço de cor ou sistema de cor, é um modo de representação tridimensional em que cada cor é posicionada dentro de um sistema de coordenadas 3D. Este modelo objetiva padronizar a especificação de cores de uma forma aceita por todos (MARQUES FILHO; NETO, 1999).

Devido à inexistência de um modelo que abranja todos os aspectos referentes às cores, existem diversos modelos, cada um sendo mais adequado para a análise de algumas características específicas (ALVES, 2010), sendo sua eficiência dependente da aplicação onde são empregados (OYAMA *et al.*, 2012).

Como exemplos de espaços de cores, pode-se citar: RGB, CMY, CMYK, YCbCr, YIQ e HSV (MARQUES FILHO; NETO, 1999).

Nos próximos itens serão abordados os dois espaços de cores utilizados no projeto prático deste trabalho: RGB e HSV.

2.5.1 MODELO RGB

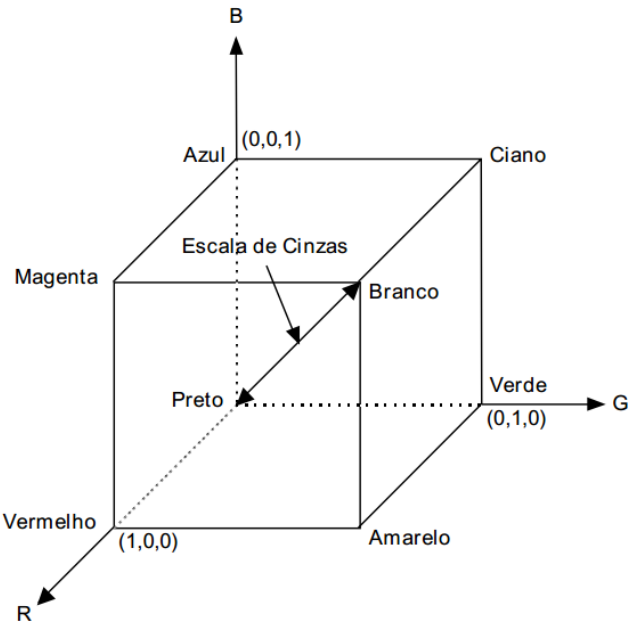
A representação do sistema RGB pode ser vista como um cubo no qual três vértices são ocupado pelas cores primárias: vermelho (R – do inglês, *Red*), verde (G – do inglês, *Green*) e azul (B – do inglês, *Blue*). Outros três vértices são para as cores secundárias e os outros dois representam o preto e o branco (MARQUES

FILHO; NETO, 1999). A Figura 13, com valores de R, G e B normalizados na faixa de 0 a 1, demonstra este cubo.

A cor de cada ponto do interior do cubo é obtida considerando as coordenadas das três cores primárias, com os valores de intensidade de R, G e B variando de 0 a 255. Conforme se aumenta ou diminui igualmente as contribuições das três cores primárias se obtêm variações nos tons de cinza ao longo da diagonal principal do cubo, que vai do vértice que representa a cor preta até o vértice que representa a cor branca (ALVES, 2010).

O RGB é o modelo de cor mais utilizado em câmeras e monitores de vídeo (MARQUES FILHO; NETO, 1999).

Figura 13 – Modelo RGB.



Fonte: Marques Filho e Neto (1999, p. 121)

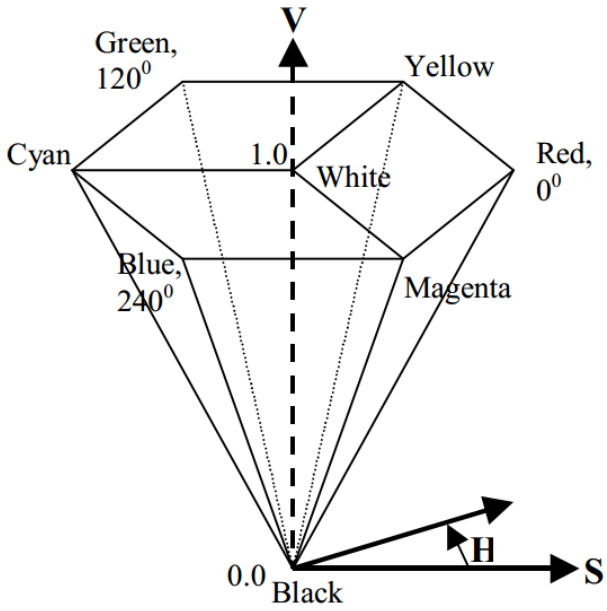
2.5.2 MODELO HSV

A sigla HSV deriva de suas componentes: matiz ou tonalidade (do inglês, *hue*), saturação (do inglês, *saturation*) e valor, brilho ou intensidade (do inglês, *value*).

O matiz (H) diz respeito ao tipo da cor ou a cor em si. A saturação (S) mede a concentração ou pureza da cor (quantidade de cor branca misturada) (GAGLIARDI, 2014), com mais tons de cinza aparecendo para menores purezas (UFRGS, 2016). Por fim, o valor (V) mede a intensidade de luz ou brilho presente (GAGLIARDI, 2014).

A representação do sistema HSV pode ser vista na Figura 14 como uma pirâmide de base hexagonal, no qual a cor (matiz) é definida pelo ângulo H de sua base, a saturação S é definida pela distância do centro até a borda e o valor V é definido pela distância vertical ou profundidade (EVANGELISTA; SILVA, 2007).

Figura 14 – Modelo HSV.



Fonte: Bradski (1998, p. 3)

De acordo com Ramos (2012), o espaço de cor HSV é uma transformação não linear do espaço de cor RGB.

A utilização do modelo HSV de cor em sistemas de visão artificial se justifica pelo fato de ser possível separar as suas componentes H, S e V, isolando uma da influência das outras, de acordo com a necessidade (MARQUES FILHO; NETO, 1999). Por exemplo, considerando-se unicamente o matiz (H) se eliminam variações de iluminação e outros efeitos, uma vez que este é afetado com menor intensidade por esses elementos. Deste modo, em um algoritmo de visão computacional que

considere a cor dos objetos no seu processamento, considerando-se uma pequena região de H como a cor do objeto de interesse, pode-se rastreá-lo de maneira mais eficaz. (GAGLIARDI, 2014). A Figura 15 exemplifica a separação das componentes H, S e V de uma imagem:

Figura 15 – Separação das componentes H, S e V de uma imagem.

Fonte: Gagliardi (2014, p. 7)

2.6 HISTOGRAMAS

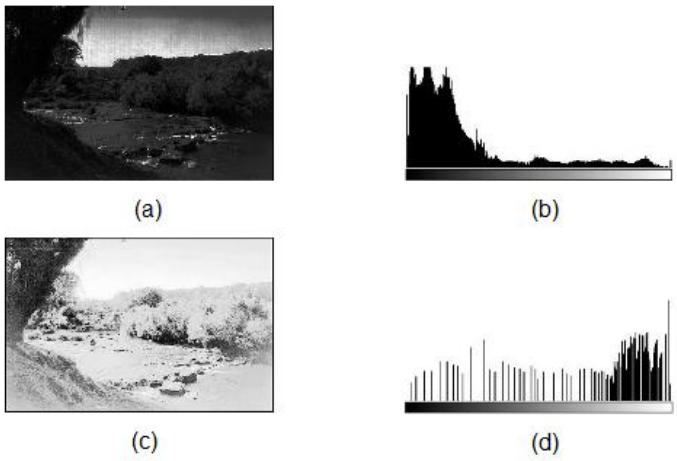
Um histograma é um conjunto de números que indica o percentual de *pixels* de uma imagem que possuem um nível de cinza específico. Normalmente ele é representado por um gráfico de barras que demonstra o número ou percentual de *pixels* para cada nível de cinza na imagem.

Histogramas são ferramentas de grande aplicação prática na área de processamento de imagens, como na melhora da definição, compressão, segmentação e descrição de imagens (MARENGONI; STRINGHINI, 2009).

Através da observação de um histograma pode-se obter a indicação da qualidade da imagem quanto ao nível de contraste e quanto ao seu brilho médio, ou seja, se ela é predominantemente clara ou escura. Para computar o histograma de uma imagem monocromática, por exemplo, percorre-se a imagem *pixel* a *pixel* e se incrementa em um vetor a posição de índice correspondente ao tom de cinza do

pixel (MARQUES FILHO; NETO, 1999). Na Figura 16 são exibidos dois exemplos de imagens e seus histogramas.

Figura 16 – Imagens e seus Histogramas. (a) Imagem escura, (b) Histograma da imagem escura (c) Imagem clara, (d) Histograma da imagem clara.



Fonte: Adaptado de Marques Filho e Neto (1999, p. 57)

Pode-se observar que pelo fato de a Figura 16.a ser predominantemente escura, o seu histograma representado na Figura 16.b apresenta grande concentração de *pixels* em níveis mais baixos de cinza, localizados mais à esquerda no histograma. Já na Figura 16.c, por ser uma imagem mais clara, os *pixels* estão concentrados em valores próximos ao limite superior da escala de cinza, localizados mais à direita no seu histograma representado na Figura 16.d (MARQUES FILHO; NETO, 1999).

2.7 DISTRIBUIÇÃO DE PROBABILIDADE DE COR

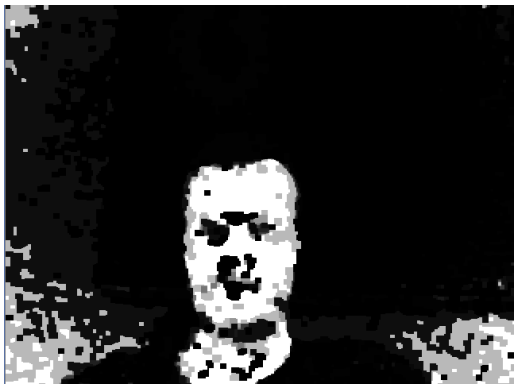
O objetivo da distribuição de probabilidade de cor é demonstrar a probabilidade de um determinado *pixel* de uma imagem pertencer a uma cor ou objeto de interesse escolhido previamente. Baseando-se no histograma deste objeto ou região pré-selecionada, tido então como um modelo de cor a ser rastreado, a distribuição de probabilidade de cor é obtida com a utilização de um algoritmo de projeção reversa de histograma (*histogram backprojection*) (LAGANIÈRE, 2011).

Este processo pode ser considerado para cada quadro de um vídeo captado por uma câmera, por exemplo.

A distribuição de probabilidade pode ser representada por uma imagem. Quando apresentada em escala de cinza, a intensidade de cada *pixel* da imagem aponta a probabilidade da cor neste ponto, com maiores intensidades de cor para maiores probabilidades. A cor preta seria a probabilidade nula (valor 0) e a cor branca seria a probabilidade total (valor 255) (GAGLIARDI, 2014).

A Figura 17 apresenta um exemplo de imagem de distribuição de probabilidade, obtida após ter sido escolhida a cor da pele da face de uma pessoa como região contendo a cor a ser considerada.

Figura 17 – Imagem de distribuição de probabilidade.



Fonte: Próprio autor

2.8 INTERAÇÃO HUMANO-COMPUTADOR (IHC)

Interação Humano-Computador (IHC) é a disciplina que estuda as comunicações ou interações entre usuários e computadores, com as responsabilidades de elaboração do projeto (design), avaliação e implementação de sistemas computacionais interativos para uso humano, estudando também os fenômenos relacionados a essa interação (REBELO, 2016).

Considerando principalmente o ponto de vista do usuário, a IHC estuda as suas ações realizadas na utilização da interface de um sistema e as interpretações que o usuário faz das respostas devolvidas pelo sistema por meio desta interface

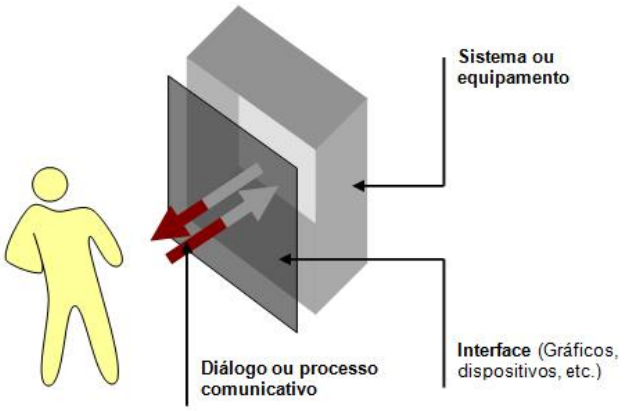
(PRATES; BARBOSA, 2003). Como interface pode-se considerar a parte do sistema com a qual o usuário mantém contato na sua utilização (PRATES; BARBOSA, 2003), sendo tanto um dispositivo de entrada de dados quanto responsável pelo retorno transmitido ao usuário como consequência de suas atividades (REBELO, 2016). Pode-se dizer, portanto, que a interface é quem permite os processos de interação (REBELO, 2016). A interface pode ser entendida como sendo o sistema de comunicação utilizado na interação entre homem e computador (PRATES; BARBOSA, 2003). Ela engloba *software* e componentes de entrada e saída, como teclados, mouses, *tablets*, monitores, impressoras e outros (PRATES; BARBOSA, 2003). A Figura 18 ilustra este processo de interação entre homem e computador.

Na sua origem, a IHC era centrada no estudo da usabilidade, visando produtos fáceis de usar, eficazes e agradáveis do ponto de vista do usuário. Com o seu avanço ela passou a ter uma maior abrangência de estudo e se preocupar com o entendimento, projeto e avaliação de um amplo leque de aspectos da experiência do usuário (ROGERS; SHARP; PREECE, 2013).

A área de IHC é caracterizada pela multidisciplinaridade, requerendo o envolvimento de vários tipos de conhecimentos, como a Psicologia, que traz à IHC o entendimento sobre as condições humanas como habilidades e limitações. Outras disciplinas como a Sociologia, Antropologia, Sistemas de Informação, Ciência da Computação, Design Gráfico e Ergonomia também contribuem para o desenvolvimento da área de IHC (REBELO, 2016). Pode-se notar também grandes contribuições da IHC para essas disciplinas, como a criação e a ampla utilização de ferramentas de visualização, busca, compilação e análise de informação (ROCHA; BARANAUSKAS, 2003).

Projetar uma IHC se refere à construção de interfaces com alta qualidade, a partir de métodos, modelos e diretrizes pré-definidos (PRATES; BARBOSA, 2003), partindo da definição do sistema interativo do qual a interface fará parte como um todo. O projeto da funcionalidade de uma interface deve se basear em um conhecimento prévio a respeito do usuário. Questões importantes como as forças e fraquezas gerais do sistema de processamento de informações humano, nível de habilidade, conhecimento específico e educação dos usuários, devem ser levadas em conta (ZILSE, 2004).

Figura 18 – Processo de interação entre homem e computador.



Fonte: Rebelo (2016)

2.8.1 INTERFACE NATURAL DO USUÁRIO

A evolução das interfaces utilizadas na interação entre homem e computador passou por diversas etapas. No processamento de dados por lote em *mainframes,* até por volta de 1960, o usuário furava cartões de papel em determinadas posições, e a comunicação, portanto, se baseava na presença ou não de furos em determinadas posições destes cartões. Havia também a possibilidade de entrada de dados por meio de fitas magnéticas. O processo era pouco interativo, pois uma vez que o usuário entrava com os dados ele deveria aguardar até que o processamento fosse finalizado (GARBIN, 2010).

Nos anos 1970, os *mainframes* apresentavam interfaces com alguma interatividade, permitindo aos usuários a interação com o computador através da digitação de comandos de texto a fim de realizar tarefas específicas. Este tipo de interface ficou conhecido como *Command-Line Interfaces* (CLI) (Interface de Linha de Comando) (GARBIN, 2010).

A necessidade de memorização dos comandos que deveriam ser digitados na CLI e as dificuldades de universalização devido à variedade de idiomas diferentes, contribuíram para o surgimento da *Graphical User Interface* (GUI) (Interface Gráfica de Usuário) (GARBIN, 2010). Neste tipo de interface apareceu a capacidade de exibir gráficos. Com o intuito de manipulação destes gráficos de maneira adequada, surgiram dispositivos como o mouse e o teclado. Um tipo de interface gráfica que

ficou muito conhecido é o modelo WIMP (*Window, Icon, Menu, Pointing device*). Apesar de ainda pouco intuitiva a interação com a GUI representou um avanço considerável comparada com a CLI. No entanto, a CLI ainda é utilizada em operações que exigem maior rapidez, como a configuração de roteadores e servidores em sistemas nos quais não há preocupação com usabilidade, pois somente os especialistas irão operá-los (MEDEIROS, 2012).

Buscando uma maior integração dos computadores na vida cotidiana das pessoas e aumentar o nível de interação humano-computador com uma experiência mais realista e natural com o usuário, surgiu a *Natural User Interface* (NUI) (Interface Natural do Usuário), onde podem ser dispensados dispositivos artificiais (físicos) na comunicação entre homem e computador (MARINATO; GOMES; OLIVEIRA, 2015) e passam a ser utilizada a interação do usuário com o conteúdo a partir da captação de movimentos, gestos e voz (RAUTERBERG, 1999 *apud* PEDROSA; NOTARGIACOMO; LOPES, 2015, p. 623).

Uma NUI faz uso das habilidades não computacionais já existentes nos usuários, seja no público em geral ou em um público específico (GARBIN, 2010). Tais habilidades não teriam sido aprendidas a princípio com o propósito de sua utilização na comunicação com uma máquina, mas sim, sendo inatas ou adquiridas por meio da prática e vivência de comunicação, tanto verbal quanto não verbal, na interação com outros seres humanos e com o meio ambiente a sua volta. Com o avanço da tecnologia, tais habilidades não computacionais puderam ser aproveitadas na comunicação com equipamentos computadorizados, utilizando ações como tocar, gesticular ou falar, através de interfaces projetadas baseando-se no modo de interação com o mundo real (FILHO; VIEIRA, 2012).

As interfaces naturais possibilitam aos usuários atingirem um alto nível de interação e satisfação mais rapidamente do que outras interfaces de interação. Isso se evidencia pela menor inclinação da curva de aprendizagem já que são utilizadas habilidades já presentes nos usuários. O uso deste leque de habilidades já existentes em cada pessoa elimina boa parte da necessidade de aprendizado por parte delas para a interação e realização dos seus desejos junto ao produto tecnológico (GARBIN, 2010).

Para Wigdor e Wixon (2011), o termo “natural”, que define esse tipo de interface, significa o modo como o usuário interage e também o que ele sente ao utilizar o produto. Ainda segundo os autores, a importância da NUI está não só na mudança de paradigma de comunicação com os computadores, mas no leque de possibilidades de uso que isso estabelece.

Buxton (2010 *apud* GARBIN, 2010, p. 52) aponta a capacidade de uma NUI de permitir ao usuário a interação direta com o conteúdo presente em um sistema. Este contato direto com o conteúdo, no entanto, não requer que necessariamente sejam eliminados todos os dispositivos físicos artificiais ou gráficos utilizados na interação, como botões e caixas de mensagens. Mas sim que eles sejam secundários no processo. O uso conjunto de interfaces é comum, e isto se explica pela maior especialização de uma ou outra interface em determinadas tarefas (GARBIN, 2010).

Características como a facilidade de aprendizado e de uso, a possibilidade de rápida adaptação dos novos usuários e a interação direta com o conteúdo, presentes em uma interface natural se mostram como um grande atrativo para a inclusão de usuários com dificuldades de utilizar outros tipos de interfaces, como as gráficas, atraindo novos consumidores para o mercado ou nova motivação para aqueles que estavam insatisfeitos com as interfaces disponíveis até então (GARBIN, 2010).

2.8.2 INTERFACES NATURAIS BASEADAS EM VISÃO COMPUTACIONAL APLICADAS A JOGOS DIGITAIS

A crescente preocupação da indústria de jogos eletrônicos com o modo como os jogadores interagem com os *games* fica evidenciada no surgimento recente de vários consoles de videogame comerciais com dispositivos de interface natural que possibilitam uma interação natural entre o jogador e o console. Esta é uma tendência surgida há algum tempo, mas vem alcançando maior aceitação e popularidade devido às pesquisas e desenvolvimento de novas tecnologias, resultando em componentes melhores e mais acessíveis (GARBIN, 2010). Tais

equipamentos possibilitam que o usuário interaja com o console a partir de movimentos corporais, gestos ou voz (PILLON, 2015).

Esta tendência pela incorporação de interfaces mais imersivas, interativas e intuitivas tem o intuito de atrair pessoas que não estão habituadas com os controles tradicionais, ampliando a fatia de mercado atingida pelas fabricantes destes aparelhos (GARBIN, 2010). Outra explicação desta evolução é o fato de que oferecer aos jogadores uma experiência interativa atraente é exatamente um dos objetivos de um jogo eletrônico (TSANG, *et al.* 2003 *apud* KIRNER; TORI; SISCOUTO, 2006, p. 200).

O uso da tecnologia de interação natural vem possibilitando uma melhor interação entre usuários e máquinas e visa dar aos computadores percepções sensoriais semelhantes às humanas. Esta tecnologia trabalha com hardwares e softwares baseados em ações humanas, como movimentar (com o reconhecimento de gestos), olhar (com a identificação da direção do olhar do usuário), falar (com o reconhecimento de voz) e tocar (com telas sensíveis ao toque) (MEDEIROS, 2012). A ampliação dos sentidos dos computadores, neste caso consoles de videogame, permite que os sentidos dos jogadores sejam mais explorados, resultando em uma maior imersão e aproximação entre o jogador e o conteúdo (GARBIN, 2010).

Entre os equipamentos que vem sendo utilizados recentemente estão os sensores inerciais, os sensores de pressão e os sensores de câmeras (VAN DIEST, 2013 *apud* PILLON, 2015, p. 57).

Os sensores inerciais, como giroscópios e acelerômetros, são utilizados para detectar a inclinação, rotação e aceleração do usuário. Um exemplo é o Wii Remote (ilustrado na Figura 19), controle principal do console Wii, lançado pela Nintendo[1] em 2006, que possui detecção através de três acelerômetros e um giroscópio (PILLON, 2015). O controle também pode ser utilizado no console Wii U, lançado pela Nintendo em 2012. A Figura 19 ilustra o Wii Remote (à direita) juntamente com um de seus acessórios, o Nunchuk (à esquerda).

Os sensores de pressão utilizam aparelhos que avaliam a pressão sobre uma superfície. Um exemplo é o Wii Balance Board do Nintendo Wii, que a partir de sensores de pressão na parte inferior, calculam o centro de pressão dos jogadores.

[1] Nintendo Company, Limited é uma empresa japonesa fabricante de videogames.

Outro exemplo é a plataforma de dança “Dance Dance Revolution” (DDR), lançado em 1998 para arcades pela Konami, que reconhece as posições pressionadas pelos pés dos jogadores (PILLON, 2015).

Os sensores de câmeras são equipamentos capazes de realizar um rastreamento do ambiente real, através de técnicas de visão computacional aplicadas no tratamento das imagens obtidas por suas câmeras, para a identificação da posição da mão, da cabeça, dos olhos e outros pontos do jogador a fim de construir um modelo deste usuário, e também para a identificação de objetos ou outros elementos do ambiente real, variando de acordo com a aplicação (GARBIN, 2010). Deste modo, o sucesso de uma interface natural baseada em sensores de câmeras, ou seja, baseada em técnicas de visão computacional, é dependente em boa parte do correto funcionamento de tais técnicas, visto que o insucesso das mesmas em rastrear o objeto de interesse presente no ambiente captado pela câmera resulta no insucesso da interface natural, e por fim, no insucesso da interação entre o usuário e o computador. A visão computacional, portanto, torna possível a captura dos movimentos dos jogadores, sem a necessidade de os jogadores terem contato físico com o equipamento. Como exemplos de sensores de câmeras, pode-se citar o EyeToy, lançado em 2003 para o Playstation 2 (conforme Figura 20.a), o Eye, seu sucessor, lançado em 2007 no Playstation 3 (conforme Figura 20.b) e em 2013 no Playstation 4 (conforme Figura 20.c) e o Microsoft Kinect, lançado em 2010 em sua primeira versão para o console Xbox 360 (conforme Figura 21.a) e em sua segunda versão, lançada em 2013, para o console Xbox One (conforme Figura 21.b) (PILLON, 2015).

Figura 19 – Wii Remote e Nunchuk

Fonte: www.wikipedia.org (2016)[2]

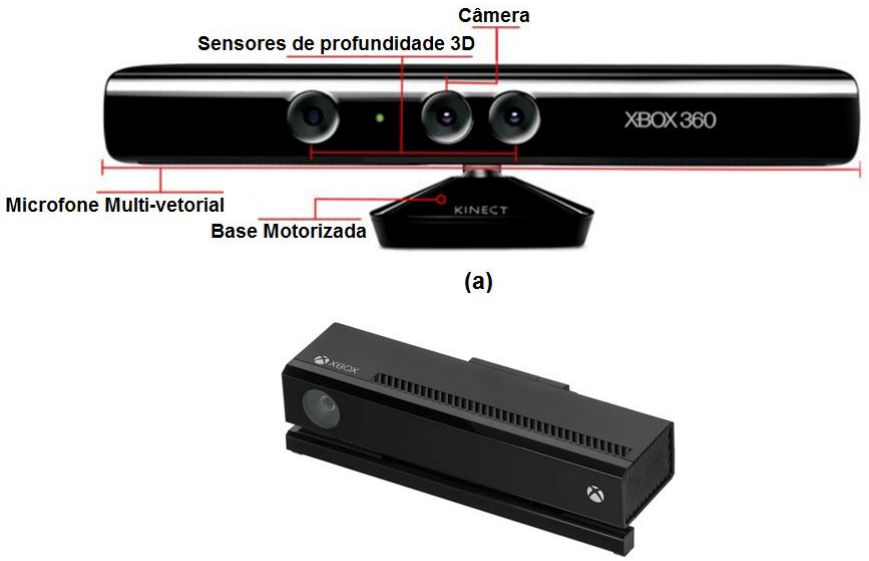
[2] WII. In: Wikipédia: a enciclopédia livre. Disponível em: <https://en.wikipedia.org/wiki/Wii#/media/File:Wiimote-in-Hands.jpg> Acesso em: 12 maio. 2016.

Figura 20 – Sensores de câmeras do Playstation. (a) EyeToy (PS2), (b) Eye (PS3), (c) Eye (PS4).

(a) (b) (c)

Fonte: (a) www.wikipedia.org (2016)[3], (b) www.wikipedia.org (2016)[4], (c) www.idigames.com (2016)[5]

Figura 21 – Sensores de câmeras do Xbox. (a) Kinect (Xbox 360) , (b) Kinect (Xbox One).



(a)

(b)

Fonte: (a) www.canaltech.com.br (2016)[6], (b) www.wikipedia.org (2016)[7]

[3] PLAYSTATION 2. In: Wikipédia: a enciclopédia livre. Disponível em: <https://pt.wikipedia.org/wiki/PlayStation_2#/media/File:PS2-Eyetoy.jpg> Acesso em: 16 maio. 2016.
[4] PLAYSTATION Eye. In: Wikipédia: a enciclopédia livre. Disponível em: <https://en.wikipedia.org/wiki/PlayStation_Eye#/media/File:PlayStation-Eye.png> Acesso em: 16 maio 2016.
[5] IDIGAMES. Câmera PS Eye: O melhor acessório para o PS4. Disponível em: <http://www.ldigames.com/blog/index.php/2014/02/27/camera-ps-eye-o-melhor-acessorio-para-o-ps4?blog=1> Acesso em: 19 maio. 2016.
[6] CANALTECH. Como funciona o Kinect. Disponível em: <http://canaltech.com.br/o-que-e/kinect/Como-funciona-o-Kinect/> Acesso em: 17 maio. 2016.
[7] XBOX One. In: Wikipédia: a enciclopédia livre. Disponível em: <https://en.wikipedia.org/wiki/Xbox_One#/media/File:Xbox-One-Kinect.jpg> Acesso em: 18 maio. 2016.

O avanço tecnológico na área de Visão Computacional e o aumento do poder de processamento dos computadores, associados à miniaturização e à popularização das câmeras, tornou o reconhecimento e rastreamento ópticos mais viáveis devido à disponibilidade e baixo custo do hardware de aquisição de vídeo em tempo real (GARBIN, 2010). As pesquisas relacionadas ao uso da visão computacional na interação com computadores são em grande parte impulsionadas pela inovação e facilidade que ela representa. Empresas como a Microsoft[8], fazem uso da visão computacional ou de tecnologias anteriores junto à visão computacional para a melhoria da interação dos jogadores com os videogames. Esta possibilidade de interação é responsável por atrair grande parcela do público e criar um mercado que atinge variadas faixas etárias, devido a ser algo diferenciado (ROCHA; SOUZA, 2014).

No processo de compreensão do comportamento humano baseado em visão computacional, a imagem é processada com o objetivo de extrair o seu significado e, dependendo do resultado, a interface em conjunto com o sistema, realiza o desejo do usuário. A interpretação que o sistema faz do comportamento humano captado pela interface passa pela avaliação de dados de posicionamento, controle, movimento e expressão de partes do corpo humano, para assim compreender a vontade dos usuários (LIU, 2010 *apud* FILHO; VIEIRA, 2012, p. 5). Como exemplo, pode-se citar a utilização de técnicas como o Camshift (*Continuously Adaptive Mean Shif*), um algoritmo de visão computacional que possibilita o rastreamento de cores em uma sequência de vídeo, podendo ser utilizado no controle de *games* e gráficos 3D (BRADSKI, 1998) (visto em detalhes na seção 4.2.1).

[8] Microsoft Corporation é uma empresa multinacional que desenvolve e vende softwares de computador, computadores e outros produtos e serviços.

3 PROJETO DO JOGO (*GAME DESIGN DOCUMENT*)

Visando a aplicação de visão computacional para interagir com um jogo de forma natural, criou-se o projeto de *game* apresentado nesta sessão. O objetivo deste espaço do trabalho, portanto, é documentar e informar aspectos conceituais e artísticos do jogo digital intitulado “OpenCV Breakout”. Este tipo de documento é geralmente conhecido como o documento de projeto de jogo (do inglês, *Game Design Document* – GDD).

3.1 CONCEITO GERAL

O “OpenCV Breakout” é um jogo 2D monojogador no estilo arcade no qual o jogador por meio da câmera de um computador (webcam) controla um bastão no cenário do *game*. O jogador tem por objetivo destruir blocos coloridos rebatendo uma bola com este bastão.

O jogo está ambientado no cenário real capturado pela webcam e os blocos coloridos são os invasores deste cenário, que o jogador deve eliminar.

3.2 PÚBLICO ALVO

O jogo é focado em um público jovem de 14 a 35 anos, uma fatia dos jogadores que é ávida por jogos que fazem uso de tecnologias inovadoras ou atuais. Porém, pelo fato de a essência da interação do usuário com o jogo ser baseada em uma interface natural, se espera que outras faixas de público sejam atraídas. A facilidade de aprendizado e de uso, a possibilidade de rápida adaptação de novos usuários e a interação direta com o conteúdo por meio de movimentos do corpo ou objetos do ambiente real são atrativos que este tipo de interface traz ao jogo.

3.3 DIFERENCIAIS DO JOGO

O jogo tem um grande diferencial em seu método de controle pelo jogador, que ocorre por meio de um algoritmo de visão computacional que através da imagem capturada pela webcam, consegue rastrear os movimentos de elementos do mundo real onde está o jogador. A partir deste rastreamento, o jogo consegue transferir os movimentos do objeto do ambiente real para o bastão controlado pelo jogador dentro do *game*.

3.4 CARACTERÍSTICAS DO JOGO

Cada partida do *game* pode ser jogada por apenas um usuário (monojogador). O seu gênero é o arcade. A câmera do ambiente virtual é ortográfica fixa. O idioma é o português. A plataforma em que roda é o Windows. Os gráficos são em duas dimensões (2D). Os controles são parte no teclado e parte via interface natural baseada em técnicas de visão computacional. Os comandos de controle realizados pelo teclado objetivam basicamente a mudança de telas no jogo, ficando o rastreamento realizado por visão computacional e a interface natural, responsáveis pelo controle do ambiente de jogo, onde é controlado o bastão.

3.5 REFERÊNCIAS DE JOGOS

O jogo “OpenCV Breakout” tem como base dois tipos de referências: o jogo “Breakout” e o jogo “Arkanoid”. Estes jogos são respectivamente apresentados nas seções 3.5.1 e 3.5.2.

3.5.1 O JOGO “BREAKOUT”

Este jogo tem grande influência do jogo “Breakout”, de 1976, que teve como idealizadores a dupla Nolan Bushnell e Steve Bristow. A ideia do jogo era seguir o

sucesso do fenômeno “Pong”, mas para apenas um jogador. O “Breakout” contou com o desenvolvimento de Steve Jobs (co-fundador da Apple[9] e criador de aparelhos como iPhone e iPad), que na época era designer da Atari[10].

Contando com a ajuda de Steve Wozniak, seu parceiro na criação da Apple, Steve Jobs criou o jogo “Breakout” em apenas quatro dias. Ele foi lançado nos fliperamas em 1976 depois de algumas modificações e posteriormente teve uma versão destinada ao famoso console Atari 2600. Anos mais tarde, “Breakout” inspirou inúmeros clones, entre eles “Arkanoid” e “Alleyway” (TECHTUDO, 2016).

O jogo “Breakout”, de 1976, está ilustrado na Figura 22.

Figura 22 – Jogo “Breakout” (1976).

Fonte: www.techtudo.com.br (2016)[11]

3.5.2 O JOGO “ARKANOID”

Outra grande influência é o jogo eletrônico “Arkanoid” desenvolvido para arcade pela Taito em 1986, baseado no *game* “Breakout”, citado na seção 3.5.1.

Na história do *game*, “Arkanoid” é a nave mãe da qual a nave Vaus do jogador escapa.

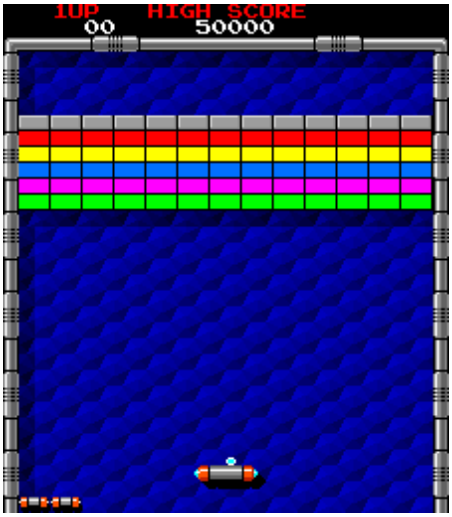
---

[9] Apple Inc. é uma empresa multinacional norte-americana que tem o objetivo de projetar e comercializar produtos eletrônicos de consumo, software de computador e computadores pessoais.
[10] Atari, Inc. é uma empresa de produtos eletrônicos e uma das principais responsáveis pela popularização dos videogames.
[11] TECHTUDO. Breakout, clássico do Atari, faz 37 anos com jogo 'escondido' no Google. Disponível em: <http://www.techtudo.com.br/noticias/noticia/2013/05/breakout-classico-da-atari-faz-37-anos-com-jogo-escondido-no-google.html> Acesso em: 12 maio. 2016.

O jogo é basicamente formado por uma bola, uma raquete e blocos coloridos. O jogador utiliza a raquete para tentar rebater a bola que se movimentava pela tela, movimentando-se para os lados. O objetivo é marcar pontos conforme a bola bate nos blocos (ARKANOID, 2016).

O jogo “Arkanoid”, de 1986, está ilustrado na Figura 23.

Figura 23 – Jogo “Arkanoid” (1986).



Fonte: www.wikipedia.org (2016)[12]

3.6 HISTÓRIA DO JOGO

Na história do *game* o ano é de 2022 e a raça humana sofre com a invasão do planeta por uma raça alienígena chamada “Hungerblockers”. Os invasores, porém não utilizaram ainda sua arma principal. A estratégia desta raça maligna para dominar um planeta é primeiramente posicionar blocos coloridos, os “Colorblocks”, na maior quantidade de locais possíveis, para então depois utilizar sua arma principal: o disparo de raios mortais a todos os seres humanos através destes blocos coloridos.

[12] ARKANOID. In: Wikipédia: a enciclopédia livre. Disponível em: <https://en.wikipedia.org/wiki/Arkanoid> Acesso em: 19 abr. 2016.

Só é conhecida uma única arma capaz de eliminar os “Colorblocks”, as “Colorballs”, bolas coloridas que viajantes intergalácticos humanos trouxeram do planeta dos alienígenas após uma expedição a muito tempo no passado.

Infelizmente os seres humanos não podem ter um contato próximo com as “Colorballs”, de modo que, para direcioná-las no ataque aos “Colorblocks”, devem utilizar um bastão azul especialmente fabricado pela Agência Espacial da Liga das Nações (AELN).

3.7 OBJETIVO E CONDIÇÕES DE VITÓRIA

O objetivo do jogo é eliminar todos os blocos coloridos (“Colorblocks”) do ambiente, a fim de ajudar a raça humana a evitar a principal arma alienígena: o disparo de raios mortais a todos os seres humanos através destes blocos coloridos.

3.8 INTERFACE

A interface entre o jogo e o usuário será composta por três elementos:

- ***Head-Up Display* (HUD):** responsável por apresentar visualmente informações ao jogador enquanto o jogo está em andamento, de forma que ele não precise desviar a atenção da sua partida (FAGERHOLT; LORENTZON, 2009).

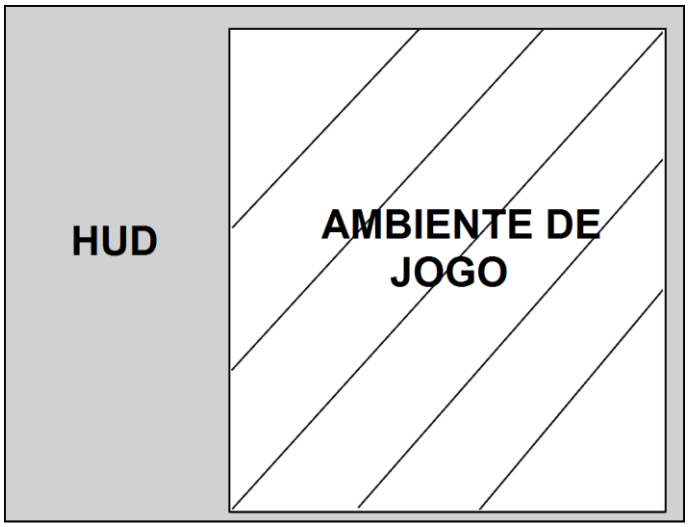
- **Ambiente de Jogo:** será o ambiente onde o jogador controlará o bastão azul a fim de rebater a bolinha (“Colorball”) para destruir os blocos coloridos inimigos (“Colorblocks”). Esse espaço terá como plano de fundo o vídeo captado em tempo real pela webcam.

- **Controles:** O controle do ambiente de jogo será realizado por meio de interface natural baseada em técnicas de visão computacional. Através destas técnicas o jogo consegue rastrear objetos do ambiente real do jogador

baseando-se na sua cor. Esse rastreamento possibilita a transferência de movimentos deste objeto para dentro do jogo. Inicialmente o jogador seleciona através do mouse uma região retangular da imagem captada pela webcam onde esteja o objeto de interesse para rastreio. Em seguida se inicia o rastreamento e a movimentação do bastão azul, o qual só se movimenta na horizontal. Outros controles estão presentes para ativação de telas: utiliza-se a tecla “I” do teclado ou o clique do botão direito do mouse para a ativação da tela de seleção de cor e a tecla “M” do teclado para a ativação da tela do *menu* inicial. Para iniciar o movimento da bolinha a ser rebatida pelo jogador é utilizada a barra de espaço do teclado.

A subdivisão da tela entre HUD e ambiente de jogo está ilustrada na Figura 24.

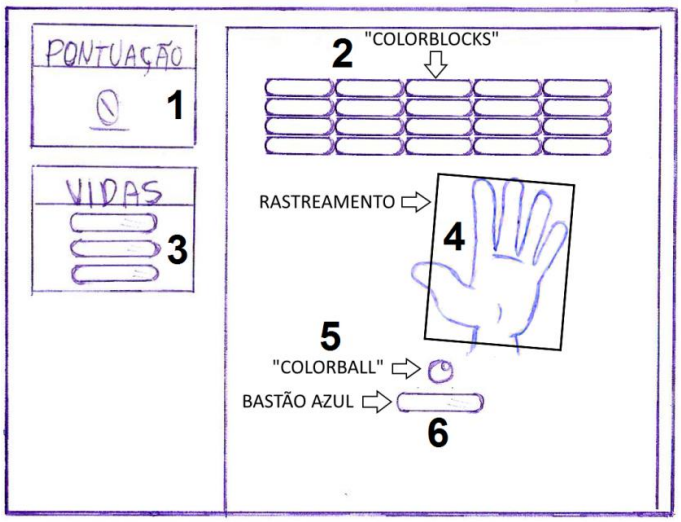
Figura 24 – Subdivisão da tela entre HUD e ambiente de jogo.



Fonte: Próprio autor

O esboço apresentado na Figura 25 permite a visão de como os itens do jogo estão posicionados na tela.

Figura 25 – Esboço do jogo “OpenCV Breakout”.



Fonte: Próprio autor

Os seguintes itens podem ser observados de acordo com a numeração apresentada:

1. Indicação da pontuação atual do jogador.
2. Localização dos inimigos (“Colorblocks”), agrupados na parte superior da tela.
3. Indicação da quantidade de vidas que o jogador ainda tem, com cada vida sendo representada por um bastão azul, do mesmo tipo do bastão controlado pelo jogador.
4. Localização do objeto rastreado pela sua cor.
5. Localização da bolinha (“Colorball”) a ser rebatida pelo bastão.
6. Localização do bastão azul controlado pelo jogador.

3.9 FLUXO DO JOGO

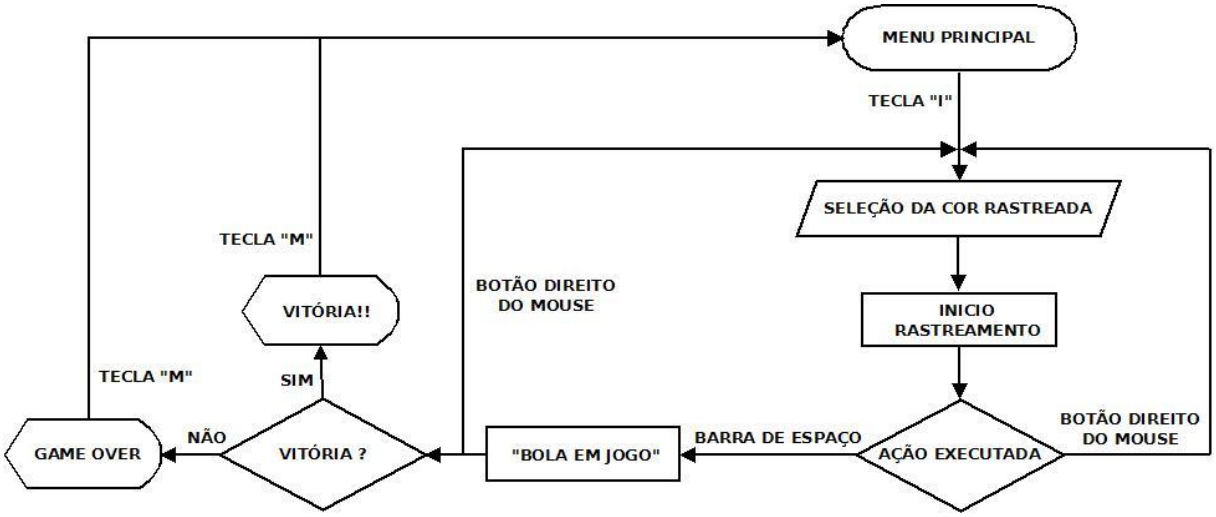
Primeiramente o jogo apresenta o *menu* inicial. Quando pressionada a tecla “I” do teclado é apresentada a tela de seleção do objeto a ser rastreado e utilizado para controle do bastão azul. Realizado esse processo, o jogador pode dar sequência ao jogo pressionando a barra de espaço do teclado, ou a qualquer momento clicar com o botão direito do mouse para voltar à tela de seleção da cor.

Ao pressionar a barra de espaço a bolinha é então impulsionada, dando inicio a disputa em si (“a bola está em jogo”), onde o jogador deve rebater a bolinha com o bastão azul controlado por ele a fim de eliminar os inimigos presentes na parte superior da tela.

Ao acabarem as vidas do jogador, é exibida a tela de derrota (*game over*), e ao eliminar todos os inimigos é exibida a tela de vitória. Ao pressionar a tecla “M” do teclado, estando em qualquer uma destas duas telas, retorna-se a tela do *menu* inicial.

O fluxo completo do jogo está ilustrado na Figura 26.

Figura 26 – Fluxo do jogo “OpenCV Breakout”.

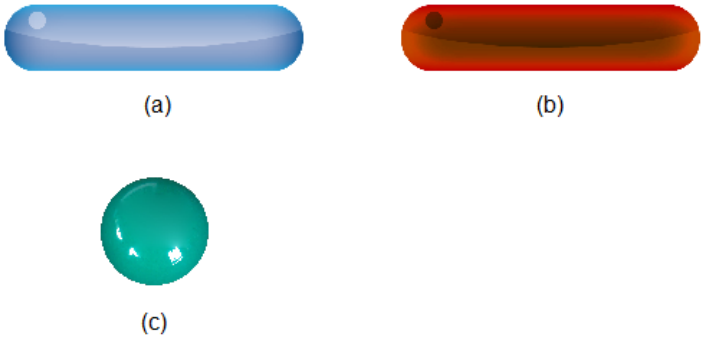


Fonte: Próprio autor

3.10 OBJETOS DO JOGO

O bastão azul controlado pelo jogador (Figura 27.a), os blocos coloridos que devem ser destruídos (“Colorblocks”) (Figura 27.b) e a bolinha a ser rebatida (“Colorball”) (Figura 27.c) são objetos presentes no ambiente de jogo.

Figura 27 – Objetos do ambiente de jogo. (a) Bastão azul, (b) Blocos coloridos (“Colorblocks”), (c) Bolinha a ser rebatida (“Colorball”).



Fonte: Próprio autor

4 IMPLEMENTAÇÃO DO JOGO

Este capítulo trata do desenvolvimento do jogo “OpenCV Breakout” proposto no capítulo 3. A seção 4.1 cita a arquitetura básica necessária para o funcionamento do *game*. A seção 4.2 aborda as técnicas de visão computacional utilizadas. A seção 4.3 apresenta as ferramentas utilizadas, bem como a integração entre elas. Nas seções de 4.4.1 até 4.4.7 são explicados os principais trechos do código fonte do script implementado em C# e os resultados obtidos com cada um destes trechos de código. Finalmente na seção 4.5 são apresentados e discutidos os resultados obtidos com o jogo em algumas situações específicas.

4.1 ARQUITETURA (COMPONENTES NECESSÁRIOS)

Baseando-se na revisão da literatura apresentada no capítulo 2, para o funcionamento do *game* desenvolvido a partir do projeto descrito no capítulo 3, se fazem necessários alguns elementos básicos:

- **Fonte de Radiação:** será a iluminação do ambiente onde estiver o jogador. Sobre este ponto é importante considerar o momento de escolha da iluminação para o funcionamento do sistema, permitindo a melhor distinção possível das cores presentes na imagem captada pela câmera.
- **Câmera com Sensor:** será a webcam do computador onde o jogador está executando o *game*.
- **Display:** será o monitor do computador onde será mostrado ao jogador tanto o resultado do rastreamento juntamente com os outros elementos do *game*, com as consequências dos movimentos rastreados com as técnicas de visão computacional na movimentação do bastão controlado pelo jogador no ambiente virtual do jogo.
- **Computador:** como citado na seção 2.3, em um sistema deste tipo, o computador centraliza todo o processo, comandando os demais componentes da arquitetura, desde a ativação da aquisição das imagens, o tratamento do software que processa as imagens, o armazenamento dos dados e a

visualização dos resultados em dispositivos como, neste caso, o monitor. Somam-se ainda ao computador os demais processamentos e responsabilidades relativas ao funcionamento dos outros processos do *game* e do sistema operacional.

4.2 TÉCNICAS DE VISÃO COMPUTACIONAL UTILIZADAS

Após um período de pesquisa a respeito de algoritmos de rastreamento de objetos, optou-se, na implementação deste protótipo, pelo uso de um algoritmo de visão computacional chamado Camshift. Ele é considerado de implementação simples e é computacionalmente eficiente no rastreamento de objetos coloridos (BRADSKI, 1998). A seção 4.2.1 traz mais detalhes desta técnica.

4.2.1 CAMSHIFT

Criado originalmente para o rastreamento de faces, o Camshift (*Continuously Adaptive Mean Shift*) é um algoritmo de visão computacional que possibilita o rastreamento baseado em cores em uma sequência de vídeo. Ele foi um primeiro passo de um grupo de pesquisadores no intuito de dar aos computadores habilidades sensoriais semelhantes às dos humanos. O objetivo maior do projeto era contribuir para a construção de melhores interfaces de comunicação entre homem e computador (BRADSKI, 1998).

Podendo ser utilizada no controle de *games* e gráficos 3D, com a utilização de rastreamento de face e transferência dos seus movimentos para os movimentos de um personagem, por exemplo, o Camshift pode tornar o uso de computadores mais natural e *games* e simulações mais divertidas (BRADSKI, 1998). Estes fatos, somados à sua simplicidade e eficiência computacional, justificam a sua escolha como o método de rastreio no desenvolvimento do jogo “OpenCV Breakout”.

O Camshift é baseado no Mean Shift, um outro algoritmo que, atuando em uma imagem de distribuição de probabilidade, a partir de uma região de interesse inicial, calcula a média ponderada dos pontos dentro desta região, encontrando

deste modo um novo centro para a região de interesse, para onde ela é então deslocada (RAMOS, 2012). A técnica do Mean Shift, porém, possui a limitação de trabalhar apenas com distribuições estáticas de probabilidade, não sendo possível usá-la para distribuições dinâmicas, como uma sequência de vídeo. Esta limitação foi resolvida pelo Camshift com a consideração deste dinamismo na reavaliação da distribuição de probabilidade no decorrer dos frames que compõe um vídeo (BRADSKI, 1998; FRANÇOIS, 2004; RAMOS, 2012). Outra limitação do Mean Shift é o tamanho fixo da janela de busca, o que pode causar problemas quando o elemento rastreado se aproxima ou se afasta da câmera, a qual também foi resolvida no Camshift com o dinamismo do tamanho desta área de busca. O Mean Shift, porém, foi incorporado dentro do Camshift como parte do seu processamento (BRADSKI, 1998; RAMOS, 2012).

O funcionamento do algoritmo Camshift é caracterizado pela sua simplicidade. Esta característica possibilita que ele seja computacionalmente eficiente, porém impõe algumas limitações (BRADSKI, 1998). Como o algoritmo é baseado somente em distribuição de probabilidade de cor, ele acaba tendo alguma sensibilidade na ocorrência de variações indesejadas nas cores da imagem, como em situações em que aja uma fonte de luz colorida, com pouca iluminação ou com muita iluminação. O algoritmo foi projetado para diminuir estes efeitos de variação de iluminação considerando nas imagens captadas apenas o canal do matiz (*hue*) no espaço de cor HSV (BRADSKI, 1998).

Outras limitações também receberam tratamento na criação do Camshift. Para o caso de variações da distância entre o objeto rastreado e a câmera, o algoritmo faz a adaptação do tamanho da janela de busca para um tamanho adequado. A influência de ruídos nas imagens captadas é minimizada pelo fato do rastreamento se basear nas cores dos objetos e pela baixa probabilidade de os ruídos serem da exata cor que está sendo rastreada. Para o caso de existirem outros elementos na imagem com a mesma cor do objeto rastreado, o algoritmo acaba por ignorá-los caso não estejam perto do objeto rastreado e tende a manter a janela de busca sobre o objeto que tenha a distribuição de probabilidade de cor dominante. E para situações em que ocorra oclusão, onde um objeto fica escondido atrás de outro objeto, o funcionamento do algoritmo permite a manutenção do rastreamento para casos em que a oclusão não seja de 100% (BRADSKI, 1998).

O Camshift é composto pelas seguintes etapas:

1 – Realiza-se o processo de calibragem do dispositivo, no qual se define a posição inicial da janela de busca por meio da seleção manual de uma região retangular sobre a imagem do vídeo capturado pela câmera (GADELHA; SANTOS, 2010). Esta região é considerada como a região de interesse (*Region Of Interest* - ROI) a partir da qual atuará o algoritmo de rastreamento, ou em outras palavras: a região onde está a cor (ou objeto) escolhida para ser rastreada.

2 - Constrói-se o histograma modelo da região selecionada na etapa anterior contendo a cor que será rastreada (BRADSKI, 1998; RAMOS, 2012).

3 – A partir do frame atual do vídeo capturado pela câmera, originalmente no sistema de cores RGB, obtêm-se a imagem no sistema de cores HSV, do qual somente a componente matiz (cor) será então considerada. Esta separação do matiz dos outros canais (saturação e brilho) no espaço de cor HSV reduz possíveis problemas de intensidade luminosa, contribuindo para melhores resultados (FRANÇOIS, 2004; GADELHA, SANTOS, 2010; BRADSKI, 1998).

4 - Calcula-se a distribuição de probabilidade da cor selecionada no frame atual do vídeo através do método de retroprojeção de histograma (*histogram backprojection*). Este algoritmo faz a projeção do histograma modelo com o histograma do frame atual do vídeo (RAMOS, 2012). Obtém-se então uma imagem em que a intensidade dos *pixels* se dá de acordo com a probabilidade dos mesmos de pertencerem ao histograma modelo (que contem a cor rastreada) (RAMOS, 2012). E é em cima desta imagem, e não em cima dos quadros originais capturados pela câmera, que atuará o algoritmo Mean Shift na próxima etapa (BRADSKI, 1998).

5 - Executa-se o algoritmo Mean Shift, por meio de dois passos:

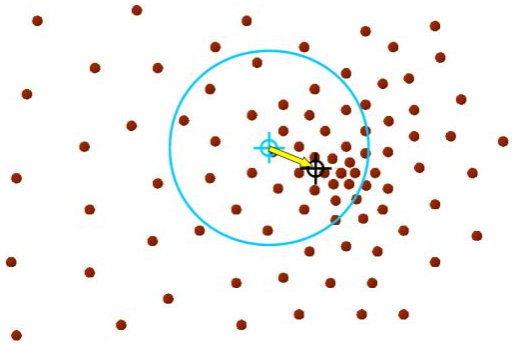
5.1 - Calcula-se o centróide (centro de massa) dentro da janela de busca (ROI), ou seja, a região com máxima densidade (pico de densidade), onde se verifica ser mais provável que os *pixels* contenham a cor do histograma modelo.

5.2 – Considera-se o centróide calculado no passo anterior como o centro da janela de busca.

Repetem-se os passos 5.1 e 5.2 do Mean Shift até que ocorra a convergência, ou seja, que o movimento da janela de busca não seja maior que um valor limite, ou até que se atinja o critério de término de iteração. Ao final desta etapa são armazenados a nova localização do centro da janela de busca e a sua nova área (BRADSKI, 1998; GONÇALVES; CORREIA, 2005 ; RAMOS, 2012).

A Figura 28 ilustra a técnica utilizada pelo algoritmo Mean Shift. Na Figura, a janela de busca está representada pelo círculo azul (maior) e o centroide calculado está representado pelo ponto na cor preta (para onde aponta a seta).

Figura 28 – Técnica utilizada pelo algoritmo Mean Shift.



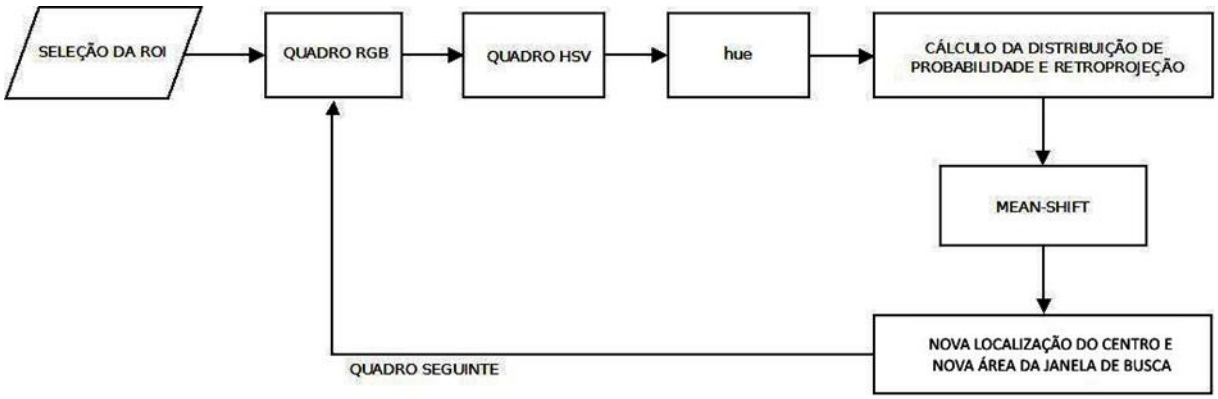
Fonte: Ukrainitz e Sarel (2004, apud RAMOS, 2012, p. 20)

6 – Para o próximo quadro do vídeo captado pela câmera, a janela de busca recebe a sua nova posição e área, de acordo com os dados armazenados na etapa 5 (BRADSKI, 1998; GONÇALVES; CORREIA, 2005; RAMOS, 2012).

7 – Volta-se ao passo 4.

A Figura 29 ilustra o funcionamento do algoritmo Camshift.

Figura 29 – Funcionamento do algoritmo Camshift.



Fonte: Próprio autor

4.3 FERRAMENTAS UTILIZADAS

Com base nas pesquisas realizadas na área de Visão Computacional e desenvolvimento de *games*, optou-se pela utilização das seguintes ferramentas no desenvolvimento do jogo proposto:

- OpenCV[13]
- OpenCVSharp[14]
- Linguagem de programação C#[15]
- Unity 3D[16]

As ferramentas escolhidas estão detalhadas nas próximas seções.

4.3.1 OPENCV E OPENCVSHARP

A OpenCV ou *Open Source Computer Vision* é uma biblioteca de visão computacional de código aberto escrita em C e C++. A biblioteca teve como

[13] OpenCV na versão 2.49, disponível em: http://www.opencv.org/downloads.html
[14] OpenCVSharp na versão 2.49, disponível em: https://github.com/shimat/opencvsharp/releases/tag/2.4.9.20141018
[15] Linguagem C# disponível juntamente com o Unity 3D em: https://unity3d.com/pt/get-unity
[16] Unity 3D utilizado na versão gratuita (5.3.3f1). Ultima versão do Unity 3D disponível em: https://unity3d.com/pt/get-unity

desenvolvedora inicial a Intel Corporation[17] e os seguintes objetivos segundo Bradski e Kaehler (2008):

- Avançar a pesquisa na área provendo um código aberto e otimizado contendo a infraestrutura básica necessária para visão computacional.
- Disseminar conhecimento de visão computacional através de uma infraestrutura básica comum entre desenvolvedores.
- Permitir o avanço de aplicações comerciais baseadas em visão computacional, através de um código portátil e otimizado para o melhor desempenho.

Passou-se então a buscar tais objetivos visando aumentar a acessibilidade da visão computacional a usuários e programadores, facilitando as implementações em áreas como Robótica e Interação Humano-Computador (IHC) em tempo real (MARENGONI; STRINGHINI, 2009). Ela conta com diversos tipos de ferramentas de interpretação de imagens, desde operações como remoção de ruídos, até operações mais complexas, como análise de movimentos, reconhecimento de padrões e reconstrução em 3D, somando mais de 500 funções disponíveis (MARENGONI; STRINGHINI, 2009).

Os principais componentes da estrutura da OpenCV são:

- **CV (*Computer Vision*):** possui os algoritmos de processamento básico de imagens e os algoritmos de visão computacional de alto nível (BRADSKI; KAEHLER, 2008).
- **MLL (*Machine Learning Library*):** apresenta funções de aprendizado de máquina (SIOLA, 2010) e de agrupamento, classificação e análise de dados (SIQUEIRA, 2014).
- **HighGUI:** apresenta funções para criação de interfaces gráficas com o usuário, assim como para captura e reprodução de vídeos e imagens (RÉZIO, 2008).

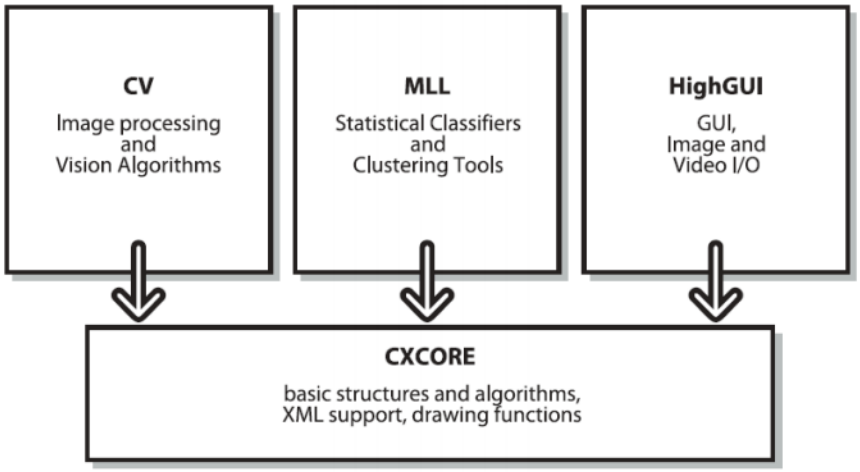
[17] Intel Corporation é uma empresa multinacional que fabrica circuitos integrados como microprocessadores (http://www.intel.com.br/content/www/br/pt/homepage.html).

- **CxCore:** permite o funcionamento em conjunto dos outros blocos, contendo as funções básicas e estruturas de dados necessárias para tal (SIOLA, 2010). Também contém funções de desenho e suporte a XML (SIQUEIRA, 2014).

A Figura 30 ilustra a estrutura básica da biblioteca OpenCV.

Já a OpenCVSharp é uma biblioteca multiplataforma, sendo uma wrapper da biblioteca OpenCV para o .NET Framework. Ela expõe as funcionalidades da OpenCV para que seja possível utilizá-la com programação na linguagem C#.

Figura 30 – Estrutura básica da biblioteca OpenCV.



Fonte: Bradski e Kaehler (2008)

4.3.2 LINGUAGEM DE PROGRAMAÇÃO C#

O C# é uma linguagem de programação desenvolvida expressamente para o .NET Framework da Microsoft (DEITEL et al., 2003), componente do Windows que permite a interoperabilidade entre o C# e várias outras linguagens (MICROSOFT, 2016) como o Visual Basic .NET, Visual C++ .NET e outras, tendo o programador a liberdade de escolher a linguagem pela qual tem preferência (DEITEL et al., 2003). É uma linguagem orientada a objetos e que permite a construção de uma variedade de aplicações seguras e robustas, compatíveis com o .NET (MICROSOFT, 2016).

A linguagem C# é simples e fácil de aprender e com a orientação a objetos suporta a utilização de conceitos como encapsulamento, herança e polimorfismo (MICROSOFT, 2016).

4.3.3 UNITY 3D

O Unity 3D é um motor de jogos (*game engine*) que auxilia no desenvolvimento de jogos em duas dimensões (2D) e três dimensões (3D) (SILVA; SILVA, 2011). Ele é um software proprietário, mas que possui também uma versão gratuita, cujo conjunto de ferramentas tem se mostrado suficientes para o desenvolvimento de um jogo completo (SILVA et al., 2014). Os jogos produzidos nesta versão gratuita da ferramenta podem ser compilados para PC, Mac ou embutidos na web. Com a aquisição de licenças específicas, pode-se compilar jogos para vários outros dispositivos como iPhone da Apple e o console Wii da Nintendo (PASSOS et al., 2009).

A *engine* suporta as linguagens de programação orientadas a objetos C# e Javascript na criação dos seus scripts. Os scripts são executados internamente através de uma versão modificada da biblioteca Mono (implementação do sistema .Net de código aberto).

4.3.4 INTEGRAÇÃO ENTRE OPENCV E UNITY 3D

Para o desenvolvimento do jogo “OpenCV Breakout”, estudou-se a possibilidade de integração da OpenCV com o Unity 3D, de modo que por meio de scripts pudesse ser implementado um jogo que fizesse uso de algum algoritmo de visão computacional que tivesse a capacidade de rastrear objetos a partir de imagens captadas por uma webcam.

Considerando-se o fato de o autor deste trabalho ter mais afinidade com a programação na linguagem C# e o fato de o Unity 3D possibilitar, como citado na seção 4.3.3, o desenvolvimento nesta linguagem, optou-se por uma interface do

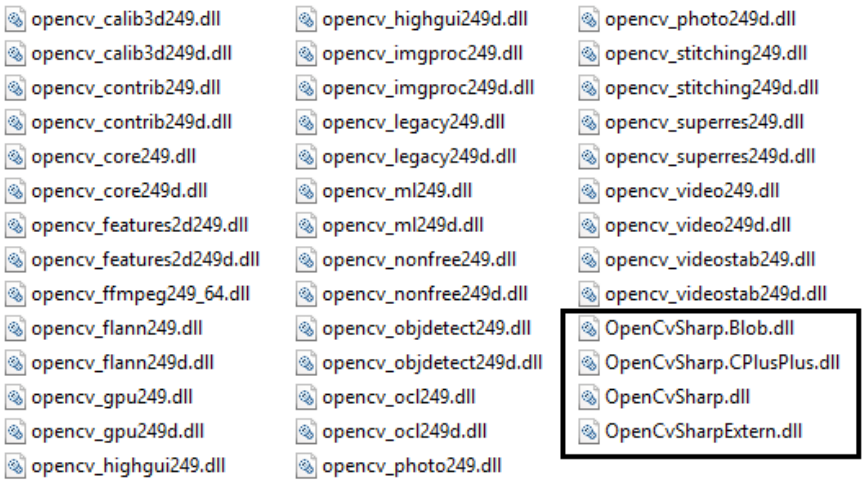
OpenCV que também possibilitasse a programação em C#. Foi escolhida então a biblioteca OpenCVSharp.

A realização da integração entre o Unity 3D e a OpenCVSharp, bem como a implementação do algoritmo Camshift, utilizado para o rastreamento baseado em cor, foram baseadas em um exemplo disponível no fórum de discussões do site do Unity 3D[18].

Para a integração entre o Unity 3D e a OpenCVSharp são primeiramente adicionadas as bibliotecas que compõe a OpenCV e a OpenCVSharp (as duas na mesma versão) no diretório do projeto do jogo dentro dos *Assets* do Unity 3D. Para a realização desta etapa levou-se em conta o tipo de Windows onde roda o jogo, se 32 ou 64 *bits*, pois tanto a OpenCV como a OpenCVSharp tem diferentes versões para serem usadas em uma ou outra situação.

As bibliotecas que foram adicionadas no Unity 3D para o projeto do jogo criado estão listadas na Figura 31.

Figura 31 – Bibliotecas da OpenCV e OpenCVSharp.

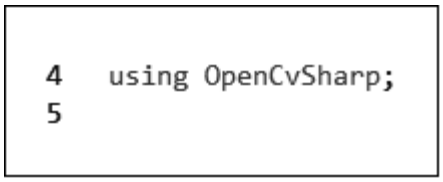


Fonte: Próprio autor

Os arquivos da OpenCVSharp estão em destaque na Figura 24, sendo o principal deles, o arquivo “OpenCvSharp.dll”, “inicializado” conforme script em C# exibido na Figura 32.

[18] REINA, G. A. OpenCVSharp for Unity, 2014. Disponível em: <http://forum.unity3d.com/threads/opencvsharp-for-unity.278033/> Acesso em: 17 fev. 2016.

Figura 32 – Inicialização da biblioteca OpenCVSharp.



```
4   using OpenCvSharp;
5
```

Fonte: Próprio autor

No desenvolvimento do jogo “OpenCV Breakout” não foi necessária a chamada direta de funções nativas da biblioteca OpenCV, visto que o objetivo da OpenCVSharp é exatamente encapsular as funções nativas da OpenCV para uso em C#. A adição dos arquivos com a extensão DLL (do inglês, *Dynamic Link Library*) da OpenCV no projeto se faz necessário pelo uso que a OpenCVSharp faz deles internamente no seu funcionamento.

Para a implementação do rastreamento baseado em cor feito pelo algoritmo Camshift, foram utilizados elementos do OpenCV que são úteis neste processo, como: funções de cálculo de histograma, cálculo de distribuição de probabilidade de cor e a própria função de Camshift, conforme pode ser observado na seção 4.4.

4.4 IMPLEMENTAÇÃO DO SCRIPT EM C#

Nesta seção é explicada a parte do código fonte implementado em C# no jogo “OpenCV Breakout” responsável pelo rastreamento de objetos realizado pelo algoritmo de visão computacional Camshift. Junto aos trechos do código fonte exibidos nas seções 4.4.1 até 4.4.7, são apresentados os resultados obtidos com cada etapa deste processo, o qual se baseia na ordem das etapas do Camshift detalhadas na seção 4.2.1. São também apresentados na seção 4.5 os resultados obtidos em algumas situações específicas, conforme se configuram alterações em algumas variáveis, como: a distância entre a câmera e o objeto rastreado, o tipo de iluminação do ambiente captado pela câmera, a presença ou não de objetos que possuem a mesma cor do objeto sendo rastreado, etc.

4.4.1 SELEÇÃO DA REGIÃO DE INTERESSE (ROI)

Inicialmente o usuário deve selecionar a região de interesse (*Region Of Interest* - ROI) na qual se baseará o processo de rastreamento a ser realizado posteriormente pelo algoritmo Camshift. Neste caso, ele deverá desenhar um retângulo sobre a imagem do quadro capturado pela webcam do ambiente real onde o jogador está. Com o clique do botão esquerdo do mouse se inicia a seleção da região de interesse pelo canto superior esquerdo, sendo guardada esta posição na variável "`_mouseDownPos`". Ao se soltar o clique do botão esquerdo é também guardada essa posição na variável "`_mouseLastPos`", que será a do canto inferior direito do retângulo formado. Deste modo está delimitada a ROI.

Como exemplo, tem-se a seleção feita na situação da Figura 33, onde o jogador seleciona o seu rosto, cujas cores serão a base para o rastreio realizado pelo algoritmo Camshift posteriormente.

Figura 33 – Seleção da face do jogador.

Fonte: Próprio autor

Na sequência são extraídas as posições do retângulo da ROI resultando no objeto "`_rectToTrack`" do tipo "`CvRect`", como pode ser visto na Figura 34.

Figura 34 – Extração das posições do retângulo selecionado.

```
313     _rectToTrack = CheckROIBounds(ConvertRect2CvRect
314         (MakePixelBox(_mouseDownPos, _mouseLastPos)));
315
```

Fonte: Próprio autor

4.4.2 EXTRAÇÃO DA REGIÃO DE INTERESSE (ROI)

Após ser obtido o objeto "`_rectToTrack`" no passo anterior, ele é usado para a obtenção, no formato "`CvMat`", da região selecionada pelo jogador na imagem capturada pela webcam através da função "`GetROI`", exibida na Figura 35.

Figura 35 – Obtenção no formato "CvMat" da região selecionada.

```
889
890 CvMat GetROI(CvMat _image, CvRect _rectToTrack)
891 {
892     CvMat img_roi;
893
894     _image.GetSubRect(out img_roi,_rectToTrack);
895
896     return (img_roi);
897 }
898
```

Fonte: Próprio autor

O parâmetro "`_image`" recebido pela função "`GetROI`" se trata do quadro atual captado pela webcam.

4.4.3 DEFINIÇÃO DO HISTOGRAMA MODELO DESTA REGIÃO DE INTERESSE (ROI)

O histograma modelo da região de interesse “`_histogramToTrack`” é então obtido a partir da chamada da função “`CalculateOneChannelHistogram`”, representada na Figura 36.

Figura 36 – Chamada da função “CalculateOneChannelHistogram”.

```
324
325         _histogramToTrack = CalculateOneChannelHistogram
326             (GetROI(videoSourceImage, _rectToTrack), 0, 179);
327
```

Fonte: Próprio autor

Na Figura 37 pode ser vista a estrutura básica da função “`CalculateOneChannelHistogram`”.

Figura 37 – Estrutura básica da função “CalculateOneChannelHistogram”.

```
546
547     CvHistogram CalculateOneChannelHistogram
548         (CvMat _image, int channelNum, float channelMax)
        {
                                ⋮
563         using (CvMat _imageHSV = ConvertToHSV(_image))
                                ⋮
574         _imageHSV.CvtPixToPlane(imgChannel, null, null, null);
                                ⋮
588         hist.Calc(Cv.GetImage(imgChannel), false, null);
                                ⋮
594         return hist;
595     }
```

Fonte: Próprio autor

Como pode ser notado a função “`CalculateOneChannelHistogram`” necessita de três parâmetros. Sendo eles:

- “CvMat _image” = a região obtida na função “GetROI” citada no passo anterior.
- ”int channelNum” = o canal utilizado na definição do histograma modelo. O número “0” é passado neste parâmetro para que posteriormente no interior da função seja a matiz da imagem HSV o canal utilizado na definição do histograma modelo.
- ”float channelMax” = O número máximo de níveis no canal escolhido.

No interior da função “CalculateOneChannelHistogram”, basicamente:

- A imagem da região de interesse (ROI) é convertida do modelo RGB para o modelo de cor HSV conforme a linha número 563 da Figura 37.
- É isolada a matiz (*hue*) do modelo HSV no objeto “_imgChannel”, conforme a linha número 574 da Figura 37.
- É calculado o histograma modelo da região de interesse conforme a linha número 588 da Figura 37.
- O histograma modelo resultante então é retornado pela função, conforme a linha número 594 da Figura 37.

As próximas etapas são executadas para cada quadro do vídeo captado pela webcam, até o encerramento do rastreamento.

4.4.4 CALCULO DA DISTRIBUIÇÃO DE PROBABILIDADE DE COR ATRAVÉS DA PROJEÇÃO REVERSA DE HISTOGRAMA

A distribuição de probabilidade de cor do histograma modelo com o histograma da imagem da webcam é obtida de acordo com o código exibido na Figura 38, a partir do quadro atual captado pela webcam (“_image”) e do histograma modelo da região de interesse.

Figura 38 – Calculo da distribuição de probabilidade de cor.

```
711
712        CvMat _backProject = CalculateBackProjection
713            (_image, _histogramToTrack);
714
```

Fonte: Próprio autor

4.4.5 MELHORIA DA IMAGEM DE DISTRIBUIÇÃO DE PROBABILIDADE DE COR

Depois de obtida a imagem de distribuição de probabilidade de cor, mostrou-se necessária a realização de uma melhoria no contraste nesta imagem. Deste modo, foram usadas duas operações morfológicas básicas que se enquadram como convoluções no domínio espacial: a Erosão e a Dilatação. O resultado é a diminuição de ruídos em favorecimento do agrupamento de regiões possuidoras de mesmas características na imagem, tornando-as mais contínuas (*blobs*). Esse ajuste foi necessário para um melhor funcionamento do posterior rastreamento dos picos de densidade (BRADSKI; KAEHLER, 2008). As chamadas destas duas operações, bem como a criação prévia dos elementos estruturantes (*kernels*) utilizados em cada uma delas, são demonstradas na Figura 39.

Figura 39 – Melhoria da imagem de distribuição de probabilidade de cor.

```
719
720    IplConvKernel elementErode = Cv.CreateStructuringElementEx
721        (10, 10, 5, 5, ElementShape.Rect, null);
722    IplConvKernel elementDilate = Cv.CreateStructuringElementEx
723        (4, 4, 2, 2, ElementShape.Rect, null);
724
725    Cv.Erode(_backProject, _backProject, elementErode, 1);
726    Cv.Dilate(_backProject, _backProject, elementDilate, 1);
727
```

Fonte: Próprio autor

Como citado na seção 2.4.2, nas operações de convolução, os elementos estruturantes (*kernels*), com o tamanho de poucos *pixels*, percorrem a imagem ponto a ponto, com a fixação de um novo valor ao *pixel* central deste *kernel.* No caso da Erosão, é fixado no *pixel* central o menor valor encontrado em toda a área de influência do *kernel*, tendo como consequência a tendência ao “emagrecimento” das regiões, na medida em que nas bordas dos elementos normalmente se encontram *pixels* mais escuros (fundo da imagem), o que acaba afetando os *pixels* da borda, torando alguns escuros também. Já para a Dilatação ocorre o contrário, são buscados e fixados valores de máxima intensidade (LAGANIÈRE, 2011).

Como resultado obtém-se a distribuição de probabilidade de cor representada na Figura 40, obtida após ter sido escolhida a face do jogador como região contendo a cor a ser rastreada.

Figura 40 – Distribuição de probabilidade da cor da face do jogador.

Fonte: Próprio autor

4.4.6 EXECUÇÃO DA FUNÇÃO CAMSHIFT

É então executada a função do OpenCvSharp correspondente à técnica Camshift. O nome desta função é “`Cv.CamShift`”, cuja chamada é feita conforme Figura 41, passando-se os seguintes parâmetros:

- “`CvArr prob_image`” = a distribuição de probabilidade considerada
- “`CvRect window`” = a região de interesse (ROI) considerada

- “CvTermCriteria criteria” = representa o número máximo de iterações do algoritmo e o limite de movimento para que se considere a convergência.
- “out CvConnectedComp comp” = referência que contem o retângulo “comp.Rect” que convergiu e que é usado como região de interesse na próxima chamada ao método “Cv.CamShift”.
- “out CvBox2D box” = referência que contem o retângulo rotacionado do qual a posição central é usada para a movimentação do objeto do ambiente virtual controlado pelo jogador.

A função “Cv.CamShift” está contida em uma condição para verificação de seu retorno no formato inteiro. Este retorno indica o número de iterações do Mean Shift interno.

Figura 41 – Chamada da função “Cv.CamShift”.

```
751
752     if (Cv.CamShift(_backProject, _rectToTrack, term_criteria,
753         out _connectComp, out _outBox) > 0)
```

Fonte: Próprio autor

Sendo o retorno da função “Cv.CamShift” maior que zero (0), considera-se que houve ao menos uma iteração do Mean Shift interno e o processo prossegue com o armazenamento da posição e tamanho da região de interesse, presentes no retorno por referência “_connectComp” da função “Cv.CamShift”, no objeto “_rectToTrack”, conforme Figura 42. Estas informações armazenadas serão posteriormente passadas na chamada seguinte ao método “Cv.CamShift” durante o processamento do próximo quadro do vídeo captado pela câmera.

Figura 42 – Armazenamento da posição e tamanho da região de interesse.

```
768
769     _rectToTrack = CheckROIBounds(_connectComp.Rect);
770
```

Fonte: Próprio autor

Já o retorno por referência "`_outBox`" da função "`Cv.CamShift`" é usado para a movimentação do bastão azul controlado pelo jogador na cena do jogo, após o seu armazenamento no objeto "`rotatedBoxToTrack`", conforme a Figura 43. Esta informação indica a posição que o jogo considera como sendo a posição do objeto (ou da cor) selecionado previamente pelo jogador.

Figura 43 – Armazenamento do retorno "_outBox".

```
781
782     rotatedBoxToTrack = _outBox;
783
```

Fonte: Próprio autor

Para possibilitar ao jogador acompanhar na tela o rastreamento feito pelo algoritmo Camshift, foi implementado um trecho de código para, a partir do objeto "`rotatedBoxToTrack`", exibir na tela do jogo um retângulo sobre o objeto rastreado durante o seu funcionamento, como pode ser observado na Figura 44, na qual é rastreada a face do jogador.

Figura 44 – Rastreamento realizado pelo algoritmo Camshift.

Fonte: Próprio autor

Já com o movimento da face do jogador para a esquerda da tela, obtém-se como resultado a imagem de distribuição de probabilidade que é demonstrada na Figura 45, onde pode-se observar que a região com cores mais claras agora se deslocou para a esquerda, pois é nesta nova posição em que agora há maior probabilidade de os *pixels* pertencerem ao histograma modelo, ou seja, que sejam da cor que está sendo rastreada.

Figura 45 – Imagem de distribuição de probabilidade após movimento do jogador para a esquerda.

Fonte: Próprio autor

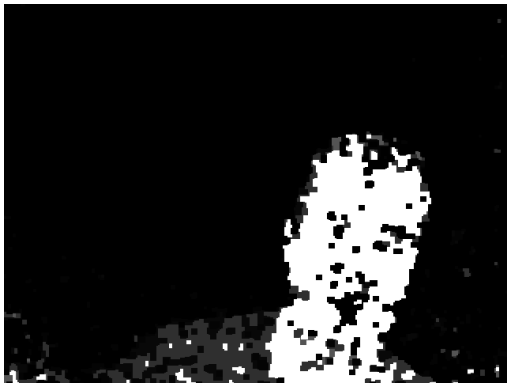
Com o novo processamento do algoritmo Camshift, o resultado é uma nova posição da face identificada pelo jogo, como demonstrado na Figura 46, como consequência do novo pico de densidade presente na imagem de distribuição de cor.

Figura 46 – Resultado do rastreamento realizado pelo algoritmo Camshift após movimento a esquerda.

Fonte: Próprio autor

Na movimentação da face do jogador para a direita obtém-se a imagem de distribuição de probabilidade exibido na Figura 47. E com a execução do algoritmo Camshift, é obtida uma nova identificação da posição da face do jogador, agora posicionada mais na direita da imagem, como pode ser visto na Figura 48.

Figura 47 – Imagem de distribuição de probabilidade após movimento do jogador para a direita.



Fonte: Próprio autor

Figura 48 – Resultado do rastreamento realizado pelo algoritmo Camshift após movimento a direita.

Fonte: Próprio autor

Pode-se notar, deste modo, o sucesso no rastreamento da face do jogador pelo jogo, como resultado do processamento realizado pelo algoritmo Camshift nas imagens captadas pela webcam.

4.4.7 MOVIMENTAÇÃO DO BASTÃO AZUL

A partir do sucesso no rastreamento do objeto rastreado pelo jogo através do algoritmo de visão computacional Camshift, se torna possível a movimentação do bastão azul controlado pelo jogador dentro do *game*, a partir do trecho de código implementado demostrado na Figura 49.

No trecho exibido se avalia a posição apontada pelo objeto "`rotatedBoxToTrack`" em relação ao bastão controlado pelo jogador, afim de realizar a movimentação horizontal do bastão. Se a posição central da região representada pelo objeto estiver mais à esquerda do bastão, o resultado é a movimentação do bastão para a esquerda. Do mesmo modo, se a posição central da região representada pelo objeto estiver mais à direita do bastão, o resultado é a sua movimentação para a direita. Considera-se "`velPlayer`" como a velocidade pré-fixada em que o bastão pode se mover. O sucesso desta operação pode ser notado nas Figuras 50.1 e 50.2 onde o jogador já liberou a bolinha e tenta rebatê-la movimentando-se respectivamente para a esquerda e depois para a direita da tela, a fim de destruir os blocos vermelhos localizados na parte superior da tela.

Figura 49 – Código responsável pela movimentação do bastão.

```
258     Vector2 origin;
259     origin.x = objectScreenPosition.position.x + scaleObjectWidth
260         (rotatedBoxToTrack.Center.X);
261
262     if (origin.x < Camera.main.WorldToScreenPoint
263         (gameObjectMovimentado.transform.position).x)
264     {
265         //Esquerda
266         gameObjectMovimentado.transform.Translate(-velPlayer, 0, 0);
267     }
268
269     if (origin.x > Camera.main.WorldToScreenPoint
270         (gameObjectMovimentado.transform.position).x)
271     {
272         //Direita
273         gameObjectMovimentado.transform.Translate(velPlayer, 0, 0);
274     }
```

Fonte: Próprio autor

Figura 50 – Jogador tentando rebater a bolinha. 1. Do lado esquerdo; 2. Do lado direito.

Fonte: Próprio autor

No Apêndice A estão relacionadas as funções do código fonte citadas neste capítulo, bem como uma breve descrição do uso de cada uma delas.

4.5 EXPERIMENTOS E RESULTADOS

Nas seções de 4.4.1 até 4.4.7 foram apresentadas algumas figuras demonstrando os resultados obtidos com o desenvolvimento da interface natural baseada em visão computacional para o jogo “OpenCV Breakout”. Porém, para se obter um conjunto de resultados mais abrangente, se faz necessária a avaliação do desempenho do jogo desenvolvido em condições específicas, onde poderiam ocorrer alguns problemas, evidenciando assim possíveis limitações. Deste modo, foram avaliadas as situações citadas nas próximas seções.

Os experimentos foram realizados em um notebook Dell Inspiron 14-3442-A40, com processador Intel Core i5-42100U 1.70 GHz, 8GB de RAM e sistema operacional Windows 8.1. A webcam utilizada foi a integrada neste próprio notebook com resolução HD 720p (1280 *pixels* de largura e 720 *pixels* de comprimento).

Também foram utilizados um conjunto de objetos coloridos e duas bolas de tênis de coloração esverdeada. Os resultados obtidos ilustram o rastreamento destes objetos em um ambiente fechado com uma fonte de iluminação fluorescente, em diferentes condições.

Foram capturadas algumas figuras da tela do computador no momento da execução dos experimentos. Elas ilustram os resultados destes experimentos. Foram também capturadas as imagens de distribuição de probabilidade de cor responsáveis por guiar o algoritmo Camshift no rastreamento dos objetos. Para facilitar o entendimento dos resultados, as imagens de distribuição de probabilidade de cor são exibidas ao lado das telas capturadas do jogo e uma numeração indica a passagem do tempo de forma crescente.

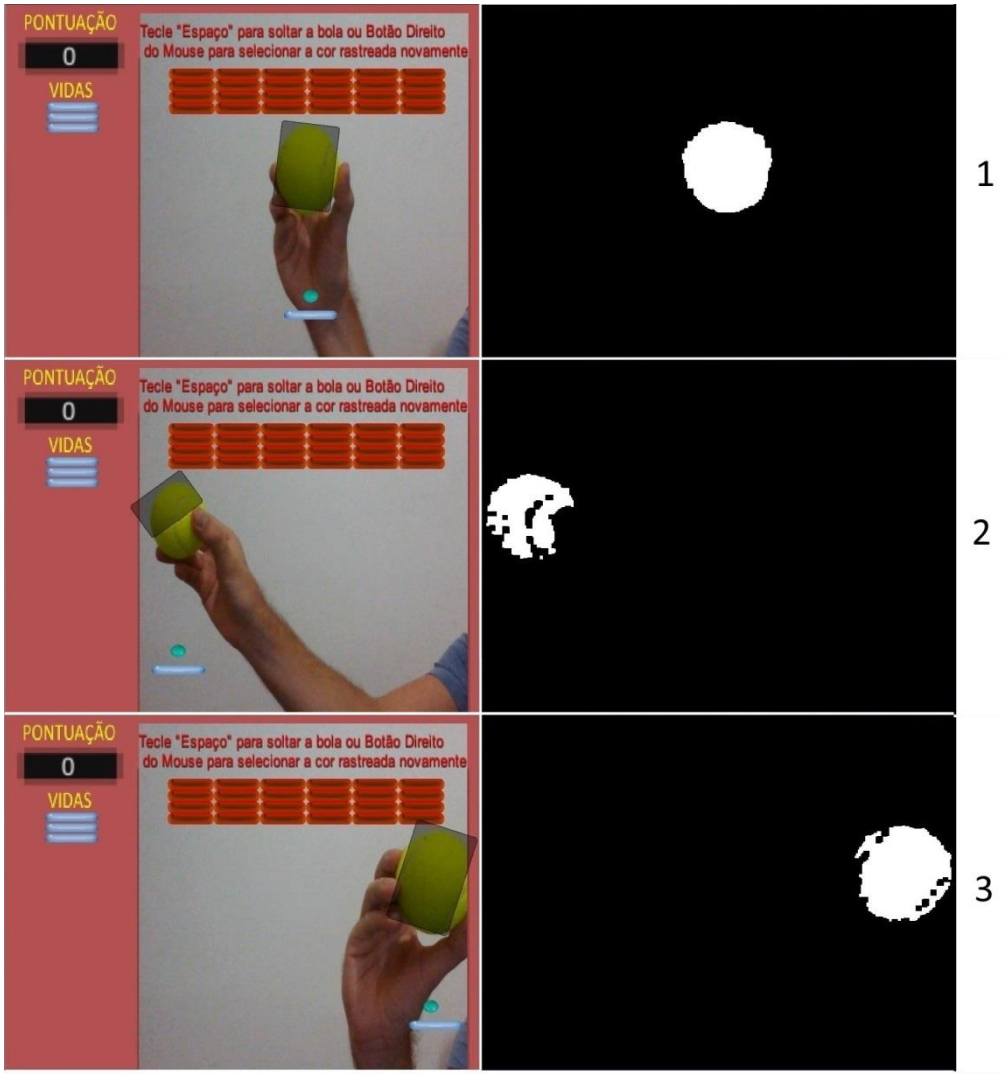
4.5.1 ILUMINAÇÃO DO AMBIENTE

A Figura 51 ilustra os resultados obtidos ao se rastrear uma bola de tênis de coloração esverdeada em um ambiente fechado com uma fonte de iluminação fluorescente. Esta figura tem a seguinte numeração:

1. A bola de tênis rastreada e o bastão azul (controlado pelo jogador) estão centralizados no ambiente de jogo.
2. A bola de tênis é movimentada para a esquerda, sendo mantido o seu rastreamento. O bastão azul é então movimentado para a esquerda.
3. A bola de tênis é movimentada para a direita, sendo mantido o seu rastreamento. O bastão azul é então movimentado para a direita.

Pode-se observar na Figura 51 que o rastreamento foi realizado de maneira satisfatória, como esperado em uma situação onde aja uma iluminação adequada. Isto permitiu ao jogador o controle sobre os movimentos do bastão azul, como consequência do rastreamento.

Figura 51 – Rastreamento em um ambiente com iluminação adequada.

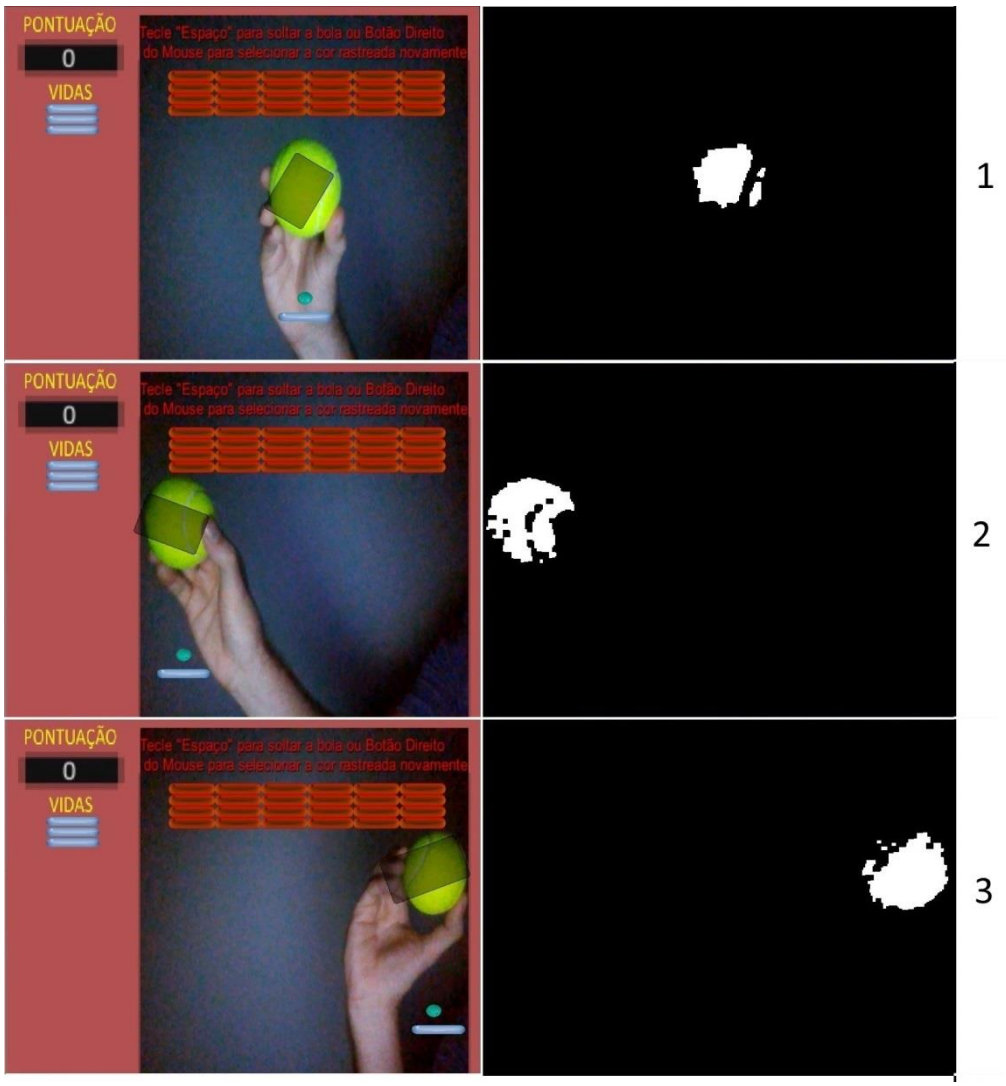


Fonte: Próprio autor

A Figura 52 ilustra os resultados obtidos ao se rastrear a bola de tênis no mesmo ambiente, porém com a luz apagada. A única iluminação é a fornecida pela própria claridade emitida pela tela do notebook. Esta figura tem a seguinte numeração:

1. A bola de tênis rastreada e o bastão azul (controlado pelo jogador) estão centralizados no ambiente de jogo.
2. A bola de tênis é movimentada para a esquerda, sendo mantido o seu rastreamento. O bastão azul é então movimentado para a esquerda.
3. A bola de tênis é movimentada para a direita, sendo mantido o seu rastreamento. O bastão azul é então movimentado para a direita.

Figura 52 – Rastreamento em um ambiente com pouca iluminação.



Fonte: Próprio autor

Pode-se observar na Figura 52 que apesar da pouca iluminação, comparada com a situação ilustrada na Figura 51, que o rastreamento novamente foi realizado de maneira satisfatória, permitindo ao jogador o controle sobre a movimentação do bastão azul, como consequência do rastreamento. Isso pode ser explicado pelo fato de o algoritmo Camshift, utilizado no rastreamento, ter sido projetado para considerar, nas imagens captadas pela webcam, apenas o canal do matiz (*hue*) no espaço de cor HSV, desconsiderando os canais de saturação (*saturation*) e brilho (*value*), minimizando desta forma os efeitos negativos da baixa iluminação. Pode ser visto na Figura 52 que embora as imagens de distribuição de probabilidade da cor esverdeada da bola de tênis apresentarem menos destaque nas regiões ocupadas

por ela, o rastreamento foi mantido, bem como o controle do jogador sobre o bastão azul.

4.5.2 OBJETOS COM A MESMA COR DO ALVO

A Figura 53 ilustra os resultados obtidos ao se rastrear uma bola de tênis de coloração esverdeada em um ambiente onde existe outro objeto com a mesma cor, neste caso outra bola de tênis. Neste experimento, a bola que está sendo rastreada está parada e a segunda bola faz um deslocamento passando entre a primeira bola e a câmera. Esta figura tem a seguinte numeração:

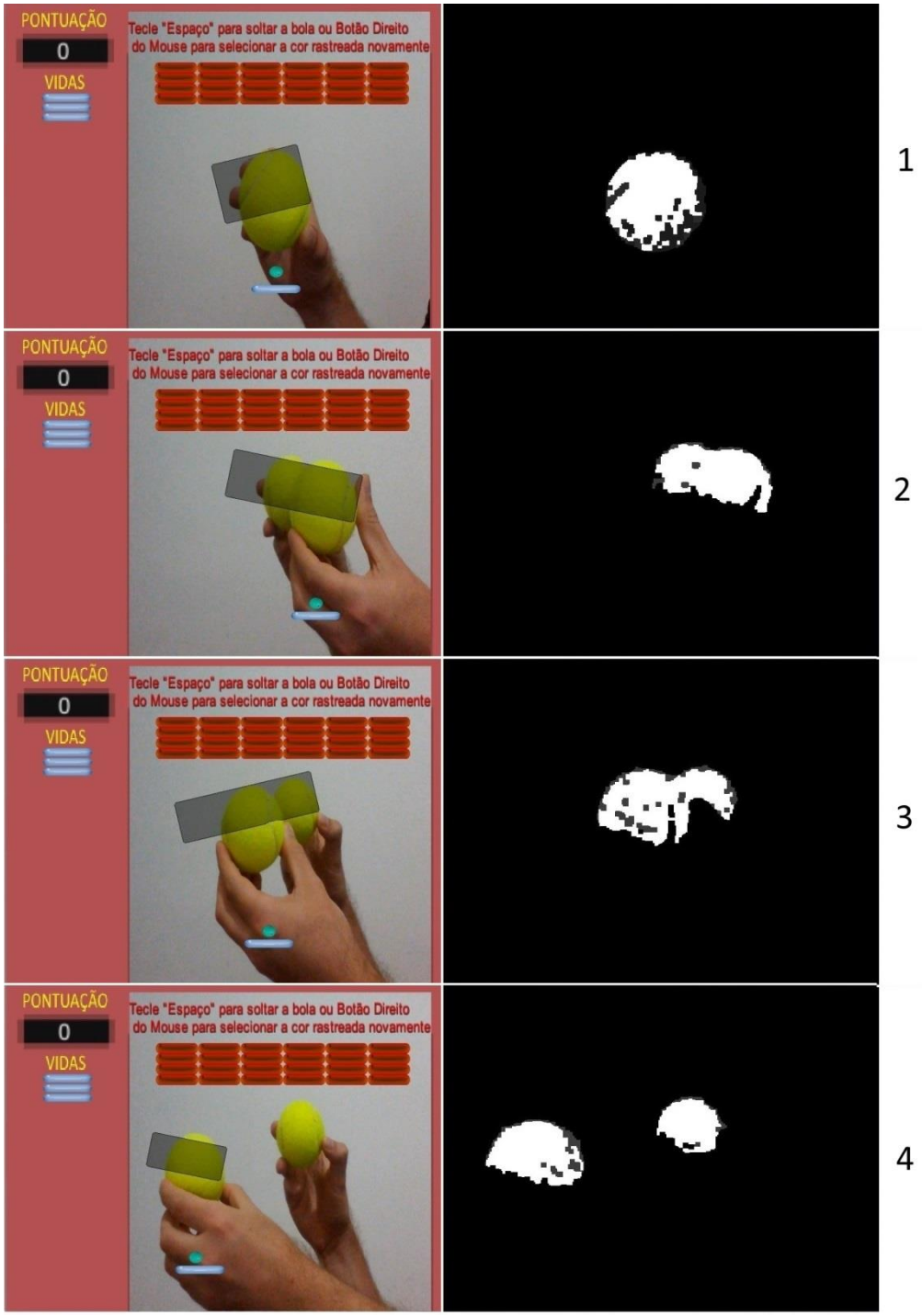
1. A bola de tênis rastreada e o bastão azul (controlado pelo jogador) estão centralizados no ambiente de jogo.
2. Entra na cena uma segunda bola de tênis (com a mesma coloração da primeira) e esta inicia um deslocamento entre a primeira bola e a câmera. Ocorre a oclusão parcial da primeira bola. A segunda bola tem maior peso na definição da posição e tamanho da janela de busca, bem como na movimentação do bastão azul.
3. A segunda bola de tênis está finalizando o seu deslocamento entre a primeira bola e a câmera. Ocorre a oclusão parcial da primeira bola. A segunda bola tem maior peso na definição da posição e tamanho da janela de busca, bem como na movimentação do bastão azul.
4. A segunda bola de tênis já não está mais entre a primeira bola e a câmera. O rastreamento agora considera somente a segunda bola de tênis. Deste modo a posição rastreada da segunda bola passa a controlar a movimentação do bastão azul.

Como observado na Figura 53, na situação deste experimento o rastreamento não foi realizado de maneira satisfatória. Quando um objeto de mesma cor do objeto que está sendo rastreado passa entre ele e a câmera, o rastreamento é afetado, tanto no momento onde os dois objetos estão próximos quanto no momento em que eles se distanciam novamente.

Quando os objetos estão próximos, como ilustrado nas Figuras 53.2 e 53.3, ocorre a união de regiões dos objetos na imagem de distribuição de probabilidade de cor que possuam a mesma cor do histograma modelo. Esta junção de regiões dos objetos reflete no calculo do centro de massa (centróide) pelo algoritmo Camshift e na tendência de manutenção da janela de busca sobre o objeto que tenha a distribuição de probabilidade de cor dominante. A distribuição dominante neste caso, pertence à bola de tênis que está mais próxima da câmera e que por isso ocupa um maior espaço na imagem captada pela câmera. Causando, portanto, uma interferência na movimentação do bastão azul, pois este é diretamente afetado pela posição central da janela de busca.

No momento em que os objetos se distanciam novamente, como ilustrado pela Figura 53.4, o objeto que estava mais próximo da câmera, acaba por “levar” consigo o rastreamento, devido à sua maior influência sobre o processo, como já explicado. Impossibilitando, portanto, o jogador de manter o controle do bastão azul utilizando o rastreamento do objeto originalmente escolhido por ele para este fim.

Figura 53 – Objeto de mesma cor passa entre o alvo e a câmera.



Fonte: Próprio autor

A Figura 54 ilustra os resultados obtidos ao se rastrear a bola de tênis de coloração esverdeada na presença da segunda bola de tênis, porém agora com a segunda se deslocando por trás da primeira. Esta figura tem a seguinte numeração:

1. A bola de tênis rastreada e o bastão azul (controlado pelo jogador) estão centralizados no ambiente de jogo.
2. Entra na cena uma segunda bola de tênis (com a mesma coloração da primeira) e esta inicia um deslocamento por trás da primeira bola. Ocorre a oclusão parcial da segunda bola. A primeira bola tem maior peso na definição da posição e tamanho da janela de busca, bem como na movimentação do bastão azul.
3. A segunda bola de tênis está finalizando o seu deslocamento por trás da primeira bola. Ocorre a oclusão parcial da segunda bola. A primeira bola tem maior peso na definição da posição e tamanho da janela de busca, bem como na movimentação do bastão azul.
4. A segunda bola de tênis finaliza completamente o seu deslocamento por trás da primeira bola. O rastreamento se mantém considerando a primeira bola de tênis. Deste modo a movimentação do bastão azul continua se baseando na posição rastreada da primeira bola.

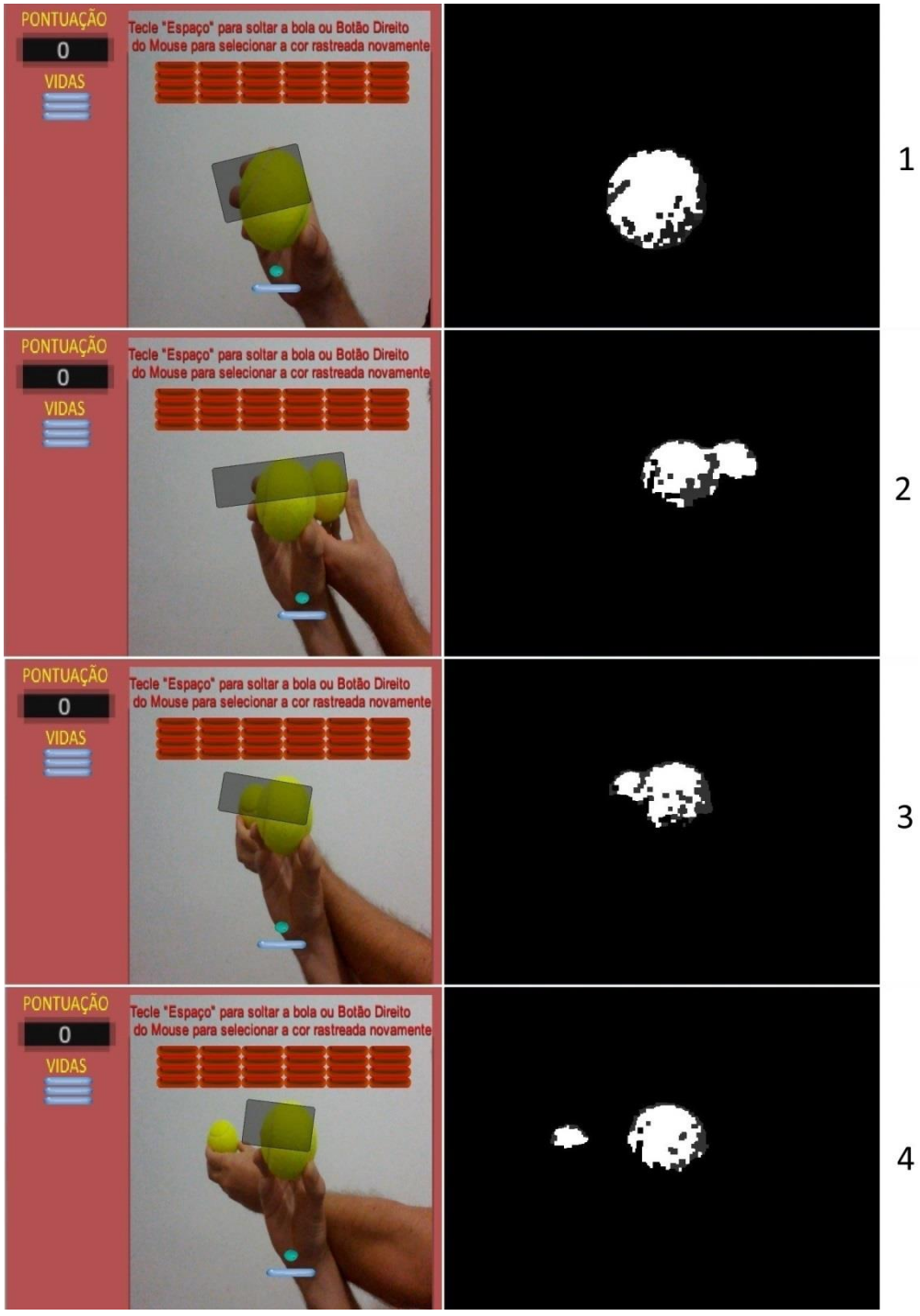
Na Figura 54 pode ser visto que na situação deste experimento o rastreamento foi realizado de maneira mais adequada comparando-se aos resultados do experimento anterior. Isto se deve ao fato de que neste experimento o objeto rastreado está sempre mais próximo da câmera do que o outro objeto de mesma cor. Isto possibilita que os efeitos negativos da presença deste outro objeto sejam minimizados.

No momento onde os objetos estão próximos (considerados como um só), como ilustrado pelas Figuras 54.2 e 54.3, o objeto originalmente rastreado tem sempre a distribuição de probabilidade de cor mais dominante comparado ao outro objeto, pela sua proximidade com a câmera. Como consequência o objeto mais próximo, causa uma maior influência no cálculo do centro de massa (centróide) do que o objeto que está mais distante da câmera. Isso minimiza a interferência causada pelo objeto que está atrás no deslocamento do bastão azul.

No momento em que os objetos se distanciam novamente, ilustrado pela Figura 55.4, pode-se notar que o objeto que passa por trás não “leva” consigo o rastreamento, pelo fato de a maior influência no rastreamento pertencer agora ao

objeto que estava originalmente sendo rastreado, pois ele está mais próximo da câmera. A movimentação do bastão azul, por consequência, se mantém baseada no rastreamento da bola de tênis originalmente escolhida.

Figura 54 – Objeto de mesma cor passa atrás do alvo.



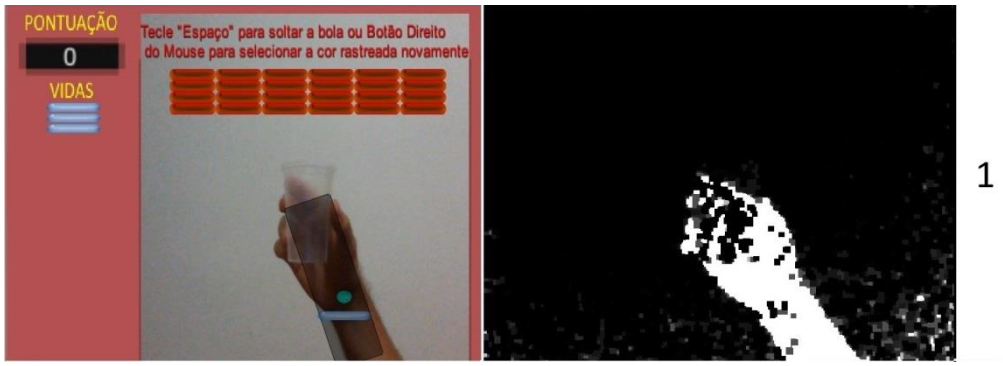
Fonte: Próprio autor

4.5.3 TRANSPARÊNCIA NO OBJETO RASTREADO

A Figura 55 ilustra os resultados obtidos ao se rastrear um objeto que possui transparência, neste caso um copo de plástico segurado pelo jogador, em um ambiente com o fundo na cor branca. Esta figura tem a seguinte numeração:

1. Após ser selecionada a região da lateral do copo de plástico como sendo a região de interesse, o resultado é o rastreamento do braço do jogador. A movimentação do bastão azul (controlado pelo jogador) fica baseada no rastreamento do braço do jogador.

Figura 55 – Rastreamento de um objeto transparente.



Fonte: Próprio autor

Observando-se a Figura 55 pode-se notar os efeitos de se basear o rastreamento em um objeto de cor transparente. O resultado é a consideração da cor do que estiver logo atrás dele, neste caso a cor da pele. Passando a cor da pele a ser a base para a construção do histograma modelo, cálculo da distribuição da probabilidade de cor, cálculo do centro de massa pela sub-rotina Mean Shift do algoritmo Camshift, os dados extraídos do rastreamento e por fim a movimentação do bastão azul dentro do jogo.

4.5.4 OCLUSÃO DO OBJETO RASTREADO

A Figura 56 ilustra os resultados obtidos ao se rastrear uma bola de tênis de coloração esverdeada em uma situação onde ocorre a oclusão da mesma por outro

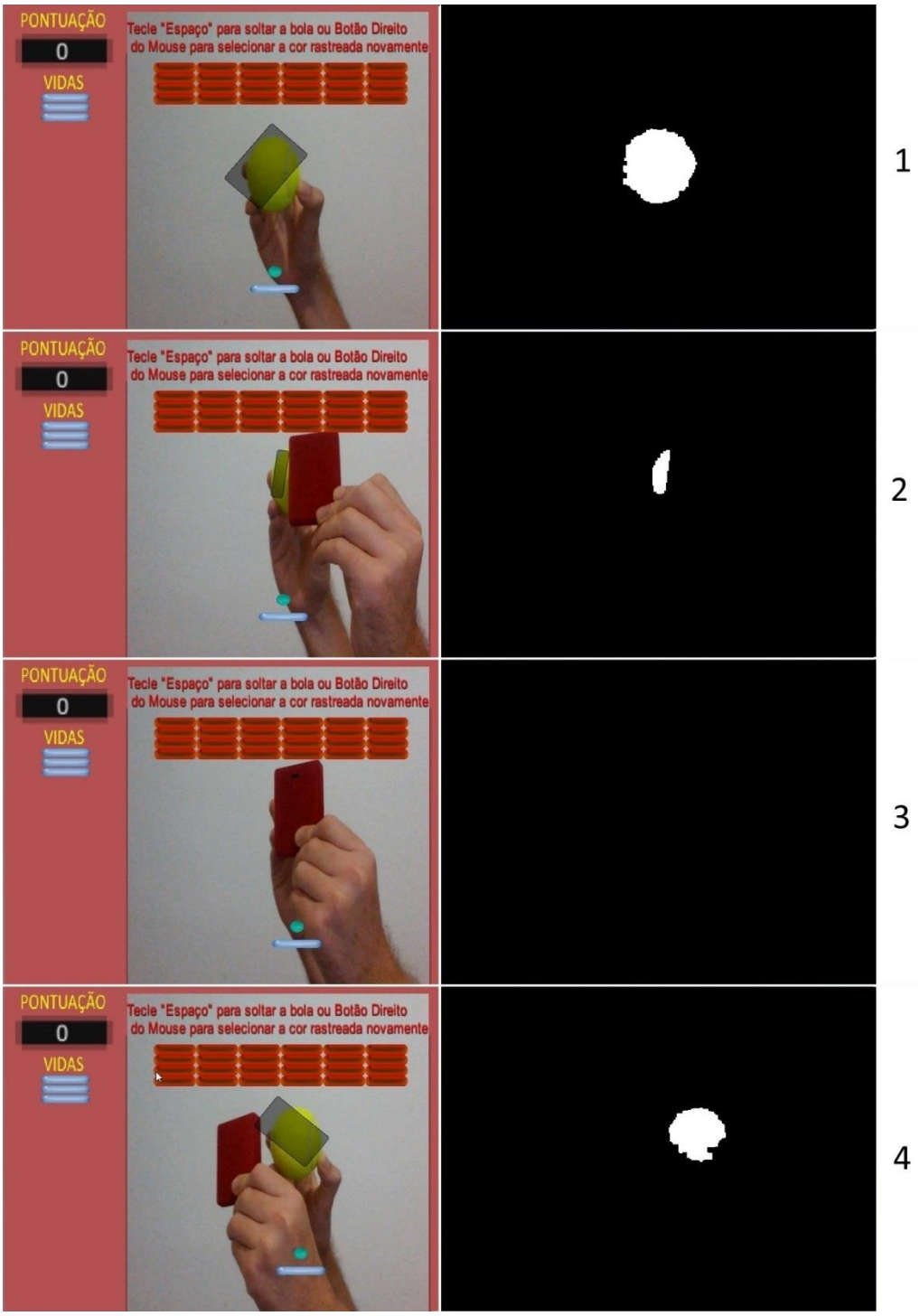
objeto, isto é a bola fica escondida atrás de outro objeto. Neste experimento, a bola de tênis fica parada e um objeto vermelho se desloca entre ela e a câmera. Esta figura tem a seguinte numeração:

1. A bola de tênis rastreada e o bastão azul (controlado pelo jogador) estão centralizados no ambiente de jogo.
2. Entra na cena um objeto vermelho e este inicia um deslocamento entre a bola de tênis e a câmera. Ocorre a oclusão parcial da bola de tênis. O rastreamento é mantido, bem como a movimentação do bastão azul.
3. Ocorre a oclusão total bola de tênis, isto é, não se pode vê-la na imagem captada, pois ela está totalmente escondida atrás do objeto vermelho. O rastreamento é perdido neste momento e a movimentação do bastão azul é interrompida.
4. O objeto vermelho já finalizou o seu deslocamento. O rastreamento da bola de tênis é recuperado, bem como a movimentação do bastão azul.

Pode-se observar na Figura 56 que, na situação da oclusão causada pelo objeto vermelho na bola de tênis, o rastreamento foi mantido para o caso de oclusão parcial, como ilustrado na Figura 56.2, mas que ele foi perdido para o caso de oclusão total da bola de tênis, como ilustrado na Figura 56.3. Isto se explica devido ao fato de o rastreamento realizado pelo algoritmo Camshif ser baseado somente nas cores dos objetos rastreados e pelo fato de que no momento ilustrado pela Figura 56.3 não ser possível visualizar a bola de tênis na imagem captada pela câmera, impossibilitando o algoritmo de rastreá-la. Neste momento de total oclusão, portanto, a movimentação do bastão azul é interrompida.

Pode-se notar na Figura 56.4 que o rastreamento da bola de tênis é recuperado quando a mesma volta a aparecer na imagem capturada. Isto se deve ao fato de que a implementação do algoritmo Camshift não interrompe a sua busca mesmo com a perda da visualização do alvo e este é novamente considerado na janela de busca e o seu rastreamento é recuperado. Considerando-se para isso que a bola de tênis não se moveu enquanto estava em oclusão total. Deste modo, a movimentação do bastão azul é também recuperada.

Figura 56 – Oclusão do objeto rastreado (1).



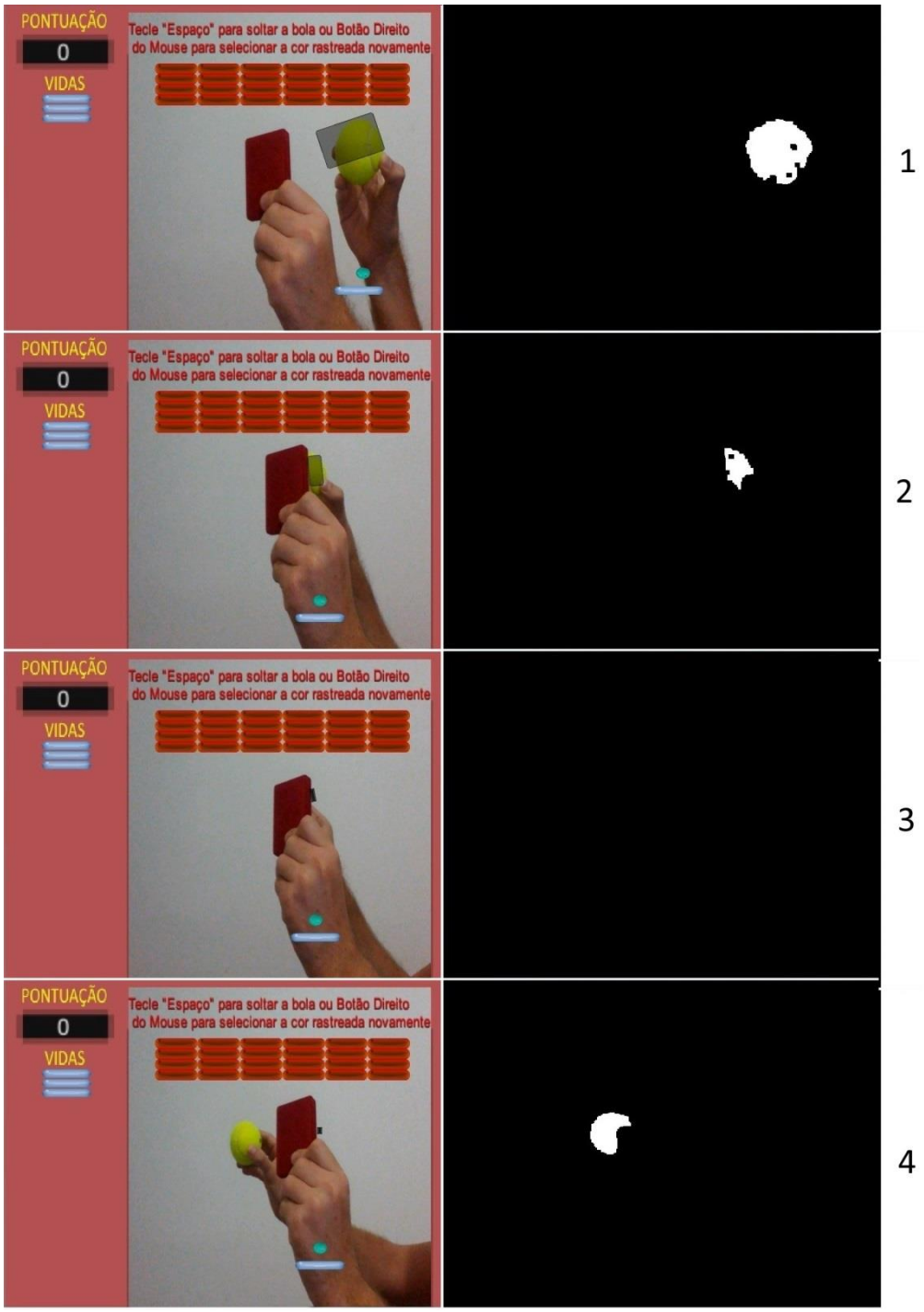
Fonte: Próprio autor

A Figura 57 ilustra os resultados obtidos ao se rastrear a mesma bola em outra situação de oclusão. Agora o objeto vermelho fica parado e é a bola de tênis que se desloca por trás dele. Esta figura tem a seguinte numeração:

1. Um objeto vermelho está centralizado no ambiente de jogo. A bola de tênis rastreada está à direita deste objeto vermelho, bem como o bastão azul (controlado pelo jogador).
2. A bola de tênis inicia um deslocamento por trás do objeto vermelho. Ocorre a oclusão parcial da bola de tênis. O rastreamento é mantido, bem como a movimentação do bastão azul.
3. Ocorre a oclusão total bola de tênis, isto é, não se pode vê-la na imagem captada, pois ela está totalmente escondida atrás do objeto vermelho. O rastreamento da bola é perdido neste momento e a movimentação do bastão azul é interrompida.
4. A bola de tênis já finalizou o seu deslocamento. O rastreamento dela não é recuperado.

Na Figura 57 pode-se notar que, com o objeto vermelho parado e com a bola de tênis se deslocando por trás do mesmo, o comportamento é semelhante ao experimento anterior, porém com a diferença ilustrada na Figura 57.4 de que como desta vez é a bola que se move, se perde a possibilidade de recuperação do rastreamento após a mesma voltar a estar presente na imagem. Isto se explica devido ao fato de que a janela de busca se manteve ativa, porém do lado contrário ao qual se deslocou a bola. Desta forma, a movimentação do bastão azul não é recuperada.

Figura 57 – Oclusão do objeto rastreado (2).



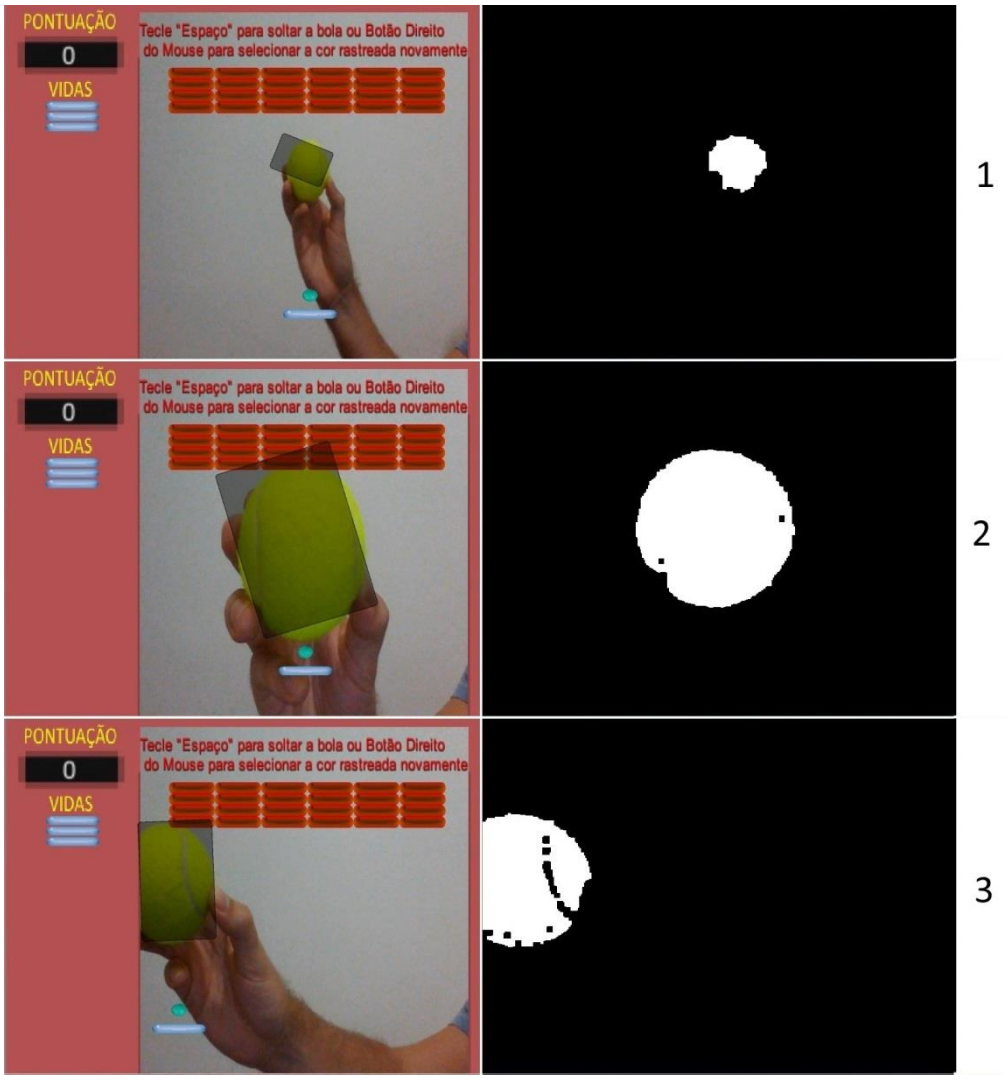
Fonte: Próprio autor

4.5.5 DISTÂNCIA ENTRE A CÂMERA E O OBJETO RASTREADO

As Figuras 58 e 59 (em conjunto) ilustram os resultados obtidos ao se rastrear uma bola de tênis de coloração esverdeada em uma situação onde ocorre a variação da distância entre esta bola e a câmera. Esta figura tem a seguinte numeração:

1. A bola de tênis rastreada e o bastão azul (controlado pelo jogador) estão centralizados no ambiente de jogo. A bola de tênis se encontra a uma distância de cerca de 30 centímetros da câmera.
2. A bola de tênis se aproxima para uma distância de cerca de 10 centímetros da câmera. O rastreamento é mantido, bem como a movimentação do bastão azul.
3. A bola de tênis é movimentada para a esquerda, sendo mantido o seu rastreamento. O bastão azul é movimentado para a esquerda.
4. A bola de tênis se afasta para uma distância de cerca de 90 centímetros da câmera. O rastreamento é mantido, bem como a movimentação do bastão azul.
5. A bola de tênis é movimentada para a direita, sendo mantido o seu rastreamento. O bastão azul é movimentado para a direita.
6. A bola de tênis se aproxima para uma distância de cerca de 10 centímetros da câmera. O rastreamento é mantido, bem como a movimentação do bastão azul.

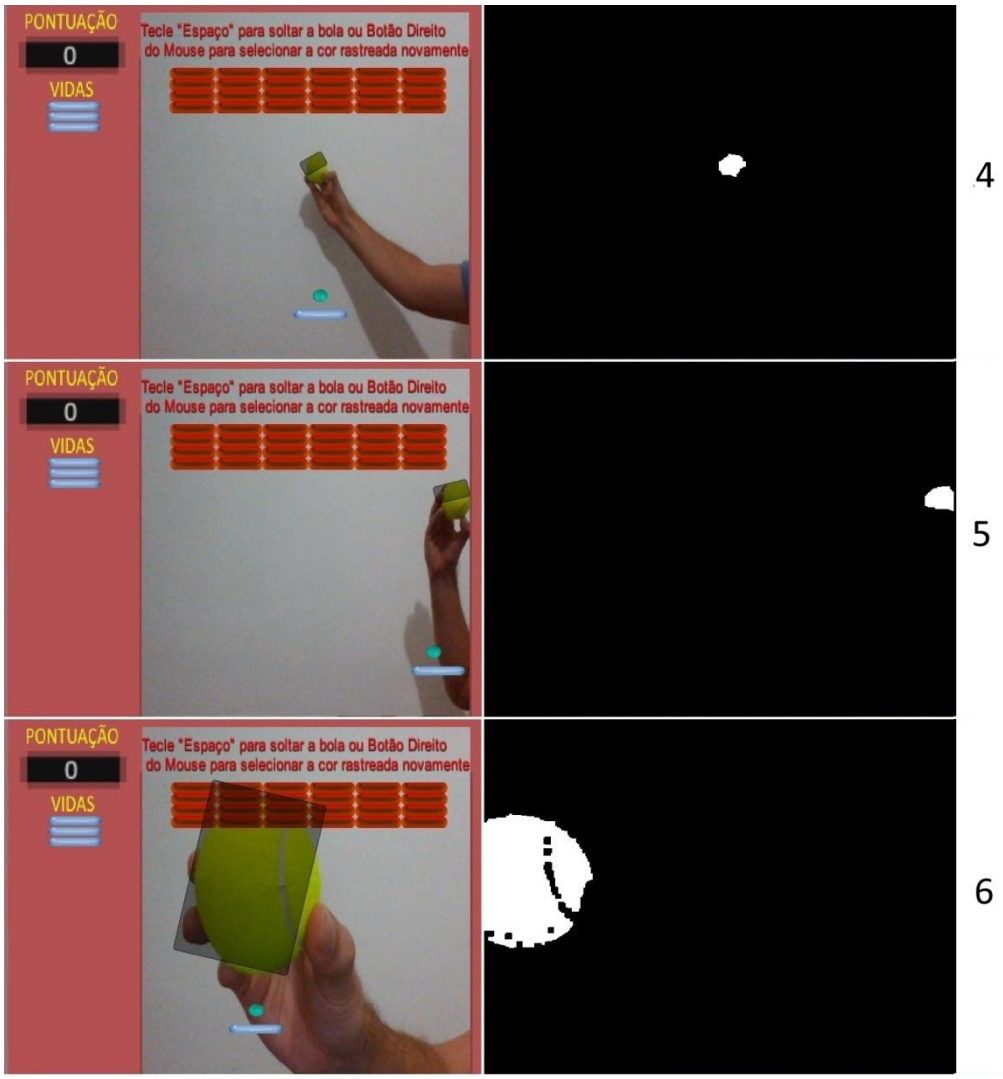
Figura 58 – Variação da distância entre a câmera e o objeto rastreado (1).



Fonte: Próprio autor

Como ilustrado nas Figuras 59 e 60, o algoritmo Camshift realiza uma adaptação do tamanho da janela de busca, sendo essa uma das características que o diferencia do algoritmo Mean Shift e um dos motivos de sua criação. Esta reavaliação do tamanho da janela faz com que não se perca o objeto rastreado conforme este se aproxima ou se afasta da câmera, possibilitando a manutenção do rastreamento e da movimentação do bastão azul pelo jogador.

Figura 59 – Variação da distância entre a câmera e o objeto rastreado (2).



Fonte: Próprio autor

4.5.6 OBJETOS COM OUTRAS CORES

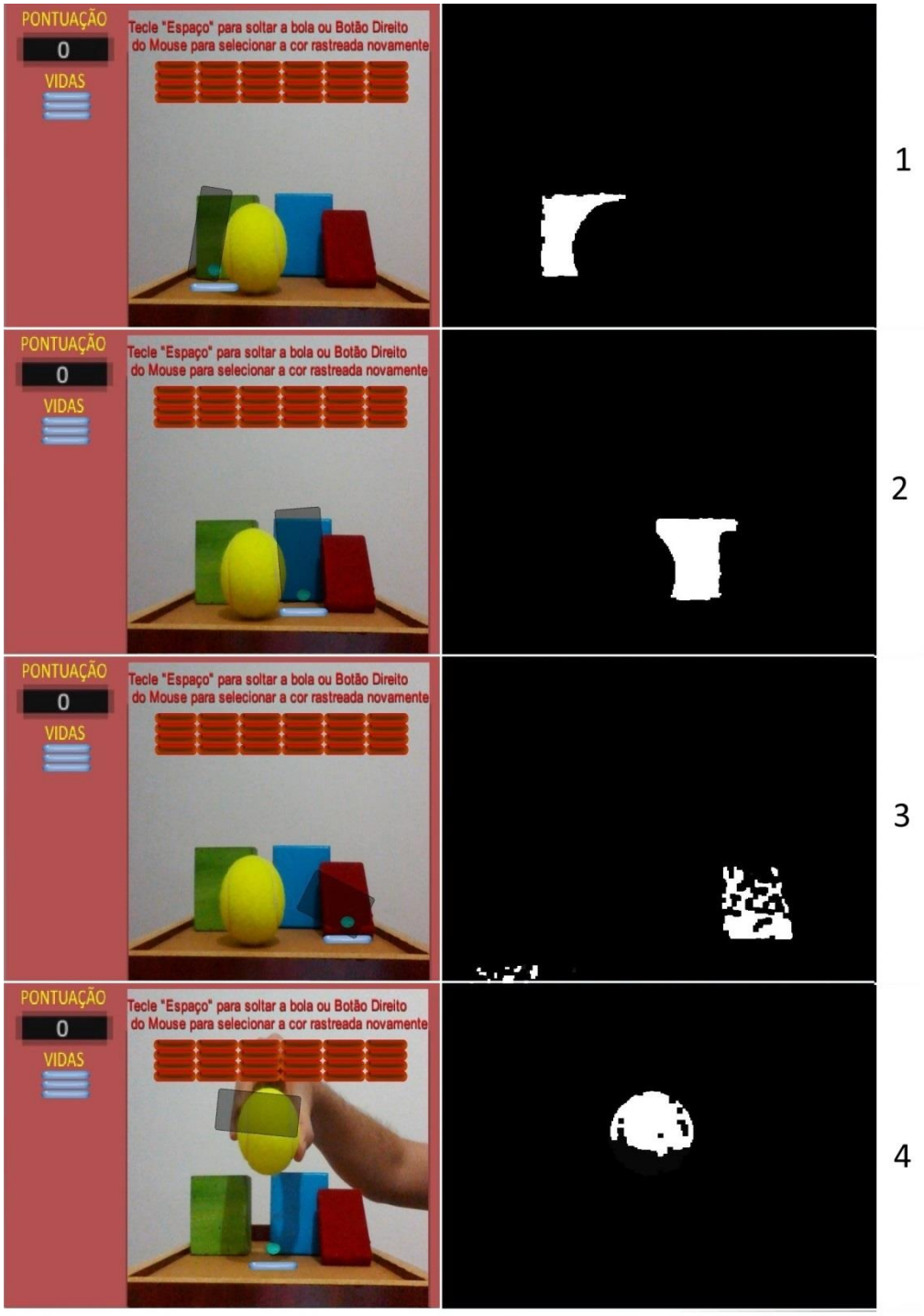
A Figura 60 ilustra os resultados obtidos ao se rastrear objetos em um ambiente onde existam vários objetos, porém cada um deles tem uma cor predominante diferente. Os elementos presentes na cena são: 1 caixa verde, 1 caixa azul, 1 objeto vermelho e uma bola de tênis esverdeada. Esta figura tem a seguinte numeração:

1. É realizado o rastreamento da caixa verde, após a sua seleção como região de interesse.

2. É realizado o rastreamento da caixa azul, após a sua seleção como região de interesse.
3. É realizado o rastreamento do objeto vermelho, após a sua seleção como região de interesse.
4. É realizado o rastreamento da bola de tênis esverdeada, após a sua seleção como região de interesse.

A situação ilustrada na Figura 60 permite avaliar as diferentes imagens de probabilidades de cor geradas de acordo com a seleção de cada um dos objetos coloridos de uma cena. Pode ser notado que os resultados são satisfatórios e que não havendo interferência de outros objetos com a mesma cor predominante do objeto sendo rastreado, não haverá maiores problemas quanto à distinção das diferentes regiões da imagem pelo algoritmo Camshift.

Figura 60 – Presença de outros objetos com outras cores na cena.



Fonte: Próprio autor

5 CONSIDERAÇÕES FINAIS

Com o propósito de utilização do algoritmo de visão computacional Camshift no desenvolvimento de jogos digitais com interfaces naturais, o presente trabalho foi iniciado com a pesquisa do material necessário para a sua fundamentação teórica em suas áreas envolvidas, sendo elas: Visão Computacional, Interação Humano-Computador (IHC) e Jogos Eletrônicos. Também foi pesquisada a já existente integração dos conceitos destas áreas.

A área de Visão Computacional foi a que ofereceu maior dificuldade de entendimento, devido a sua complexidade e a diversidade de técnicas existentes. Tal diversidade, como visto no capítulo 2, se dá pela grande variedade de situações onde a Visão Computacional é aplicada.

Em seguida foram pesquisadas algumas possibilidades de integração de tecnologias destas áreas, bem como sistemas que já apresentam tal integração. Deste modo, foram então escolhidas as ferramentas Unity 3D e OpenCVSharp, a partir das quais foi possível o desenvolvimento do jogo “OpenCV Breakout”. O jogo desenvolvido faz uso de uma webcam para a captação de imagens e da técnica de visão computacional Camshift (presente na biblioteca de visão computacional OpenCV) para a extração dos dados necessários destas imagens para o funcionamento da interação.

Nos experimentos realizados com o jogo desenvolvido, exibidos na seção 4.5, foram observadas algumas limitações do algoritmo de visão computacional utilizado. Como descrito na revisão da literatura realizada no capítulo 2, para o correto funcionamento de uma interface natural baseada em técnicas de visão computacional, é fundamental que tais técnicas tenham um desempenho que possibilite o correto funcionamento desta interface. Como o algoritmo de visão computacional responsável pela realização do rastreamento de elementos nas imagens captadas pela webcam no jogo “OpenCV Breakout” é baseado somente nas cores dos objetos presentes nas imagens captadas, a qualidade da interação do jogador com o jogo desenvolvido acaba por depender em boa parte de algumas variáveis que influenciem esta distribuição de cores.

Deve-se considerar, porém, que após ser feita uma configuração adequada do ambiente que será captado pela câmera, muitas das influências negativas ao rastreamento e a interação, podem ser minimizados. Tal configuração inclui algumas considerações de acordo com os resultados obtidos nos experimentos realizados, como:

- O ambiente a ser captado pela câmera deve possuir uma fonte adequada de luz, para evitar problemas na captação das cores dos objetos presentes.
- Deve-se evitar a presença de outros objetos de cor semelhante à cor do objeto que está sendo rastreado. Principalmente evitando que os mesmos se desloquem entre a câmera e o alvo.
- Deve-se evitar o uso de objetos de cor transparente como alvos do rastreamento, pois neste caso acabam sendo consideradas as cores do objeto presente imediatamente atrás do objeto transparente na definição do histograma modelo.
- Deve-se evitar que ocorra a oclusão do objeto rastreado, pois esta diminui a região do alvo captada pela câmera e consequentemente afeta no cálculo da distribuição de probabilidade feito com esta imagem captada e o rastreamento. Caso a ocorrência de oclusões seja inevitável, deve-se tentar manter o alvo imóvel enquanto ele estiver em oclusão total.

O baixo processamento necessário ao funcionamento do algoritmo Camshift, devido à simplicidade da sua técnica, também deve ser considerado, visto que possibilita uma gama maior de equipamentos onde se faz possível um desempenho satisfatório do *game*.

Com todas as experiências e dados levantados, pode-se notar que a interação com o jogo se mostra simples e intuitiva e ocorre de maneira razoavelmente satisfatória. Deste modo, o objetivo de integração dos diversos conceitos e ferramentas envolvidos foi atingido.

Devido à limitação de tempo, foi permitida a presença de equipamentos físicos como o teclado e o mouse na interação com o jogo. Este tipo de interação, no entanto, não foi visto como um problema, pois possibilitou a concentração da

interação natural dentro do ambiente de jogo onde as consequências dos movimentos do jogador puderam ser avaliadas de maneira mais clara. Esta mistura de tipos de interfaces na interação com um mesmo sistema pode ser observada no próprio histórico de evolução da área de IHC, como visto no capítulo 2.

Como trabalhos futuros são considerados alguns melhoramentos no jogo desenvolvido e nas técnicas de rastreamento. Também é considerada a possibilidade de utilização de tais técnicas em um jogo mais complexo.

Seria possibilitado um maior número de opções ao jogador dentro do *game* se houvesse o acréscimo de alguns elementos, como:

- O número de fases.
- O número de inimigos diferentes, com a possibilidade de pontuações diferentes para cada tipo deles.
- O número de tipos de bolinhas com diferentes características, como cor, tamanho e velocidade.
- O número de manobras possíveis ao bastão azul, como a movimentação na vertical.
- Sons causados por diferentes situações dentro do *game*, como em colisões ou no momento de vitória ou derrota.
- *Menus* mais elaborados.
- A possibilidade de pausar o jogo.

Com os melhoramentos citados, o jogo “OpenCV Breakout” pode servir como base para o desenvolvimento de um jogo mais elaborado, do ponto de vista dos elementos e opções disponíveis dentro do *game*.

A técnica de rastreamento utilizada no jogo “OpenCV Breakout” poderia ser aplicada em um jogo onde aja por exemplo, um ambiente virtual 3D com maiores possiblidades de movimentação dos seus elementos. Esta técnica também poderia sofrer melhoramentos, por meio do acréscimo de outros tratamentos, como:

- Uma detecção inicial automática da cor a ser rastreada, eliminando-se a necessidade de o jogador fazer a seleção da região de interesse. A mesma poderia ser escolhida uma única vez sendo então guardada pelo jogo e sempre utilizada como cor a ser seguida. Tal detecção poderia ser utilizada também na necessidade de recuperar-se o rastreamento do alvo, após a sua perda.
- A utilização de algoritmos preditores, capazes de estimar a posição do alvo mesmo com a sua oclusão, evitando assim que o rastreamento seja perdido.

REFERÊNCIAS BIBLIOGRÁFICAS

ALVES, D. R. **Avaliação dos Modelos de Cores RGB e HSV na segmentação de Curvas de Nível em Cartas Topográficas Coloridas**. 65 p. Dissertação (Mestrado em Engenharia Elétrica) - Pontifícia Universidade Catolica de Minas Gerais, Belo Horizonte, 2010.

ARKANOID. In: Wikipédia: a enciclopédia livre. Disponível em: <https://en.wikipedia.org/wiki/Arkanoid> Acesso em: 19 abr. 2016.

BALLARD, D. H.; BROWN, C. M. **Computer Vision**. New Jersey: Prentice-Hall, 1982. 539 p.

BRADSKI, G.; KAEHLER, A. **Learning OpenCV**. 1. ed. Sebastopol: O´ Reilly, 2008. 571 p.

BRADSKI, G. R. Computer Vision Face Tracking For Use in a Perceptual User Interface. **Intel Technology Journal**, 1998.

CAETANO, A. Visão computacional como possibilidade de interação uma cyberTV. 2009.

CANALTECH. **Como funciona o Kinect**. Disponível em: <http://canaltech.com.br/o-que-e/kinect/Como-funciona-o-Kinect/> Acesso em: 17 maio. 2016.

DEITEL, H. M. et al. **C# - como programar**. Tradução João Eduardo Nóbrega Tortello. São Paulo: Pearson Education, 2003.

DESCHAMPS, F. **Contribuições para o desenvolvimento de um sistema de visão aplicado ao monitoramento do desgaste de ferramentas de corte – o sistema Toolspy**. 139 p. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal de Santa Catarina, 2004.

EVANGELISTA, B.; SILVA, A. Criando Efeitos Fotorealistas e Não-Fotorealistas Para Jogos. **Brazilian Symposium on Games and Digital Entertainment - SBGames 2007**, 2007.

FAGERHOLT, E.; LORENTZON, M. **Beyond the HUD. User Interfaces for Increased Player Immersion in FPS Games**. 118 p. Dissertação (Mestrado em Ciência) - Divisão de Design de Interação, Chalmers University of Technology, Gotemburgo, 2009.

FILHO, D. A. M.; VIEIRA, A. Um estudo sobre as Interfaces Naturais. 2012.

FORTE, C. E. et al. Visão Computacional Aplicada ao Processo de Contagem de Células em Imagens de Imuno-Histoquímica. In: **WVC 2012, VIII Workshop de Visão Computacional**. Goiânia, 2012.

FORTE, C. E. **Identificação de pontos robustos em marcadores naturais e aplicação de metodologia baseada em aprendizagem situada no desenvolvimento de sistemas de realidade aumentada**. 116 p. Tese (Doutorado

em Engenharia Elétrica) - Universidade Presbiteriana Mackenzie, São paulo, 2015.

FRANCOIS, Alexandre R. **CAMSHIFT tracker design experiments with Intel OpenCV and SAI**. UNIVERSITY OF SOUTHERN CALIFORNIA LOS ANGELES INST FOR ROBOTICS AND INTELLIGENT SYSTEMS, 2004.

GADELHA, M. A.; SANTOS, S. R. DOS. Controle de Navegacão em Ambientes Virtuais 3D através do Rastreamento de Objetos. **VII Workshop de Realidade Virtual e Aumentada**, p. 84-89, 2010.

GAGLIARDI, G. M. **Sistema robusto de acompanhamento de trajetória de alvos móveis**. 57 p. Dissertação (Graduação em Engenharia da Computação) - Escola de Engenharia de São Carlos - Universidade de São Paulo, São Carlos, 2014.

GARBIN, S. M. **Estudo da evolução das interfaces homem-computador.** 86 p. Dissertação (Graduação em Engenharia Elétrica) - Escola de engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2010.

GONÇALVES, L. S. **Sistema de informação**. Disponível em: <http://www2.videolivraria.com.br/pdfs/6519.pdf>. Acesso em: 9 maio. 2016.

GONÇALVES, Paulo J. Sequeira; CORREIA, Luís MPM. Controlo em tempo-real, baseado em visão, de um robô móvel. **Proceedings of Engenharia**, 2005.

GONZALEZ, R. C.; WOODS, R. E. **Digital image processing**. 2. ed. New Jersey: Pearson Education, 2002.

MILANO, Danilo de; HONORATO, Luciano Barrozo. Visão computacional. **Universidade Estadual de Campinas–Faculdade de Tecnologia**, 2010.

IDIGAMES. **Câmera PS Eye: O melhor acessório para o PS4**. Disponível em: <http://www.ldigames.com/blog/index.php/2014/02/27/camera-ps-eye-o-melhor-acessorio-para-o-ps4?blog=1> Acesso em: 19 maio. 2016.

JÄHNE, B. **Digital Image Processing**. 5. ed. Berlin, Heidelberg: Springer-Verlag, 2009. 598 p.

JÄHNE, B.; HAUBECKER, H. **Computer Vision And Applications: a guide for students and practitioners**. [s.l.] Academic Press, 2000. 702 p.

TORI, R. KIRNER, C., SISCOUTO, R. **Fundamentos e tecnologia de realidade virtual e aumentada**. Belém: SBC, 2006.

LAGANIÈRE, R. **OpenCV 2 Computer Vision Application Programming Cookbook: Over 50 recipes to master this library of programming functions for real-time computer vision**. Birmingham: Packt Publishing Ltd., 2011. 304 p.

MARENGONI, M.; STRINGHINI, D. Tutorial: Introdução à Visão Computacional usando OpenCV. **Revista de Informática Teórica e Aplicada**, v. 16, n. 1, p. 125–160, 2009.

MARINATO, G. P.; GOMES, M. S.; OLIVEIRA, W. C. DE. IHC – Interação Humano

Computador : um estudo sobre a sua importância e evolução ao longo do tempo. 2015.

MARQUES FILHO, O.; NETO, H. V. **Processamento Digital de Imagens**. Rio de Janeiro: Brasport, 1999. 331 p.

MEDEIROS, A. C. S. **Interação Natural baseada em Gestos como Interface de Controle para Modelos Tridimensionais**. 63 p. Dissertação (Bacharelado em Ciência da Computação) - Universidade Federal da Paraíba, João Pessoa, 2012.

MICROSOFT. **Introdução à linguagem C# e ao .NET Framework**. Disponível em:

<https://msdn.microsoft.com/pt-br/library/z1zx9t92.aspx>. Acesso em: 13 maio. 2016.

NETO, F. D. S. **Interfaces de usuário e jogos digitais: possibilidades de aprendizagem**. 179 p. Dissertação (Mestrado em Modelagem Computacional e Tecnologia Industrial) - Faculdade de Tecnologia SENAI CIMATEC, Salvador, 2011.

OYAMA, P. I. DE C. et al. **Sistema para Classificação Automática de Café em Grãos por Cor e Forma Através de Imagens Digitais**. [s.l.] Omnipax Editora, 2012.

PASSOS, E. B. et al. Tutorial: Desenvolvimento de Jogos com Unity 3d. **Brazilian Symposium on Games and Digital Entertainment - SBGames 2009**, 2009.

PEDROSA, Danielle Cordeiro; NOTARGIACOMO, Pollyana; LOPES, Paulo Batista. Jogo Educativo Pré-escolar com Interface NUI para Ensino. **Nuevas Ideas en Informática Educativa TISE 2015**, p. 621-626, 2015.

PILLON, C. B. **Requisitos para o desenvolvimento de jogos digitais utilizando a interface natural a partir da perspectiva dos usuários idosos caidores**. 227 p. Dissertação (Mestrado em Design) - Universidade Federal do Rio Grande do Sul, Porto Alegre, 2015.

PLAYSTATION 2. In: Wikipédia: a enciclopédia livre. Disponível em: <https://pt.wikipedia.org/wiki/PlayStation_2#/media/File:PS2-Eyetoy.jpg> Acesso em: 16 maio. 2016.

PLAYSTATION Eye. In: Wikipédia: a enciclopédia livre. Disponível em: <https://en.wikipedia.org/wiki/PlayStation_Eye#/media/File:PlayStation-Eye.png> Acesso em: 16 maio 2016.

PRATES, R. O.; BARBOSA, S. D. J. Avaliação de Interfaces de Usuário – Conceitos e Métodos. **Anais do XXIII Congresso Nacional da Sociedade Brasileira de Computação. XXII Jornadas de Atualização em Informática (JAI). SBC 2003**, 2003.

RAMOS, G. DE A. **Detecção e rastreamento de lábios em dispositivos móveis**. 59 p. Dissertação (Mestrado em Ciências) - Instituto de Matemática e Estatística, Universidade de Sao Paulo, São Paulo, 2012.

REBELO, Irla B. **Interação entre homem e computador**. 2009. Disponível em: <https://irlabr.wordpress.com/apostila-de-ihc/parte-1-ihc-na-pratica/introducao-a-

interacaoentre-homem-e-computador-ihc/ >. Acesso em: 09 maio. 2016.

REINA, G. A. **OpenCVSharp for Unity**, 2014. Disponível em: <http://forum.unity3d.com/threads/opencvsharp-for-unity.278033/> Acesso em: 17 fev. 2016.

RÉZIO, A. C. C. **Reconhecimento e rastreamento de objetos**. 86 p. Dissertação (Bacharelado em Ciência da Computação) - Universidade Católica de Goiás, Goiás, 2008.

ROCHA, H. V. DA; BARANAUSKAS, M. C. C. **Design e Avaliação de Interfaces Humano-Computador**. Campinas: Universidade Estadual de Campinas, 2003.

ROCHA, C. D. S.; SOUZA, V. R. L. DE. **Design de interação para interfaces não convencionais**. Disponível em: <http://medialab.ufg.br/art/anais/textos/CleomarRocha-ViniciusSouza.pdf>. Acesso em: 20 maio. 2016.

ROGERS, Yvonne, SHARP, Helen, PREECE, Jennifer. **Design de Interação**. Bookman Editora, 2013.

SCURI, A. E. **Fundamentos da Imagem Digital**. Rio de Janeiro: Tecgraf/PUC-Rio, 2002. 95 p.

SILVA, H. et al. Estudo comparativo de soluções para o desenvolvimento de um laboratório virtual 3D interativo. 2014.

SILVA, R. C.; SILVA, A. R. Tecnologias para Construção de Mundos Virtuais: Um Comparativo Entre as Opções Existentes no Mercado. **FAZU em Revista**, n. 8, p. 211–215, 2011.

SIOLA, F. B. **Desenvolvimento de um software para reconhecimento de sinais em LIBRAS através de vídeo**. 54 p. Dissertação (Bacharelado em Ciência da Computação) - Universidade Federal do ABC, Santo André, 2010.

SIQUEIRA, T. T. **Framework para reconhecimento de maturação de frutos**. 63 p. Dissertação (Bacharelado em Ciência da Computação) - Universidade Regional do Noroeste do Estado do Rio Grande do Sul – UNIJUÍ, Ijuí, 2014.

SOLEM, J. E. **Programming Computer Vision with Python: Tools and algorithms for analyzing images**. [s.l.] O'Reilly Media, 2012. 300 p.

SZELISKI, R. **Computer Vision: Algorithms and Applications**. Londres: Springer Science & Business Media, 2010. 979 p.

TAKEMURA, C. M.; DRUCKER, D. P. Processamento de imagens digitais e gestão da informação. In: **Coleção 500 Perguntas, 500 Respostas**. [s.l.] Embrapa - Empresa Brasileira de Pesquisa Agropecuária, 2014. p. 80–91.

TECHTUDO. **Breakout, clássico do Atari, faz 37 anos com jogo 'escondido' no Google**. Disponível em: <http://www.techtudo.com.br/noticias/noticia/2013/05/breakout-classico-da-atari-faz-

37-anos-com-jogo-escondido-no-google.html> Acesso em: 12 maio. 2016.

UFRGS. **Formação das Cores**. Disponível em: <http://www.ufrgs.br/engcart/PDASR/formcor.html> Acesso em: 23 maio. 2016.

VELLOSO, B. P.; BRITO, R. F. DE; PEREIRA, A. C. Desenvolvimento de jogos baseados em visão computacional de baixo processamento para educação a distância. **4º Congresso Nacional de Ambientes Hipermídia para Aprendizagem**, 2009.

VILAR, W. T. S. **Classificação individual de sementes de mamona usando espectroscopia de reflectância no visível, imagens digitais e análises multivariadas**. 113 p. Dissertação (Mestrado em Química) - Universidade Federal da Paraíba, João Pessoa, 2014.

WIGDOR, Daniel; WIXON, Dennis. **Brave NUI world: designing natural user interfaces for touch and gesture**. Elsevier, 2011

WII. In: Wikipédia: a enciclopédia livre. Disponível em: <https://en.wikipedia.org/wiki/Wii#/media/File:Wiimote-in-Hands.jpg> Acesso em: 12 maio. 2016.

XBOX One. In: Wikipédia: a enciclopédia livre. Disponível em: <https://en.wikipedia.org/wiki/Xbox_One#/media/File:Xbox-One-Kinect.jpg> Acesso em: 18 maio. 2016.

YOUNG, I. T.; GERBRANDS, J. J.; VAN VLIET, L. J. **Fundamentals of Image Processing**. The Netherlands: Delft University of Technology, 1998. 113 p.

ZILSE, R. **Análise ergonômica do trabalho dos desenvolvedores versus o modelo mental dos usuários, tendo como foco a arquitetura da informação de websitess: estudo de caso : sites de universidades cariocas**. Dissertação (Mestrado em Design) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2004.

APÊNDICE A - Descrição de uso das funções do código fonte citadas na seção 4.4.

FUNÇÃO	*PERTENCE A BIBLIOTECA OPENCVSHARP*	*DESCRIÇÃO DE USO*
`MakePixelBox`	NÃO	Define um objeto tipo "`Rect`" da Unity 3D a partir da região selecionada.
`ConvertRect2CvRect`	NÃO	Converte um objeto do tipo "`Rect`" da Unity 3D para um objeto do tipo "`CvRect`" da OpenCVSharp.
`CheckROIBounds`	NÃO	Verifica se a região selecionada está dentro dos limites da imagem capturada pela webcam e tem retorno do tipo "`CvRect`".
`GetROI`	NÃO	Obtém parte da imagem no formato "`CvMat`" da OpenCVSharp a partir da região selecionada ("`CvRect`").
`GetSubRect`	SIM	Obtém a região da imagem do tipo "`CvMat`" a partir da região selecionada ("`CvRect`").
`CalculateOneChannelHistogram`	SIM	Calcula o histograma de apenas um canal de uma imagem com retorno do tipo "`CvHistogram`".
`ConvertToHSV`	NÃO	Converte a imagem de RGB para HSV com retorno do tipo "`CvMat`".
`CvtPixToPlane`	SIM	Extrai o canal da matiz (*hue*) da imagem HSV.
`GetImage`	SIM	Obtém uma imagem no formato "`IplImage`" a partir de uma imagem no formato "`CvMat`".
`Calc`	SIM	Calcula o histograma de uma imagem que esteja no formato "`IplImage`".
`CalculateBackProjection`	NÃO	Obtém a distribuição de probabilidade de cor do histograma modelo com o histograma da imagem da

		webcam. Internamente tem-se a função da OpenCVSharp chamada "`CalcBackProject`" que a partir da imagem HSV da webcam e considerando o histograma padrão obtém esta distribuição.
`CreateStructuringElementEx`	SIM	Criação dos elementos estruturantes (*kernels*) necessários para as funções "`Erode`" e "`Dilate`".
`Erode`	SIM	Operação para melhoria no contraste na imagem de distribuição de probabilidade.
`Dilate`	SIM	Operação para melhoria no contraste na imagem de distribuição de probabilidade.
`CamShift`	SIM	Responsável pelo rastreamento da cor de interesse pré-selecionada.
`scaleObjectWidth`	SIM	Ajusta a largura do objeto do OpenCV para a Unity 3D.
`WorldToScreenPoint`	NÃO	Encontra as posições da tela que correspondem as posições no ambiente virtual da Unity 3D.
`Translate`	NÃO	Utilizado para movimentar um objeto no Unity 3D.

Fonte: Próprio autor

APÊNDICE B – Script em C# - Bola.cs

```
using UnityEngine;
using System.Collections;

public class Bola : MonoBehaviour
{

	//// Status do Jogo - MI SR AE ED GO
	// * Menu Inicial = MI
	public GameObject objStatusMenuInicial;

	// * Seleção da Cor Rastreada = SR
	public GameObject objStatusSelecaoROI;

	// * Aguardando Espaço = AE;
	public GameObject objStatusAguardandoEspaco;

	// * Game Over = GO
	public GameObject objStatusGameOver;

	// * Vitória = VI
	public GameObject objStatusVitoria;
	////

	// Controle do Status do Jogo
	public GameObject ObjStatusJogo;
	private TextMesh txtStatusJogo;
	//

	public GameObject InimigosVermelhos;

	// Pontuação e Vida
	int pontuacao = 0;
	int vidas = 3;
	public GameObject ObjPontuacao;
	private TextMesh txtPontuacao;
	public GameObject ObjVida1;
	public GameObject ObjVida2;
	public GameObject ObjVida3;
	//

	public Transform Jogador;
	public float posBolinhaRenasc;
	public float velBola;

	private Rigidbody2D rigi;


	void Awake()
	{
		rigi = GetComponent<Rigidbody2D>();
	}

	void Start()
	{

		txtPontuacao = ObjPontuacao.GetComponent<TextMesh>();
```

```
        txtStatusJogo = ObjStatusJogo.GetComponent<TextMesh>();

        AtualizaVida();

        ControlaTextos ();

        InstanciaInimigos();

        ExibeVidas();
}

void InstanciaInimigos()
{
        Instantiate(InimigosVermelhos, new Vector3(-2.35f, -0.3f, 0),
        Quaternion.identity);
}

void ExibeVidas()
{
        ObjVida3.transform.position = new
        Vector3(ObjVida3.transform.position.x,
        ObjVida3.transform.position.y, -5);
        ObjVida2.transform.position = new
        Vector3(ObjVida2.transform.position.x,
        ObjVida2.transform.position.y, -5);
        ObjVida1.transform.position = new
        Vector3(ObjVida1.transform.position.x,
        ObjVida1.transform.position.y, -5);
}

void ControlaTextos()
{
        switch (txtStatusJogo.text)
        {
           case "MI": // * Menu Inicial = MI
               objStatusMenuInicial.transform.position = new
               Vector3(objStatusMenuInicial.transform.position.x,
               objStatusMenuInicial.transform.position.y, -5);
               objStatusSelecaoROI.transform.position = new
               Vector3(objStatusSelecaoROI.transform.position.x,
               objStatusSelecaoROI.transform.position.y, 5);
               objStatusAguardandoEspaco.transform.position = new
               Vector3(objStatusAguardandoEspaco.transform.position.x,
               objStatusAguardandoEspaco.transform.position.y, 5);
               objStatusGameOver.transform.position = new
               Vector3(objStatusGameOver.transform.position.x,
               objStatusGameOver.transform.position.y, 5);
               objStatusVitoria.transform.position = new
               Vector3(objStatusVitoria.transform.position.x,
               objStatusVitoria.transform.position.y, 5);
               break;
           case "SR": // * Seleção da Cor Rastreada = SR
               objStatusMenuInicial.transform.position = new
               Vector3(objStatusMenuInicial.transform.position.x,
               objStatusMenuInicial.transform.position.y, 5);
               objStatusSelecaoROI.transform.position = new
               Vector3(objStatusSelecaoROI.transform.position.x,
               objStatusSelecaoROI.transform.position.y, -5);
```

```
        objStatusAguardandoEspaco.transform.position = new
        Vector3(objStatusAguardandoEspaco.transform.position.x,
        objStatusAguardandoEspaco.transform.position.y, 5);
        objStatusGameOver.transform.position = new
        Vector3(objStatusGameOver.transform.position.x,
        objStatusGameOver.transform.position.y, 5);
        objStatusVitoria.transform.position = new
        Vector3(objStatusVitoria.transform.position.x,
        objStatusVitoria.transform.position.y, 5);
        break;
    case "AE": // * Aguardando Espaço = AE;
        objStatusMenuInicial.transform.position = new
        Vector3(objStatusMenuInicial.transform.position.x,
        objStatusMenuInicial.transform.position.y, 5);
        objStatusSelecaoROI.transform.position = new
        Vector3(objStatusSelecaoROI.transform.position.x,
        objStatusSelecaoROI.transform.position.y, 5);
        objStatusAguardandoEspaco.transform.position = new
        Vector3(objStatusAguardandoEspaco.transform.position.x,
        objStatusAguardandoEspaco.transform.position.y, -5);
        objStatusGameOver.transform.position = new
        Vector3(objStatusGameOver.transform.position.x,
        objStatusGameOver.transform.position.y, 5);
        objStatusVitoria.transform.position = new
        Vector3(objStatusVitoria.transform.position.x,
        objStatusVitoria.transform.position.y, 5);
        break;
    case "ED": // * Em Disputa = ED
        objStatusMenuInicial.transform.position = new
        Vector3(objStatusMenuInicial.transform.position.x,
        objStatusMenuInicial.transform.position.y, 5);
        objStatusSelecaoROI.transform.position = new
        Vector3(objStatusSelecaoROI.transform.position.x,
        objStatusSelecaoROI.transform.position.y, 5);
        objStatusAguardandoEspaco.transform.position = new
        Vector3(objStatusAguardandoEspaco.transform.position.x,
        objStatusAguardandoEspaco.transform.position.y, 5);
        objStatusGameOver.transform.position = new
        Vector3(objStatusGameOver.transform.position.x,
        objStatusGameOver.transform.position.y, 5);
        objStatusVitoria.transform.position = new
        Vector3(objStatusVitoria.transform.position.x,
        objStatusVitoria.transform.position.y, 5);
        break;
    case "GO": // * Game Over = GO
        objStatusMenuInicial.transform.position = new
        Vector3(objStatusMenuInicial.transform.position.x,
        objStatusMenuInicial.transform.position.y, 5);
        objStatusSelecaoROI.transform.position = new
        Vector3(objStatusSelecaoROI.transform.position.x    ,
        objStatusSelecaoROI.transform.position.y, 5);
        objStatusAguardandoEspaco.transform.position = new
        Vector3(objStatusAguardandoEspaco.transform.position.x,
        objStatusAguardandoEspaco.transform.position.y, 5);
        objStatusGameOver.transform.position = new
        Vector3(objStatusGameOver.transform.position.x,
        objStatusGameOver.transform.position.y, -5);
```

```
            objStatusVitoria.transform.position = new
            Vector3(objStatusVitoria.transform.position.x,
            objStatusVitoria.transform.position.y, 5);
            break;
        case "VI": // * Vitória = VI
            objStatusMenuInicial.transform.position = new
            Vector3(objStatusMenuInicial.transform.position.x,
            objStatusMenuInicial.transform.position.y, 5);
            objStatusSelecaoROI.transform.position = new
            Vector3(objStatusSelecaoROI.transform.position.x,
            objStatusSelecaoROI.transform.position.y, 5);
            objStatusAguardandoEspaco.transform.position = new
            Vector3(objStatusAguardandoEspaco.transform.position.x,
            objStatusAguardandoEspaco.transform.position.y, 5);
            objStatusGameOver.transform.position = new
            Vector3(objStatusGameOver.transform.position.x,
            objStatusGameOver.transform.position.y, 5);
            objStatusVitoria.transform.position = new
            Vector3(objStatusVitoria.transform.position.x,
            objStatusVitoria.transform.position.y, -5);
            break;
        default:
            break;
    }
}

void Update()
{
    // * Menu Inicial = I
    if (txtStatusJogo.text == "MI" && Input.GetKeyDown(KeyCode.I))
    {
        txtStatusJogo.text = "SR";
        ControlaTextos();
    }

    // * Game Over = GO ou Vitória  = VI
    if ((txtStatusJogo.text == "GO" || txtStatusJogo.text == "VI")
    && (Input.GetKeyDown("m") || Input.GetKeyDown("M")))
    {

        txtStatusJogo.text = "MI";
        ControlaTextos();

        ////Objetos na posicao inicial do Game
        transform.position = new Vector2(Jogador.position.x,
        Jogador.position.y + posBolinhaRenasc); // Bolinha
        pontuacao = 0;
        txtPontuacao.text = "0";
        vidas = 3;
        //Recolocando os Objetos de Vida e Inimigos
        InstanciaInimigos();
        ExibeVidas();
        //
        ////
    }

    // Bolinha aguardando o espaço ser pressionado
    if (txtStatusJogo.text == "AE")
    {
```

```
            transform.position = new Vector2(Jogador.position.x,
            transform.position.y);
      }
      //

      // Soltando a bolinha
      if (Input.GetKeyDown(KeyCode.Space) && txtStatusJogo.text ==
      "AE")
      {
            rigi.velocity = new Vector2(Random.Range(-2, 2), velBola);
            txtStatusJogo.text = "ED";
            ControlaTextos();
      }
      //
}

void FixedUpdate()
{
      // Ajuste de movimento da bolinha
      if (txtStatusJogo.text == "ED")
      {
            if (rigi.velocity.y < 2 && rigi.velocity.y > -2)
            {
                  rigi.gravityScale = 3;
            }
            else
            {
                  rigi.gravityScale = 0;
            }
      }
      //
}

void OnCollisionEnter2D(Collision2D outro)
{
      if (outro.gameObject.tag == "PerdeVida")
      {
            //Perde uma Vida
            vidas = vidas - 1;
            AtualizaVida();
            //

            if (vidas > 0)
            {
                  txtStatusJogo.text = "AE";
                  transform.position = new
                  Vector2(Jogador.position.x,
                  Jogador.position.y + posBolinhaRenasc);
            }

            rigi.velocity = new Vector2 (0, 0);
      }

      if (outro.gameObject.tag == "Player")
      {
            float resultadoCalculo = colisaoBolinha
            (transform.position, outro.transform.position,
            ((BoxCollider2D)outro.collider).size.x);
```

```
            if (resultadoCalculo > 0)
            {
                  resultadoCalculo = resultadoCalculo + 0.15f;
            }
            else
            {
                  resultadoCalculo = resultadoCalculo - 0.15f;
            }

            Vector2 novaDirecao = new Vector2(resultadoCalculo,
            1).normalized;

            rigi.velocity = novaDirecao * velBola;
      }
}

float colisaoBolinha(Vector2 posicaoBolinha, Vector2 posicaoJogador,
   float larguraJogador)
{
      return (posicaoBolinha.x - posicaoJogador.x) / larguraJogador;
}

void OnCollisionExit2D(Collision2D outro)
{
      if (outro.gameObject.tag == "Bloco")
      {
            Destroy(outro.gameObject);
            pontuacao = pontuacao + 10;
            AtualizaVida();
      }
}

void AtualizaVida()
{
      txtPontuacao.text = pontuacao.ToString();

      if (pontuacao >= 240)
      {
            txtStatusJogo.text = "VI";
            ControlaTextos();
      }
      else
      {
            switch (vidas)
            {
                  case 3:
                        break;
                  case 2:
                        ObjVida3.transform.position = new Vector3
                        (ObjVida3.transform.position.x,
                        ObjVida3.transform.position.y, 5);
                        break;
                  case 1:
                        ObjVida2.transform.position = new Vector3
                        (ObjVida2.transform.position.x,
                        ObjVida2.transform.position.y, 5);
                        break;
                  case 0:
```

```
                        ObjVida1.transform.position = new Vector3
                        (ObjVida1.transform.position.x,
                        ObjVida1.transform.position.y, 5);

                        txtStatusJogo.text = "GO";
                        ControlaTextos ();
                        break;
                    default:
                        break;
                }
            }
        }
}
```

Fonte: Próprio autor

APÊNDICE C – Script em C# - VideoCaptureScript.cs

```
// OpenCVSharp for Unity CAMShift Tracking
// Attach this to a Unity Game Object and that object will display
// the webcam image while tracking a region of interest defined by
// the user when they select a box with the mouse.
//
// Tony Reina
// Created: 30 October 2014
//
// $Id: VideoCaptureScript.cs 43 2014-12-25 04:58:45Z tbreina $
//
// This file may be used under the terms of the 2-clause BSD license:
//
// Copyright (c) 2014, G. Anthony Reina
// All rights reserved.
//
// Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:
//
// * Redistributions of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.
// * Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation and/or
other materials provided with the distribution.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
//

using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using OpenCvSharp;   // OpenCVSharp 2.4.9
using System.Runtime.InteropServices;

// Parallel is used to speed up the for loop when converting CvMat to 2D texture
using Uk.Org.Adcock.Parallel;  // Stewart Adcock's implementation of parallel
processing

public class VideoCaptureScript : MonoBehaviour
{
      //// Status do Jogo - MI SR AE ED GO VI
      // * Menu Inicial = MI
      public GameObject objStatusMenuInicial;
      // * Seleção da Cor Rastreada = SR
      public GameObject objStatusSelecaoROI;
      // * Aguardando Espaço = AE;
      public GameObject objStatusAguardandoEspaco;
      // * Game Over = GO
```

```
public GameObject objStatusGameOver;
// * Vitória = VI
public GameObject objStatusVitoria;
////

public GameObject ObjStatusJogo;
private TextMesh txtStatusJogo;

public GameObject Bolinha;
private Rigidbody2D rigi;
public Transform Jogador;
public float posBolinhaRenasc;

public float velPlayer;

public GameObject gameObjectMovimentado;

// Flip the video source axes (webcams are usually mirrored)
// Unity and OpenCV images are flipped
public bool FlipUpDownAxis = false, FlipLeftRightAxis = true;
// Displays the region of interest chosen by mouse
public bool DisplayROIFlag = true;
// Object parameters - rectangle where the attached gameObject is in
screen coordinates
Rect objectScreenPosition = new Rect(0, 0, 1, 1);

//// Video parameters
private WebCamTexture _webcamTexture; //Texture retrieved from the
webcam
private string deviceName;  //Input video devicename
private int imWidth;  //Input devices image width
private int imHeight;  //Input devices image height
private int imColorChannels = 3; //Number of color channels (red, blue,
green (or HSV))
private MatrixType MonoColorMatrix = MatrixType.U8C1;  //Unsigned 8-bit one
channel color (0-255)
private MatrixType TriColorMatrix = MatrixType.U8C3; //Unsigned 8-bit three
channel color (0-255)
////

CvMat videoSourceImage;  //Image from the video source (webcam)

Texture2D TextureBackProj;

//// Select region of interest (ROI)
// Select box via the mouse
// Allows user to select a sub-region (aka Region Of Interest) from the
source video
bool _mouseIsDown = false;   // Mouse button is down
Vector2 _mouseDownPos = Vector2.zero;  // Mouse position when mouse
button clicked
Vector2 _mouseLastPos = Vector2.zero;  // Current mouse position

// For CamShift
CvHistogram _histogramToTrack;  // Histogram in the region of interest
we want to keep
CvRect _rectToTrack;  // The rectangle defining the region of interest
ROI) to track
CvBox2D rotatedBoxToTrack;
```

```
bool trackFlag = false;  // If true, then track the ROI with CamShift

// For the threshold window
// HSV Theshold Range Slider values
int _hueLow = 10;
int _hueHigh = 179;
int _satLow = 10;
int _satHigh = 255;

// Display flags
bool backprojWindowFlag = false;
bool histoWindowFlag = false;
bool trackWindowFlag = false;

// Kalman filter
// Create the Kalman Filter
CvKalman _kalman;
CvPoint lastPosition;

// Use this for initialization of the class
void Start()
{

       rigi = Bolinha.GetComponent<Rigidbody2D>();

       txtStatusJogo = ObjStatusJogo.GetComponent<TextMesh>();

       //Webcam initialisation
       WebCamDevice[] devices = WebCamTexture.devices;
       //Debug.Log ("Number of video devices = " + devices.Length);

       if (devices.Length > 0)
       {
          // If there is at least one camera

         _webcamTexture = new WebCamTexture(devices[0].name); // Grab
          first camera
         //Debug.Log ("Device name = " + devices [0].name);

         // Attach camera to texture of the gameObject
         GetComponent<Renderer>().material.mainTexture =
         _webcamTexture;

         // Un-mirror the webcam image
         if (FlipLeftRightAxis)
         {
             transform.localScale = new Vector3(-
             transform.localScale.x, transform.localScale.y,
              transform.localScale.z);
         }

         if (FlipUpDownAxis)
         {
             transform.localScale = new Vector3(transform.localScale.x,
             -transform.localScale.y, transform.localScale.z);
         }

          _webcamTexture.Play();  // Play the video source
```

```
        // Get the vieo source image width and height
        imWidth = _webcamTexture.width;
        imHeight = _webcamTexture.height;

        // Create standard CvMat image based on web camera video input
        // 3 channels for color images with unsigned 8-bit depth of
        color values
        videoSourceImage = new CvMat(imHeight, imWidth,
        TriColorMatrix);
     }

}

// Find the attached GameObject's position in screen space
void FindObjectScreenPosition()
{
       // Update where the object's top left corner is in screen
       coordinates
       Vector3 offset = Vector3.zero;
       // Make offset the top-left corner of the gameObject
       offset.x = -Mathf.Abs(transform.localScale.x / 2.0f); // Half of
       the x  scale
       offset.y = Mathf.Abs(transform.localScale.y / 2.0f); // Half of
       the y scale
       // Convert world position to screen (pixel) position
       // transform.position is the exact center of the gameObject
       Vector3 objectTopLeftScreen = Camera.main.WorldToScreenPoint
       (transform.position + offset);
       // Screen y axis is flipped of world y axis.
       objectTopLeftScreen.y = Screen.height - objectTopLeftScreen.y;

       Vector3 objectBottomRightScreen =
       Camera.main.WorldToScreenPoint(transform.position - offset);
       objectBottomRightScreen.y = Screen.height -
       objectBottomRightScreen.y;

       objectScreenPosition.Set(objectTopLeftScreen.x,
       objectTopLeftScreen.y, Mathf.Abs(objectTopLeftScreen.x -
       objectBottomRightScreen.x), Mathf.Abs(objectTopLeftScreen.y -
       objectBottomRightScreen.y));
}

// Update and OnGUI are the main loops
void Update()
{
       FindObjectScreenPosition();

       if (_webcamTexture.isPlaying)
       {
          if (_webcamTexture.didUpdateThisFrame)
          {
             //convert Unity 2D texture from webcam to CvMat
             Texture2DToCvMat();

             // Do some image processing with OpenCVSharp on this image
                frame
             ProcessImage(videoSourceImage);
          }
        }
```

```
else
{
   Debug.Log("Can't find camera!");
}

if (Input.GetKeyDown(KeyCode.H))  // "h" key turns histogram
screen on/off
histoWindowFlag = !histoWindowFlag;

if (trackFlag)
{
   if (Input.GetKeyDown(KeyCode.B))  // "b" key turns back
    projection on/off
   backprojWindowFlag = !backprojWindowFlag;
   if (Input.GetKeyDown(KeyCode.T))  // "t" key turns tracking
   openCV window on
   trackWindowFlag = !trackWindowFlag;

   //// Movimentacao do Bastão
   Vector2 origin;
   origin.x = objectScreenPosition.position.x +
   scaleObjectWidth(rotatedBoxToTrack.Center.X);

   if (origin.x < Camera.main.WorldToScreenPoint
   (gameObjectMovimentado.transform.position).x)
   {
      //Esquerda
      gameObjectMovimentado.transform.Translate
       (-velPlayer, 0,0);
   }

   if (origin.x > Camera.main.WorldToScreenPoint
   (gameObjectMovimentado.transform.position).x)
  {
      //Direita
      gameObjectMovimentado.transform.Translate
       (velPlayer, 0, 0);
  }
  ////
}

if (Input.GetMouseButtonDown(1) && (txtStatusJogo.text == "ED"
|| txtStatusJogo.text == "AE")) //Botão Direito Finaliza o
 Rastreamento
{
   // Perda do Rastreamento
   txtStatusJogo.text = "SR";
   ControlaTextos();
   FimdoRastreamento();
   //
}

if (txtStatusJogo.text == "GO" || txtStatusJogo.text == "VI") //
Game Over Finaliza o Rastreamento
{
   FimdoRastreamento();
}

if (txtStatusJogo.text == "SR")
```

```
        {
            if (Input.GetMouseButtonDown(0))
            {
               // Left mouse button
               if (!_mouseIsDown)
               {
                  _mouseDownPos = Input.mousePosition;
                  trackFlag = false;
               }

               _mouseIsDown = true;
            }

            if (Input.GetMouseButtonUp(0))
            {
                // Left mouse button is up
                // If mouse went from down to up, then update the region
               of interest using the box
               if (_mouseIsDown)
               {
                  // Calculate the histogram for the selected region of
                  interest (ROI)
                  _rectToTrack = CheckROIBounds
                     (ConvertRect2CvRect(MakePixelBox
                      (_mouseDownPos, _mouseLastPos)));

                  // Use Hue channel histogram only
                  _histogramToTrack = CalculateOneChannelHistogram
                      (GetROI(videoSourceImage, _rectToTrack), 0, 179);

                  //KalmanFilter
                  lastPosition = new CvPoint(Mathf.FloorToInt
                       (_rectToTrack.X), Mathf.FloorToInt
                       (_rectToTrack.Y));
                       InitializeKalmanFilter();
                  //

                  trackFlag = true;
                  txtStatusJogo.text = "AE";
                  ControlaTextos();
               }

               _mouseIsDown = false;
            }
        }
}

void ProcessImage(CvMat _image)
{
    if (trackFlag)
    {
        CalculateCamShift(_image);
    }

    if (histoWindowFlag)
    {
       Draw1DHistogram(_image);
    }
}
```

```
// Creates an image from a 2D Histogram (x axis = Hue, y axis =   Saturation)
void DrawHSHistogram(CvHistogram hist)
{
     // Get the maximum and minimum values from the histogram
    float minValue, maxValue;
    hist.GetMinMaxValue(out minValue, out maxValue);

    int xBins = hist.Bins.GetDimSize(0);  // Number of hue bins (x axis)
    int yBins = hist.Bins.GetDimSize(1);  // Number of saturation bins
   (y axis)

    // Create an image to visualize the histogram
    int scaleHeight = 5, scaleWidth = 5;
    CvMat hist_img = new CvMat(yBins * scaleHeight, xBins * scaleWidth,
   TriColorMatrix);
    hist_img.Zero(); // Set all the pixels to black

    double binVal;
    int _intensity;
    for (int h = 0; h < xBins; h++)
    {
       for (int s = 0; s < yBins; s++)
       {
           binVal = Cv.QueryHistValue_2D(hist, h, s);
           _intensity = Cv.Round(binVal / maxValue * 255); // 0 to 255

           // Draw a rectangle (h, s) to (h+1, s+1)  (scaled by window
              size)
           // The pixel value is the color of the histogram value at bin
              (h, s)
           hist_img.Rectangle(Cv.Point(h * scaleWidth, s * scaleHeight),
               Cv.Point((h + 1) * scaleWidth - 1, (s + 1) * scaleHeight
              - 1), Cv.RGB(_intensity, _intensity, _intensity),
               Cv.FILLED);
       }
   }

       Cv.ShowImage("HS Histogram", hist_img);
}

// Creates an image from a 1D Histogram
void Draw1DHistogram(CvMat _image)
{
     float channelMax = 255;

     CvHistogram hist1 = CalculateOneChannelHistogram(_image, 0,
      channelMax);
     CvHistogram hist2 = CalculateOneChannelHistogram(_image, 1,
      channelMax);
     CvHistogram hist3 = CalculateOneChannelHistogram(_image, 2,
      channelMax);

     // Get the maximum and minimum values from the histogram
     float minValue, maxValue;
     hist1.GetMinMaxValue(out minValue, out maxValue);

     int hBins = hist1.Bins.GetDimSize(0);  // Number of bins
```

```
    // Create an image to visualize the histogram
    int scaleWidth = 3, scaleHeight = 1;
    int histWidth = hBins * imColorChannels * scaleWidth, histHeight =
    Mathf.FloorToInt(channelMax * scaleHeight);
    CvMat hist_img = new CvMat(histHeight, histWidth, TriColorMatrix);
    hist_img.Zero(); // Set all the pixels to black

    double binVal;
    int _intensity;
    for (int h = 0; h < hBins; h++)
    {
        // Draw Channel 1
       binVal = Cv.QueryHistValue_1D(hist1, h);
       _intensity = Cv.Round(binVal / maxValue * channelMax) *
       scaleHeight; // 0 to channelMax

       // Draw a rectangle (h, s) to (h+1, s+1)  (scaled by window size)
       //  The pixel value is the color of the histogram value at bin
        (h, s)
       hist_img.Rectangle(Cv.Point(h * imColorChannels * scaleWidth,
        histHeight), Cv.Point(h * imColorChannels * scaleWidth + 1,
        histHeight - _intensity), CvColor.Red, Cv.FILLED);

       // Draw Channel 2
       binVal = Cv.QueryHistValue_1D(hist2, h);
       _intensity = Cv.Round(binVal / maxValue * channelMax) *
        scaleHeight; // 0 to channelMax

       // Draw a rectangle (h, s) to (h+1, s+1)  (scaled by window size)
       //  The pixel value is the color of the histogram value at bin
        (h, s)
       hist_img.Rectangle(Cv.Point(h * imColorChannels * scaleWidth + 2,
        histHeight * scaleHeight), Cv.Point(h * imColorChannels *
        scaleWidth + 3, histHeight * scaleHeight - _intensity),
        CvColor.Blue, Cv.FILLED);

       // Draw Channel 3
       binVal = Cv.QueryHistValue_1D(hist3, h);
       _intensity = Cv.Round(binVal / maxValue * channelMax) *
        scaleHeight; // 0 to channelMax

       // Draw a rectangle (h, s) to (h+1, s+1)  (scaled by window size)
       // The pixel value is the color of the histogram value at bin (h,
        s)
       hist_img.Rectangle(Cv.Point(h * imColorChannels * scaleWidth + 4,
        histHeight * scaleHeight),
        Cv.Point(h * imColorChannels * scaleWidth + 5, histHeight *
        scaleHeight - _intensity), CvColor.Green, Cv.FILLED);
   }

    Cv.ShowImage("Histogram", hist_img);
}

// Takes an image and calculates its histogram in HSV color space
// Color images have 3 channels
// Webcam captures them in (R)ed, (G)reen, (B)lue.
// Convert to (H)ue, (S)aturation (V)alue to get better separation for
 thresholding
CvHistogram CalculateHSVHistogram(CvMat _image)
```

```
{
    // Hue, Saturation, Value or HSV is a color model that describes
       colors (hue or tint)
    // in terms of their shade (saturation or amount of gray)
    // and their brightness (value or luminance).
    // For HSV, Hue range is [0,179], Saturation range is [0,255] and
       Value range is [0,255]

    // hue varies from 0 to 179, see cvtColor
    float hueMin = 0, hueMax = 179;
    float[] hueRanges = new float[2] { hueMin, hueMax };
    // saturation varies from 0 (black-gray-white) to
    // 255 (pure spectrum color)
    float satMin = 0, satMax = 255;
    float[] saturationRanges = new float[2] { satMin, satMax };

    float valMin = 0, valMax = 255;
    float[] valueRanges = new float[2] { valMin, valMax };

    float[][] ranges = { hueRanges, saturationRanges, valueRanges };

    // Note: You don't need to use all 3 channels for the histogram.
    int hueBins = 32;  // Number of bins in the Hue histogram (more bins
         = narrower bins)
    int satBins = 32; // Number of bins in the Saturation histogram (more
         bins = narrower bins)
    int valueBins = 8;  // Number of bins in the Value histogram (more
         bins = narrower bins)

    float maxValue = 0, minValue = 0;  // Minimum and maximum value of
         calculated histogram

    // Number of bins per histogram channel
    // If we use all 3 channels (H, S, V) then the histogram will have 3
       dimensions.
    int[] hist_size = new int[] { hueBins, satBins, valueBins };

    CvHistogram hist = new CvHistogram(hist_size, HistogramFormat.Array,
     ranges, true);

    using (CvMat _imageHSV = ConvertToHSV(_image)) // Convert the image
     to HSV
    // We could keep the image in B, G, R, A if we wanted to.
    // Just split the channels into B, G, R planes

    using (CvMat imgH = new CvMat(_image.Rows, _image.Cols,  MonoColorMatrix))
    using (CvMat imgS = new CvMat(_image.Rows, _image.Cols,  MonoColorMatrix))
    using (CvMat imgV = new CvMat(_image.Rows, _image.Cols,  MonoColorMatrix))
    {

        // Break image into H, S, V planes
        // If the image were RGB, then it would split into R, G, B planes
           respectively
        _imageHSV.CvtPixToPlane(imgH, imgS, imgV, null);  // Cv.Split
         also does this

        // Store HSV planes as an IplImage array to pass to openCV's hist
         function
```

```
        IplImage[] hsvPlanes = { Cv.GetImage(imgH), Cv.GetImage(imgS),
         Cv.GetImage(imgV) };

        hist.Calc(hsvPlanes, false, null);  // Call hist function (no
         accumulatation, no mask)

        hist.GetMinMaxValue(out minValue, out maxValue);
        // Scale the histogram to unity height
        hist.Normalize(_imageHSV.Width * _imageHSV.Height * hist.Dim *
         hueMax / maxValue);
     }

     return hist;  // Return the histogram
}

// Takes an image and calculates its histogram for one channel.
// Color images have 3 channels
// Webcam captures them in (R)ed, (G)reen, (B)lue.
// Convert to (H)ue, (S)aturation (V)alue to get better separation for
   thresholding

CvHistogram CalculateOneChannelHistogram(CvMat _image, int channelNum,
   float channelMax)
{
     // Hue, Saturation, Value or HSV is a color model that describes
         colors (hue or tint)
     // in terms of their shade (saturation or amount of gray)
     //  and their brightness (value or luminance).
     // For HSV, Hue range is [0,179], Saturation range is [0,255] and
         Value range is [0,255]

    if (channelNum > imColorChannels)
        Debug.LogError("Desired channel number " + channelNum + " is out
         of range.");

    float channelMin = 0;
    float[] channelRanges = new float[2] { channelMin, channelMax };

    float[][] ranges = { channelRanges };

    // Note: You don't need to use all 3 channels for the histogram.
    int channelBins = 32;  // Number of bins in the Hue histogram (more
   bins = narrower bins)

    // Number of bins per histogram channel
    // If we use all 3 channels (H, S, V) then the histogram will have 3
      dimensions.
    int[] hist_size = new int[] { channelBins };

    CvHistogram hist = new CvHistogram(hist_size, HistogramFormat.Array,
     ranges, true);

    using (CvMat _imageHSV = ConvertToHSV(_image)) // Convert the image
     to HSV
    // We could keep the image in B, G, R, A if we wanted to.
    // Just split the channels into B, G, R planes

    using (CvMat imgChannel = new CvMat(_imageHSV.Rows, _imageHSV.Cols,
     MonoColorMatrix))
```

```
    {
         // Break image into H, S, V planes
        // If the image were BGR, then it would split into B, G, R planes
           respectively
       switch (channelNum)
        {
            case 0:
                _imageHSV.CvtPixToPlane(imgChannel, null, null, null);
                 // Cv.Split also does this
                break;
            case 1:
                _imageHSV.CvtPixToPlane(null, imgChannel, null, null);
               // Cv.Split also does this
                break;
            case 2:
                _imageHSV.CvtPixToPlane(null, null, imgChannel, null);
               // Cv.Split also does this
                break;
            default:
                Debug.LogError("Channel is out of range");
                _imageHSV.CvtPixToPlane(imgChannel, null, null, null);
               // Cv.Split also does this
                break;
        }

        hist.Calc(Cv.GetImage(imgChannel), false, null);  // Call hist
         function (no accumulatation, no mask)
     }

     return hist;  // Return the histogram
}

// Call the Back Projection method and display the results in a window
void DrawBackProjection(CvMat _image)
{
    if (trackFlag)
        Cv.ShowImage("Back Projection", CalculateBackProjection(_image,
         _histogramToTrack));
}

CvMat CalculateBackProjection(CvMat _image, CvHistogram hist)
{
    CvMat _backProject = new CvMat(_image.Rows, _image.Cols,
     MonoColorMatrix);

    using (CvMat _imageHSV = ConvertToHSV(_image)) // Convert the image
     to HSV
    using (CvMat imgH = new CvMat(_image.Rows, _image.Cols,
     MonoColorMatrix))
    using (CvMat imgS = new CvMat(_image.Rows, _image.Cols,
     MonoColorMatrix))
    using (CvMat imgV = new CvMat(_image.Rows, _image.Cols,
     MonoColorMatrix))
    {
         // Break image into H, S, V planes
        // If the image were BGR, then it would split into B, G, R planes
         respectively
        _imageHSV.CvtPixToPlane(imgH, imgS, imgV, null);  // Cv.Split
         also does this
```

```
        // Store HSV planes as an IplImage array to pass to openCV's hist
         function

        IplImage[] hsvPlanes = { Cv.GetImage (imgH), Cv.GetImage (imgS),
            Cv.GetImage (imgV)};

        hist.CalcBackProject(hsvPlanes, _backProject);

    }
    return _backProject;
}

// Convert _outBox.BoxPoints (type CvPoint2D32f) into CvPoint[][] for    use
// in DrawPolyLine
CvPoint[][] rectangleBoxPoint(CvPoint2D32f[] _box)
{
    CvPoint[] pts = new CvPoint[_box.Length];
    for (int i = 0; i < _box.Length; i++)
       pts[i] = _box[i];  // Get the box coordinates (CvPoint)

    // Now we've got the 4 corners of the tracking box returned by
       CamShift
    // in a format that DrawPolyLine can use
    return (new CvPoint[][] { pts });
}

// Set up the initial values for the Kalman filter
void InitializeKalmanFilter()
{
    // Create the Kalman Filter
    _kalman = Cv.CreateKalman(4, 2, 0);

    // Set the Kalman filter initial state
    _kalman.StatePre.Set2D(0, 0, _rectToTrack.X); // Initial X position
    _kalman.StatePre.Set2D(1, 0, _rectToTrack.Y); // Initial Y position
    _kalman.StatePre.Set2D(2, 0, 0);  // Initial X velocity
    _kalman.StatePre.Set2D(3, 0, 0);  // Initial Y velocity

    // Prediction Equations
    // X position is a function of previous X positions and previous X
       velocities
    _kalman.TransitionMatrix.Set2D(0, 0, 1);
    _kalman.TransitionMatrix.Set2D(1, 0, 0);
    _kalman.TransitionMatrix.Set2D(2, 0, 1);
    _kalman.TransitionMatrix.Set2D(3, 0, 0);

    // Y position is a function of previous Y positions and previous Y
       velocities
    _kalman.TransitionMatrix.Set2D(0, 1, 0);
    _kalman.TransitionMatrix.Set2D(1, 1, 1);
    _kalman.TransitionMatrix.Set2D(2, 1, 0);
    _kalman.TransitionMatrix.Set2D(3, 1, 1);

    // X velocity is a function of previous X velocities
    _kalman.TransitionMatrix.Set2D(0, 2, 0);
    _kalman.TransitionMatrix.Set2D(1, 2, 0);
    _kalman.TransitionMatrix.Set2D(2, 2, 1);
    _kalman.TransitionMatrix.Set2D(3, 2, 0);
```

```
   // Y velocity is a function of previous Y velocities
   _kalman.TransitionMatrix.Set2D(0, 3, 0);
   _kalman.TransitionMatrix.Set2D(1, 3, 0);
   _kalman.TransitionMatrix.Set2D(2, 3, 0);
   _kalman.TransitionMatrix.Set2D(3, 3, 1);

   // set Kalman Filter
   Cv.SetIdentity(_kalman.MeasurementMatrix, 1.0f);
   Cv.SetIdentity(_kalman.ProcessNoiseCov, 0.4f);  //0.4
   Cv.SetIdentity(_kalman.MeasurementNoiseCov, 3f); //3
   Cv.SetIdentity(_kalman.ErrorCovPost, 1.0f);
}

void DrawROIBox(CvMat _image)
{
    _image.DrawRect(_rectToTrack, CvColor.Snow);
    Cv.ShowImage("ROI", _image);
}

// Use the CamShift algorithm to track to base histogram throughout the
// succeeding frames
void CalculateCamShift(CvMat _image)
{
    CvMat _backProject = CalculateBackProjection(_image,
    _histogramToTrack);

   // Create convolution kernel for erosion and dilation
   IplConvKernel elementErode = Cv.CreateStructuringElementEx(10, 10, 5,
    5, ElementShape.Rect, null);
   IplConvKernel elementDilate = Cv.CreateStructuringElementEx(4, 4, 2,
    2, ElementShape.Rect, null);

   // Try eroding and then dilating the back projection
   // Hopefully this will get rid of the noise in favor of the blob
     objects.
   Cv.Erode(_backProject, _backProject, elementErode, 1);
   Cv.Dilate(_backProject, _backProject, elementDilate, 1);

   if (backprojWindowFlag)
   {
       Cv.ShowImage("Back Projection", _backProject);
   }

   // Parameters returned by Camshift algorithm
   CvBox2D _outBox;
   CvConnectedComp _connectComp;

   // Set the criteria for the CamShift algorithm
   // Maximum 10 iterations and at least 1 pixel change in centroid
   CvTermCriteria term_criteria = Cv.TermCriteria(CriteriaType.Iteration
    | CriteriaType.Epsilon, 10, 1);

   // Draw object center based on Kalman filter prediction
   CvMat _kalmanPrediction = _kalman.Predict();

   int predictX = Mathf.FloorToInt((float)_kalmanPrediction.GetReal2D(0,
   0));
```

```
int predictY = Mathf.FloorToInt((float)_kalmanPrediction.GetReal2D(1,
 0));

// Run the CamShift algorithm
if (Cv.CamShift(_backProject, _rectToTrack, term_criteria, out
 _connectComp, out _outBox) > 0)
{
    // Use the CamShift estimate of the object center to update the
     Kalman model
    CvMat _kalmanMeasurement = Cv.CreateMat(2, 1, MatrixType.F32C1);
    // Update Kalman model with raw data from Camshift estimate
    _kalmanMeasurement.Set2D(0, 0, _outBox.Center.X); // Raw X
     position
    _kalmanMeasurement.Set2D(1, 0, _outBox.Center.Y); // Raw Y
     position
    //_kalmanMeasurement.Set2D (2, 0, _outBox.Center.X -
     lastPosition.X);
    //_kalmanMeasurement.Set2D (3, 0, _outBox.Center.Y -
     lastPosition.Y);

    lastPosition.X = Mathf.FloorToInt(_outBox.Center.X);
    lastPosition.Y = Mathf.FloorToInt(_outBox.Center.Y);

    _kalman.Correct(_kalmanMeasurement); // Correct Kalman model with
     raw data

    // CamShift function returns two values: _connectComp and
     _outBox.

    // _connectComp contains is the newly estimated position and size
    // of the region of interest. This is passed into the subsequent
    // call to CamShift
    // Update the ROI rectangle with CamShift's new estimate of the
       ROI
    _rectToTrack = CheckROIBounds(_connectComp.Rect);

    // Draw a rectangle over the tracked ROI
    // This method will draw the rectangle but won't rotate it.
    _image.DrawRect(_rectToTrack, CvColor.Aqua);
    _image.DrawMarker(predictX, predictY, CvColor.Aqua);

    // _outBox contains a rotated rectangle esimating the position,
      size, and orientation
    // of the object we want to track (specified by the initial
      region of interest).
    // We then take this estimation and draw a rotated bounding box.
    // This method will draw the rotated rectangle
    rotatedBoxToTrack = _outBox;

    // Draw a rotated rectangle representing Camshift's estimate of
      the object's position, size, and orientation.
    _image.DrawPolyLine(rectangleBoxPoint(_outBox.BoxPoints()), true,
     CvColor.Red);
}
else
{
     UnityEngine.Debug.Log ("Object lost by Camshift tracker");
```

```
            _image.DrawMarker(predictX, predictY, CvColor.Purple,
             MarkerStyle.CircleLine);

            _rectToTrack = CheckROIBounds(new CvRect(predictX -
             Mathf.FloorToInt(_rectToTrack.Width / 2), predictY -
             Mathf.FloorToInt(_rectToTrack.Height / 2),
             _rectToTrack.Width, _rectToTrack.Height));
            _image.DrawRect(_rectToTrack, CvColor.Purple);
        }

        if (trackWindowFlag)
            Cv.ShowImage("Image", _image);
    }

    void FimdoRastreamento()
    {
            trackFlag = false;
            _mouseIsDown = false;

            rigi.velocity = new Vector2 (0, 0);
            Bolinha.transform.position = new Vector2(Jogador.position.x,
            Jogador.position.y + posBolinhaRenasc); // Bolinha na posição
            inicial
    }

    // Converts Unity's Rect type to CvRect
    // CVRect has type int and Rect has type float
    CvRect ConvertRect2CvRect(Rect _roi)
    {
        CvRect _cvroi = new CvRect(Mathf.FloorToInt(_roi.x),
                            Mathf.FloorToInt(_roi.y),
                            Mathf.FloorToInt(_roi.width),
                            Mathf.FloorToInt(_roi.height));

        return _cvroi;
    }

    // Determine if pixel box (ROI) is within the bounds of the image
    // Bounds are (0, 0, imWidth, imHeight)
    CvRect CheckROIBounds(CvRect _roi)
    {
         int _x = _roi.X, _y = _roi.Y,
        _width = Mathf.Abs(_roi.Width), _height = Mathf.Abs(_roi.Height);

        if (_roi.X < 0)
        {
            _x = 0;
            Debug.LogWarning("X is outside of image");
        }

        if (_roi.Y < 0)
        {
            _y = 0;
            Debug.LogWarning("Y is outside of image");
        }

        if (_roi.Width < 2)
            _width = 2;
        if (_roi.Height < 2)
```

```
        _height = 2;

    if ((_x + _width) > imWidth)
    {
        _width = Mathf.Abs(imWidth - _x);
        Debug.LogWarning("Width is outside of image");
    }

    if ((_y + _height) > imHeight)
    {
        _height = Mathf.Abs(imHeight - _y);
        Debug.LogWarning("Height is outside of image");
    }
    //Debug.Log (new CvRect (_x, _y, _width, _height));

    return new CvRect(_x, _y, _width, _height);
}

// Return a region of interest (_rect_roi) from within the image _image
// This doesn't need to be its own function, but I had so much trouble
// finding a method that didn't crash the program that I separated it.
CvMat GetROI(CvMat _image, CvRect rect_roi)
{
    // Get the region of interest
    CvMat img_roi;  // Get the region of interest

    // Grab the region of interest using the mouse-drawn box
    _image.GetSubRect(out img_roi, rect_roi);

    return (img_roi);
}

// Convert the Texture2D type of Unity to OpenCV's CvMat
// This uses Adcock's parallel C# code to parallelize the conversion and  make
  it faster
// I found the code execution dropped from 180 msec per frame to 70 msec  per
  frame with parallelization
void Texture2DToCvMat()
{
     //float startTime = Time.realtimeSinceStartup;
     Color[] pixels = _webcamTexture.GetPixels();

     // Parallel for loop
     Parallel.For(0, imHeight, i =>
    {
       for (var j = 0; j < imWidth; j++)
       {
           var pixel = pixels[j + i * imWidth];
           var col = new CvScalar
           {
               Val0 = (double)pixel.b * 255,
               Val1 = (double)pixel.g * 255,
               Val2 = (double)pixel.r * 255
           };

           videoSourceImage.Set2D(i, j, col);
       }
   });
```

```
    // Flip up/down dimension and right/left dimension
    if (!FlipUpDownAxis && FlipLeftRightAxis)
        Cv.Flip(videoSourceImage, videoSourceImage, FlipMode.XY);
    else if (!FlipUpDownAxis)
        Cv.Flip(videoSourceImage, videoSourceImage, FlipMode.X);
    else if (FlipLeftRightAxis)
        Cv.Flip(videoSourceImage, videoSourceImage, FlipMode.Y);
}

// Convert the image to HSV values
CvMat ConvertToHSV(CvMat img)
{
    CvMat imgHSV = img.EmptyClone();  // Assign destination matrix of
     same size and type

    Cv.CvtColor(img, imgHSV, ColorConversion.BgrToHsv);

    return (imgHSV);
}

// Use the two corners of the mouse-drawn box to define the pixel box (aka
  ROI)
// Need to check to see if the box is actually on the texture
// If not, then retrict pixel box dimensions to (0, 0, imHeight, imWidth).
Rect MakePixelBox(Vector2 point1, Vector2 point2)
{
    Vector2 _topLeft, _bottomRight;
    float boxHeight, boxWidth;

    boxHeight = Mathf.Abs(point1.y - point2.y) /
  objectScreenPosition.height * imHeight;
    boxWidth = Mathf.Abs(point1.x - point2.x) /
  objectScreenPosition.width * imWidth;

    // Top-Left corner
    // (0,0) is the bottom left corner
    // OpenCV puts (0,0) in top left corner
    _topLeft.x = (Mathf.Min(point1.x, point2.x) - objectScreenPosition.x)
  / objectScreenPosition.width * imWidth;  // Axis increases left to     right
    _topLeft.y = ((Screen.height - Mathf.Max(point1.y, point2.y)) -
  objectScreenPosition.y)
   / objectScreenPosition.height * imHeight; // Axis increases bottom to  top

   // Clamp top left corner to within image limits
   _topLeft.x = Mathf.Clamp(_topLeft.x, 0, imWidth);
   _topLeft.y = Mathf.Clamp(_topLeft.y, 0, imHeight);

   // Clamp opposite corner within image limits
   _bottomRight.x = _topLeft.x + boxWidth;
   _bottomRight.y = _topLeft.y + boxHeight;
   _bottomRight.x = Mathf.Clamp(_bottomRight.x, 0, imWidth);
   _bottomRight.y = Mathf.Clamp(_bottomRight.y, 0, imHeight);

    // Recalculate height and width based on clamped values of corners
    boxHeight = Mathf.Abs(_bottomRight.y - _topLeft.y);
    boxWidth = Mathf.Abs(_bottomRight.x - _topLeft.x);

    return new Rect(_topLeft.x, _topLeft.y, boxWidth, boxHeight);
}
```

```
// The gameObject might not have the same resolution as the webcam image
// used by OpenCV. So we need to scale the position to the gameObject's
   resolution
float scaleObjectHeight(float _height)
{
    _height = _height / imHeight * objectScreenPosition.height;

    return _height;
}

// The gameObject might not have the same resolution as the webcam image
// used by OpenCV. So we need to scale the position to the gameObject's
   resolution
float scaleObjectWidth(float _width)
{
    _width = _width / imWidth * objectScreenPosition.width;

    return _width;
}

void OnGUI()
{
    // Draw the selection box to identify region to track
    if (_mouseIsDown)
    {
        _mouseLastPos = Input.mousePosition;

        // Find the corner of the box
        Vector2 origin;

        origin.x = Mathf.Min(_mouseDownPos.x, _mouseLastPos.x);
        // GUI and mouse coordinates are the opposite way around.
        origin.y = Mathf.Max(_mouseDownPos.y, _mouseLastPos.y);

        //Compute size of box
        Vector2 size = _mouseDownPos - _mouseLastPos;

        Rect rectBox = new Rect(origin.x, Screen.height - origin.y,
         Mathf.Abs(size.x), Mathf.Abs(size.y));

        GUI.Box(rectBox, "Região\nde\nInteresse");  // Draw empty box as
         GUI overlay
    }

    // If tracking with CamShift, then draw GUI box over the tracked
      object
    else if (trackFlag)
    {
         // Figure out where the tracking box is relative to top-left
           corner of gameObject
         CvPoint p1 = rotatedBoxToTrack.BoxPoints()[1];  // Top left
         corner
        CvRect _rect = ConvertRect2CvRect(new Rect(p1.X, p1.Y,
         rotatedBoxToTrack.Size.Width, rotatedBoxToTrack.Size.Height));

        Vector2 origin;
        origin.x = objectScreenPosition.position.x +
         scaleObjectWidth(_rect.X);
```

```
        origin.y = objectScreenPosition.position.y +
         scaleObjectHeight(_rect.Y);

        GUIUtility.RotateAroundPivot(rotatedBoxToTrack.Angle, origin);
        GUI.Box(new Rect(origin.x, origin.y,
         scaleObjectWidth(_rect.Width),
         scaleObjectHeight(_rect.Height)), "");

        // Rotate GUI opposite way so that successive GUI calls won't be
         rotated.
        GUIUtility.RotateAroundPivot(-rotatedBoxToTrack.Angle, origin);
    }
}

public void OnDestroy()
{
    Cv.DestroyAllWindows();
}

void ControlaTextos()
{
     switch (txtStatusJogo.text)
     {
          case "MI": // * Menu Inicial = MI
                objStatusMenuInicial.transform.position = new Vector3
                (objStatusMenuInicial.transform.position.x,
                objStatusMenuInicial.transform.position.y, -5);
                objStatusSelecaoROI.transform.position = new Vector3
                (objStatusSelecaoROI.transform.position.x,
                objStatusSelecaoROI.transform.position.y, 5);
                objStatusAguardandoEspaco.transform.position = new Vector3
                (objStatusAguardandoEspaco.transform.position.x,
                objStatusAguardandoEspaco.transform.position.y, 5);
                objStatusGameOver.transform.position = new Vector3
                (objStatusGameOver.transform.position.x,
                objStatusGameOver.transform.position.y, 5);
                objStatusVitoria.transform.position = new Vector3
                (objStatusVitoria.transform.position.x,
                objStatusVitoria.transform.position.y, 5);
                break;
          case "SR": // * Seleção da Cor Rastreada = SR
                objStatusMenuInicial.transform.position = new Vector3
                (objStatusMenuInicial.transform.position.x,
                objStatusMenuInicial.transform.position.y, 5);
                objStatusSelecaoROI.transform.position = new Vector3
                (objStatusSelecaoROI.transform.position.x,
                objStatusSelecaoROI.transform.position.y, -5);
                objStatusAguardandoEspaco.transform.position = new Vector3
                (objStatusAguardandoEspaco.transform.position.x,
                objStatusAguardandoEspaco.transform.position.y, 5);
                objStatusGameOver.transform.position = new Vector3
                (objStatusGameOver.transform.position.x,
                objStatusGameOver.transform.position.y, 5);
                objStatusVitoria.transform.position = new Vector3
                (objStatusVitoria.transform.position.x,
                objStatusVitoria.transform.position.y, 5);
                break;
          case "AE": // * Aguardando Espaço = AE;
                objStatusMenuInicial.transform.position = new Vector3
```

```
        (objStatusMenuInicial.transform.position.x,
        objStatusMenuInicial.transform.position.y, 5);
        objStatusSelecaoROI.transform.position = new Vector3
        (objStatusSelecaoROI.transform.position.x,
        objStatusSelecaoROI.transform.position.y, 5);
        objStatusAguardandoEspaco.transform.position = new Vector3
        (objStatusAguardandoEspaco.transform.position.x,
        objStatusAguardandoEspaco.transform.position.y, -5);
        objStatusGameOver.transform.position = new Vector3
        (objStatusGameOver.transform.position.x,
        objStatusGameOver.transform.position.y, 5);
        objStatusVitoria.transform.position = new Vector3
        (objStatusVitoria.transform.position.x,
        objStatusVitoria.transform.position.y, 5);
        break;
    case "ED": // * Em Disputa = ED
        objStatusMenuInicial.transform.position = new Vector3
        (objStatusMenuInicial.transform.position.x,
        objStatusMenuInicial.transform.position.y, 5);
        objStatusSelecaoROI.transform.position = new Vector3
        (objStatusSelecaoROI.transform.position.x,
        objStatusSelecaoROI.transform.position.y, 5);
        objStatusAguardandoEspaco.transform.position = new Vector3
        (objStatusAguardandoEspaco.transform.position.x,
        objStatusAguardandoEspaco.transform.position.y, 5);
        objStatusGameOver.transform.position = new Vector3
        (objStatusGameOver.transform.position.x,
        objStatusGameOver.transform.position.y, 5);
        objStatusVitoria.transform.position = new Vector3
        (objStatusVitoria.transform.position.x,
        objStatusVitoria.transform.position.y, 5);
        break;
    case "GO": // * Game Over = GO
        objStatusMenuInicial.transform.position = new Vector3
        (objStatusMenuInicial.transform.position.x,
        objStatusMenuInicial.transform.position.y, 5);
        objStatusSelecaoROI.transform.position = new Vector3
        (objStatusSelecaoROI.transform.position.x,
        objStatusSelecaoROI.transform.position.y, 5);
        objStatusAguardandoEspaco.transform.position = new Vector3
        (objStatusAguardandoEspaco.transform.position.x,
        objStatusAguardandoEspaco.transform.position.y, 5);
        objStatusGameOver.transform.position = new Vector3
        (objStatusGameOver.transform.position.x,
        objStatusGameOver.transform.position.y, -5);
        objStatusVitoria.transform.position = new Vector3
        (objStatusVitoria.transform.position.x,
        objStatusVitoria.transform.position.y, 5);
        break;
    case "VI": // * Vitória = VI
        objStatusMenuInicial.transform.position = new Vector3
        (objStatusMenuInicial.transform.position.x,
        objStatusMenuInicial.transform.position.y, 5);
        objStatusSelecaoROI.transform.position = new Vector3
        (objStatusSelecaoROI.transform.position.x,
        objStatusSelecaoROI.transform.position.y, 5);
        objStatusAguardandoEspaco.transform.position = new Vector3
        (objStatusAguardandoEspaco.transform.position.x,
        objStatusAguardandoEspaco.transform.position.y, 5);
```

```
                    objStatusGameOver.transform.position = new Vector3
                    (objStatusGameOver.transform.position.x,
                    objStatusGameOver.transform.position.y, 5);
                    objStatusVitoria.transform.position = new Vector3
                    (objStatusVitoria.transform.position.x,
                    objStatusVitoria.transform.position.y, -5);
                    break;
                default:
                    break;
            }
        }
}
```

Fonte: Adaptado de Reina(2014)