UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM PARA DETECÇÃO DE *FOREGROUND-BACKGROUND* EM VÍDEOS

TRABALHO DE GRADUAÇÃO

Leonardo Quatrin Campagnolo

Santa Maria, RS, Brasil

2013

UMA ABORDAGEM PARA DETECÇÃO DE *FOREGROUND-BACKGROUND* EM VÍDEOS

Leonardo Quatrin Campagnolo

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

Trabalho de Graduação

Orientador: Prof. Dr. Cesar Tadeu Pozzer

354
Santa Maria, RS, Brasil

2013

Campagnolo, Leonardo Quatrin

Uma abordagem para detecção de *foreground-background* em vídeos / por Leonardo Quatrin Campagnolo. – 2013.
63 f.: il.; 30 cm.

Orientador: Cesar Tadeu Pozzer
Monografia (Graduação) - Universidade Federal de Santa Maria, Centro de Tecnologia, Curso de Ciência da Computação, RS, 2013.

1. Codebook. 2. Modelo de background. 3. Segmentação de vídeos. I. Pozzer, Cesar Tadeu. II. Título.

E-mail: lquatrin@inf.ufsm.br

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

UMA ABORDAGEM PARA DETECÇÃO DE *FOREGROUND-BACKGROUND* EM VÍDEOS

elaborado por
Leonardo Quatrin Campagnolo

como requisito parcial para obtenção do grau de
Trabalho de Graduação

COMISSÃO EXAMINADORA:

Cesar Tadeu Pozzer, Dr.
(Presidente/Orientador)

Lisandra Manzoni Fontoura, Dr. (UFSM)

Giovani Rubert Librelotto, Dr. (UFSM)

Santa Maria, 18 de Fevereiro de 2013.

AGRADECIMENTOS

Dedico meus agradecimentos inicialmente ao professor Pozzer, pela orientação durante o tempo que estive trabalhando com ele no Laboratório de Computação Aplicada (LaCA).

À minha família por todo apoio e incentivo que deram a mim nesta etapa. Aos meus pais Antonio e Liséte, aos meus irmãos Fernando e Ângela e ao Felipe, marido da minha irmã.

À minha namorada, Sabrina, pelo carinho, apoio e paciência em todos os momentos. Além disso, sempre me escutou quando eu precisava desabafar e me dava forças para seguir em frente. Obrigado por tudo, meu amor.

Aos meus colegas de aula, Alfredo, Hugo, Bruno, Vitor, Yuri, Davi, Felipe, Mariana e Ikarus, e aos meus colegas do LaCA, Schardong, Victor, Mizdal, Schirmer, Netto, Alex e Dudu, pela parceria em trabalhos, projetos e jogatinas.

Aos meus amigos do Wx3, Marcos, Diego, Bernardo, Germano, Alexandre, Vinícius, Maria Eugênia e Analu, pelos momentos de descontração, filmes, jogatinas e festas.

Por fim, obrigado ao LaTeX, pois sem ele a formatação deste trabalho seria uma desgraça.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

UMA ABORDAGEM PARA DETECÇÃO DE *FOREGROUND-BACKGROUND* EM VÍDEOS

AUTOR: LEONARDO QUATRIN CAMPAGNOLO
ORIENTADOR: CESAR TADEU POZZER
Local da Defesa e Data: Santa Maria, 18 de Fevereiro de 2013.

Este trabalho tem como objetivo desenvolver uma abordagem que, a partir de um filme provindo de uma câmera parada, seja capaz de detectar o fundo de uma cena, realçando os elementos não estáticos. Para dar suporte a esta abordagem, foi desenvolvida uma aplicação do algoritmo *Codebook Background Subtraction* (*CBS*), que consiste na detecção do foreground e background de arquivos de vídeo partindo de um treinamento da câmera com um fundo estático sem a necessidade do uso de Chroma key, sendo que a fase de treinamento é composta pela construção de uma estrutura de dados derivada das cores de cada píxel analisado. Além disso, o estudo dos espaços de cor existentes se faz necessário para que seja possível analisar qual deles pode gerar um resultado mais satisfatório. Os modelos estudados são: HLS, HSV, LAB, RGB e YCbCr. Então, deseja-se implementar o algoritmo *CBS* com diferentes espaços de cores buscando os melhores resultados, a partir da análise do comportamento do algoritmo em cada uma das implementações desenvolvidas.

Palavras-chave: Codebook. modelo de background. segmentação de vídeos.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

RGB	*Red, Green, Blue* (Vermelho, Verde, Azul)
LAB	*Luminance, A channel, B channel* (Luminosidade, Gama de cor A, Gama de cor B)
HSV	*Hue, Saturation, Value* (Matiz ou Tonalidade, Saturação, Valor ou Brilho)
HSL	*Hue, Saturation, Lightness* (Matiz ou Tonalidade, Saturação, Luminosidade)
YCrCb	*Luma, red-difference chroma component, blue-difference chroma component* (Luminância, componente de crominância do vermelho, componente de crominância do azul)
MOG	*Mixture of Gaussians*
OpenCV	*Open Source Computer Vision*
MRF	*Markov Random Field*
CBS	*CodeBook Background Subtraction*
MCR	*MATLAB Compiler Runtime*

SUMÁRIO

1 INTRODUÇÃO

Fazer a extração de objetos desconhecidos em sequências de vídeos é um problema típico na área de visão computacional. Embora a identificação dos limites de objetos e superfícies ocorrem naturalmente para um observador humano, realizar uma segmentação precisa é algo difícil e complexo. Conseguir realizar uma segmentação adequada depende principalmente da elaboração e utilização de técnicas adequadas para detectar e separar as características de cada objeto (CELENK, 1990).

Uma das abordagens mais comuns para realizar a detecção de elementos novos, móveis ou desconhecidos em uma cena é através da subtração de píxeis conhecidos em cada frame de um vídeo. Essa abordagem é intuitiva e simples, mas dependendo de como for utilizada não obtém resultados satisfatórios, muitas vezes por causa da iluminação do ambiente ser bastante variante e também pela própria iluminação artificial presente. Além disso, as características da câmera utilizada para a gravação afetam no resultado final, sendo elas: capacidade de foco automático, mudança automática de brilho e resolução de captura da câmera utilizada. Uma leve mudança de iluminação no ambiente ou no ajuste da câmera pode resultar em píxeis, que antes eram conhecidos, acabarem se tornando desconhecidos (KIM et al., 2005).

Além das características da câmera, as características do algoritmo também afetam diretamente a eficiência da detecção, uma vez que, para uma abordagem simples e intuitiva, é realizado um treinamento utilizando uma cena de referência. Neste treinamento, dados são capturados de um vídeo com o objetivo de guardar os valores provenientes de cada píxel capturado da cena de referência e guardá-los em uma estrutura de dados. Esta estrutura de dados criada é posteriormente utilizada como um modelo para detectar em uma cena de entrada quais píxeis de cada frame podem ser classificados como *foreground* ou *background*. Esta cena de entrada é construída com mesmo *background* presente na cena de referência que foi utilizada no treinamento, porém são adicionados novos elementos como pessoas ou objetos.

Kyungnam Kim introduziu uma adordagem para modelagem de múltiplos *backgrounds* em uma estrutura compacta, chamado de *CodeBook Background Subtraction* (KIM et al., 2005). O algoritmo é baseado na detecção de píxeis através de funções de distorção de cor e brilho, calculados a partir da representação de dois píxeis em um espaço de cor (RGB, HSL, HSV, etc). Este algoritmo se mostra bastante robusto para variações de iluminação e movimentação, mesmo apresentando alguns problemas como a presença contínua de ruídos e detecções

incompletas de elementos pertencentes ao *foreground*.

Nesse trabalho pretende-se realizar uma pesquisa para estudar possíveis estratégias de detecção de *foreground-background* de uma cena utilizando diferentes abordagens, partindo do algoritmo *Codebook background subtraction*. O algoritmo será usado para composição de cenas, evitando a necessidade do uso da técnica de efeito visual Chroma key. Esta, por sua vez, é utilizada para isolar os personagens ou objetos de interesse a partir da escolha de uma cor padrão de fundo, que é posteriormente anulada. Além da implementação, pretende-se buscar formas de melhorar o resultado final gerado pela implementação do algoritmo base, tornando-o mais satisfatório. Dentre as estratégias previstas estão: estudo de outros espaços de cor, buscando maneiras de modelar os valores de cor capturados e a utilização de técnicas de processamento de imagens para refinar o resultado, como operadores morfológicos e *FloodFill*.

O algoritmo *Codebook background subtraction* realiza a detecção do *background* de uma cena através da análise do comportamento de cada píxel da tela. Tal análise é feita de forma independente, ou seja, o resultado da detecção de um píxel não afeta no resultado de outro píxel vizinho.

1.1 Objetivos

1.1.1 Objetivos gerais

Tendo em vista a segmentação de *foreground-background* de vídeos, esse trabalho tem como objetivo propor uma abordagem que possibilite a detecção de elementos pertencentes ao *background* de um vídeo de outros novos elementos ou elementos não estáticos (como pessoas ou objetos) sem a utilização de chroma keys. Além disso, realizar um estudo sobre os espaços de cor e técnicas de segmentação e processamento de imagens existentes para posteriormente serem utilizadas em outras versões modificadas da implementação inicial, buscando melhores resultados.

Para alcançar os objetivos desse trabalho pretende-se utilizar o algoritmo *CodeBook Background Subtraction* juntamente com algoritmos de pós-processamento para a manipulação de imagens e modificações do algoritmo que será utilizado. O algoritmo *CodeBook Background Subtraction* proporciona a composição de cenas de múltiplos backgrounds em uma estrutura de dados simples e compacta.

Por esse trabalho se tratar de uma pesquisa buscando diferentes abordagens para um

mesmo problema, o objetivo final será analisar o comportamento de todas as implementações desenvolvidas com diferentes vídeos e verificar, para cada vídeo testado, qual a melhor estratégia a ser utilizada.

1.1.2 Objetivos específicos

Define-se como objetivos específicos de desenvolvimento:

- Construção modularizada de forma que o programa possa ser facilmente modificado;
- Análise, compreensão e implementação do algoritmo *Codebook background subtraction* proposto por Kyungnam Kim;
- Realizar uma pré-análise dos resultados gerados a partir da implementação inicial;
- Estudar os espaços de cor existentes e funções para processamento de imagens;
- Adicionar métodos ao algoritmo para refinar os resultados parciais gerados a partir da implementação inicial;
- Comparação dos resultados obtidos em cada uma das implementações para verificar qual delas gerou resultados mais satisfatórios.

1.2 Organização do Texto

O presente trabalho está organizado da seguinte forma: o capítulo 2 contém uma revisão bibliográfica abordando os tópicos de interesse para o trabalho, como modelos de *background*, o algoritmo *CodeBook Background Subtraction*, espaços de cor e funções de processamento de imagens. O capítulo 3 relata os métodos e as técnicas utilizadas para esse trabalho.

No capítulo 4 é detalhada a estrutura da implementação do trabalho. No capítulo 5 são apresentados e comentados os resultados obtidos provindos de cada uma das abordagens propostas. O capítulo 6 contém as conclusões do trabalho e sugestoes para trabalhos futuros.

Por fim, é apresentada a bibliografia utilizada para a elaboração do presente trabalho.

2 REVISÃO BIBLIOGRÁFICA

Em Visão Computacional, a detecção de *foreground* e *background* em vídeos é um processo que consiste em particionar cada frame em duas camadas, simplificando sua representação. Um frame é considerado um quadro ou uma imagem fixa pertencente a um vídeo. Logo, os vídeos podem ser interpretados como sequências de imagens ou sequência de frames. A segmentação de imagens é muito utilizada para o reconhecimento de objetos em imagens, através da segmentação de conjuntos de píxeis. Desta forma, é possível isolar os elementos de interesse de uma imagem em diferentes camadas (SHAPIRO; STOCKMAN, 2001).

No presente capítulo serão apresentados conceitos relacionados a *foreground* e *background* e alguns trabalhos relacionados sobre modelagem de *backgrounds*. Posteriormente, o algoritmo *CodeBook Background Subtraction* será detalhado, mostrando as características, os passos principais de execução e como são calculadas as funções de distorção de cor e brilho para a classificação e cálculo de distância entre píxeis. Por fim, são apresentadas as características de alguns espaços de cor e técnicas de pós-processamento para manipulação de imagens que são utilizadas nesse trabalho.

2.1 *Foreground* e *Background*

O *Background* de uma cena, ou também chamado de fundo de uma cena, pode ser considerado como os elementos que se mantêm praticamente estáticos durante um vídeo, como uma parede, um quadro na parede ou algum objeto que não se move ou possui movimentos periódicos durante uma filmagem.

Já o *Foreground* de uma cena , ou também chamado de frente de uma cena, é considerado como os elementos que são adicionados à uma cena que corresponde a um *background*. Estes elementos podem ou não se mover durante a cena. A figura 2.1 exemplifica esta classificação.

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

Figura 2.1: (a) Cena de Referência, (b) Cena de entrada, (c) *Foreground* da cena e (d) *Background* da cena.

2.2 Modelo de *Background*

A modelagem de *backgrounds* em algoritmos de segmentação entre *foreground* e *background* é de extrema importância para que seja possível alcançar um resultado satisfatório, ou seja, uma boa detecção de *foreground* e *background*. Thanarat Horprasert demonstra um algoritmo que modela o fundo através de uma distribuição unimodal (HORPRASERT; HARWOOD; DAVIS, 1999), porém esta abordagem acaba gerando conflito na segmentação quando a cena apresenta fundos não estáticos, como o movimento das folhas em uma árvore, por exemplo. Dessa forma, estes eventos que ocorrem no *background* da cena acabam sendo classificados como *foreground*.

Para a modelagem de pequenas variações durante uma cena em *backgrounds* não estáticos, foi proposto um modelo de fundo utilizando MOG (*Mixture of Gaussians*) (STAUFFER; GRIMSON, 1999). Porém uma das grandes desvantagens deste modelo é a dificuldade de modelar *backgrounds* que apresentam rápidas variações, diminuindo a confiabilidade do método quando for necessária uma detecção mais precisa. Além disso, dependendo do tipo de treinamento realizado, o algoritmo pode, muitas vezes, deixar de detectar objetos que aparecem de relance em uma cena, e tambem inferir que objetos que se movem lentamente façam parte do *background* da cena (TOYAMA et al., 1999).

Partindo deste modelo utilizando MOG para a detecção de *backgrounds* de uma cena, outros trabalhos utlizando esta técnica como base foram desenvolvidos, utilizando redes bayesianas (LEE; HULL; EROL, 2003), utilizando abordagens de detecção através das informações de regiões dos píxeis (CRISTANI; BICEGO; MURINO, 2002). Outros trabalhos relacionados sobre modelagem de *backgrounds* utilizando MOG são discutidos por Bouwmans (BOUWMANS; BAF; VACHON, 2008).

A classificação de *backgrounds* em clusters possibilitou a detecção de pequenas variações de cor durante uma cena de entrada sem a necessidade de uma grande quantidade de memória, é o caso do algoritmo NHD proposto por Butler (BUTLER; JR.; SRIDHARAN, 2005). Nessa aplicação, a cena de referência é modelada através de clusters que possuem um centróide (representado por um píxel) e um valor de peso que representa a quantidade de píxeis que corresponderam com este cluster, ou seja, a probabilidade deste cluster ter um píxel pertencente ao *background* da cena. O número de clusters nesse algoritmo varia de acordo com a precisão e complexidade computacional desejada. Além disso, são mantidas informações sobre o histórico

de cada píxel. Neste algoritmo, deve-se assumir que a câmera não possui ajustes automáticos de ganho e brilho.

Esta classificação de píxeis em clusters é dada partir da distância de Manhattan entre o centróide do cluster e o píxel do frame. Dessa forma, é possivel atualizar o valor do centróide dos cluster a cada píxel novo computado, depois de gerada a estrutura de clusters com os pesos provenientes da fase de treino.

Seguindo a ideia da utilização de clusters para a modelagem de fundos, tem-se o algoritmo proposto por Kyungnam Kim (KIM et al., 2005), chamado *CodeBook Background Subtraction*, que introduz uma estratégia para a modelagem de múltiplos backgrounds. Dessa forma, é possível guardar em uma estrutura compacta pequenas variações que ocorrem em uma cena de referência (*background*). É um algoritmo que adota a técnica de quantização e clustering para construir um modelo de *background* de uma cena a partir de um série de observações a respeito do comportamento de cada píxel separadamente. Este algoritmo será detalhado nas próximas seções e é o algoritmo base desse trabalho.

Um trabalho publicado por Mohamad Hoseyn Sigari(SIGARI; FATHY, 1999), baseado no algoritmo CodeBook Background Subtraction (KIM et al., 2005), propôs a ideia do densenvolvimento de um algoritmo que possui duas camadas de detecção. A primeira camada é correspondente ao algoritmo base, que treina um CodeBook a partir dos píxeis de uma cena de referência, e a segunda camada consiste em um CodeBook que contém píxeis de fundo não permanentes que aparecem fora do treinamento (ou seja, na cena de entrada).

A utilização de GPUs veio para auxiliar na rapidez destes algoritmos e muitas vezes alcança resultados além do esperado, é o caso do algoritmo apresentado por Andreas Griesser (GRIESSER et al., 2005). Este trabalho apresenta um algoritmo em GPU que realiza a segmentação de vídeos em menos de 4ms por frame. O trabalho utiliza testes de similaridade de cor entre pequenas vizinhança de píxeis, integrado a um framework bayesiano de estimativas. Utiliza técnicas de adaptação de *threshold* para determinação do valor limiar de decisão entre o que é *foreground* ou não, compensação de brilho em regiões da imagem e um *MRF framework* para o refinamento dos resultados, explorando o paralelismo suportado pelas placas gráficas.

2.3 Codebook Background Subtraction

Como foi mencionado na subseção anterior de Modelo de *backgrounds*, o algoritmo Codebook background subtraction (KIM et al., 2005) adota a técnica de quantização e clustering

para construir um modelo do fundo de uma cena a partir de um série de observações a respeito do comportamento de cada píxel separadamente. O algoritmo possui algumas características principais, sendo elas:

- Constrói um modelo de background com suporte a pequenas variações no *background*, como a presença de eventos periódicos (movimentação das folhas de uma árvores, por exemplo);
- Capacidade de armazenar diferentes características (cores e intensidades de brilho) para cada píxel, suportando múltiplos *backgrounds*;
- Detecta, a partir de um modelo de *background* construído, quais regiões de uma imagem fazem parte do modelo construído e quais regiões não fazem parte, sendo estas classificadas como *foreground*;
- Cria de uma estrura de dados compacta que armazena as características do *background* correspondente para cada píxel.

Este método possui duas partes principais: a primeira parte, que será chamada de fase de treinamento, consiste na realização de um treinamento a partir de uma cena de referência utilizada. Essa cena de referência contém o *background* que será usado de modelo para a segmentação posterior entre *foreground-background*. A segunda parte do método consiste na utilização da estrutura de dados criada na fase de treinamento para a detecção do *foreground* de um vídeo. Essa fase, que será chamada de fase de detecção, recebe um vídeo de entrada e gera uma máscara resultante, demarcando o que foi considerado *foreground* e o que foi considerado *background* na cena de entrada dada. É importante salientar que na cena de referência e na cena de entrada a câmera deve estar posicionada no mesmo lugar. Dessa forma, o algoritmo irá buscar uma correspondência do *background* conhecido, capturado na cena de referência, na cena de entrada.

Entre a fase de treinamento e a fase de detecção, tem-se o refinamento da estrutura de dados criada. Nessa parte algumas características que haviam sido armazenadas sobre alguns píxeis serão descartadas devido a baixa relevância dessas informações para o modelo de *background*. A figura 2.2 ilustra o funcionamento geral do algoritmo. A explicação de cada parte do algoritmo será detalhada nas subseções seguintes.

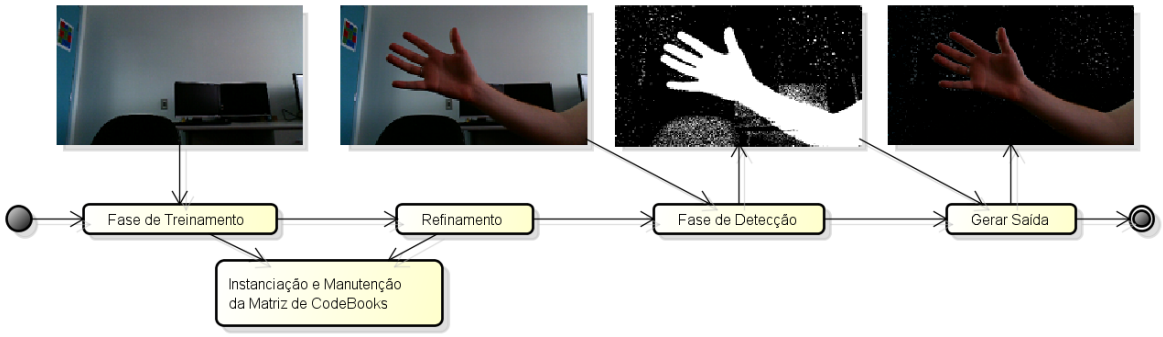


Figura 2.2: Passos do algoritmo *CodeBook Background Subtraction*.

Este algoritmo trabalha dentro do espaço de cor RGB, e a estrutura do modelo de *background* proposto consiste na criação de CodeBooks e CodeWords. Nesta abordagem, um CodeBook é construído para cada píxel, e cada CodeBook possui um ou mais CodeWords.

Os CodeBooks são representados como estruturas que guardam todas as características de um píxel em um certo intervalo de tempo, e estas caracteristicas são armazenadas nos CodeWords. Os CodeWords podem ser considerados como clusters arranjados dentro do espaço de cor correspondente (no caso, RGB), e possuem os seguintes valores:

- Um vetor de cores que armazena o valor RGB do píxel;
- Uma tupla contendo 6 valores para controle dos limiares de brilho e controle de frequência, sendo eles:
  - O valor máximo ($\hat{I}$) e mínimo ($\check{I}$) de brilho que o CodeWord abrange;
  - A frequência (f) da ocorrência deste CodeWord (ou seja, quantas vezes este CodeWord foi utilizado para classificar um píxel como *background*);
  - O maior intervalo (λ) em que o CodeWord não foi utilizado na fase de treinamento;
  - O número do frame do primeiro (p) e último (q) acesso do respectivo CodeWord.

Os valores de máximo e mínimo de brilho de um CodeWord são ajustados a cada novo píxel atribuído ou classificado a ele. As equações abaixo representam respectivamente os dois vetores (vetor RGB e tupla) armazenados em cada CodeWord:

$$RGB_i = (R_i, G_i, B_i)$$

$$tuple_i = \langle \check{I}_i, \hat{I}_i, f_i, \lambda_i, p_i, q_i \rangle$$

Estes valores compõem a estrutura de um CodeWord dentro de um CodeBook, podendo ser classificado como um cluster dentro do espaço RGB. A figura 2.3 ilustra como os CodeWords foram modelados dentro do espaço de cor RGB.

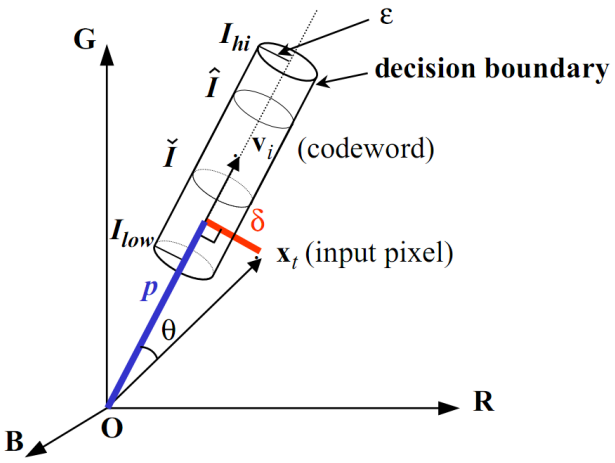


Figura 2.3: Representação do formato de um CodeWord construído (forma de um cilindro).

Um CodeWord pode detectar vários píxeis com valores de cor diferentes, dependendo da variação da imagem durante as observações, e são construídos a partir de sequências de imagens na fase de treinamento do algoritmo. É importante ressaltar que os CodeWords são armazenados em vetores e cada CodeBook possui um vetor de CodeWords. Os CodeBooks, por sua vez, correspondem às características de cada píxel, ou seja, há um CodeBook para cada píxel de um vídeo. Exemplificando, se tivermos um vídeo com dimensão de 640x480, teremos $640 * 480$ CodeBoooks.

Depois do treinamento realizado, a detecção do algoritmo é dada utilizando dois testes para cada píxel capturado. Um dos testes é referente à distorção de cor, o outro se refere à distorção de brilho. Para cada teste, os píxeis são ditos conhecidos ou não. Se for conhecido, então o píxel faz parte do background da cena, caso contrário, o píxel faz parte do foreground da cena.

2.3.1 Treinamento, Refinamento e Detecção

Como foi mencionado, as etapas do algoritmo *CodeBook Background Subtraction* podem ser divididas em: fase de treinamento (onde os CodeWords são construídos), refinamento (onde CodeWords pouco utilizados são descartados) e fase de detecção (na qual os clusters são utilizados para a detecção de foreground-background da cena). A detecção do *foreground* e *background* é realizada através de testes envolvendo a diferença, por píxel, de um frame com o

modelo de *background* criado na fase de treinamento. Nas subseções abaixo serão apresentadas as características de cada uma das fases do algoritmo *CodeBook Background Subtraction*.

2.3.1.1 Algoritmo de Treinamento

A fase de treinamento realiza a construção da matriz de CodeBooks e os respectivos CodeWords de cada CodeBook. A ideia consiste em agrupar o maior número de píxeis no menor número possível de CodeWords, visando o menor uso possivel de memória. Além disso, quanto menor o número de CodeWords em um CodeBook, mais rápida será a classificação de cada novo píxel em *foreground* e *backgorund*, visto que a quantidade de CodeWords que deverão ser testados será menor.

Para cada píxel novo na fase de treinamento, os CodeWords já criados são consultados, para verificar se algum destes é correspondente com um novo píxel de entrada. Essa correspondência é realizada através de funções de distorção de cor e de brilho. O treinamento para construção de um CodeBook para um píxel é ilustrado no algoritmo 1. As funções de distorção de cor e brilho serão apresentadas nas subseções seguintes.

Algoritmo 1 Algoritmo de treinamento: Construção de um CodeBook (para um píxel)

Require: Tendo L um inteiro e ϑ um vetor de CodeWords.
$L = 0, \vartheta = (vazio)$
for $t = 1 \rightarrow N$ **do**
$x_t = (R_t, G_t, B_t), I = \sqrt[2]{R_t^2 + G_t^2 + B_t^2}$
Procurar um CodeWord c_m em $\vartheta = \{c_i | 1 \leqslant i \leqslant L\}$ que corresponda/combine com o novo píxel capturado x_t baseado nas condições:
(a)*colordist*$(x_t, RGB_m) \leqslant \epsilon_1$
(b)*brightness*$(I, \langle \check{I}, \hat{I} \rangle) = true$
if $c_m! = null$ **then**
Atualiza o CodeWord encontrado
$RGB_m = (R_m, G_m, B_m)$ e $tuple_m = \langle \check{I}_m, \hat{I}_m, f_m, \lambda_m, p_m, q_m \rangle$
$RGB_m = (\frac{f_m R_m + R_t}{f_m + 1}, \frac{f_m G_m + G_t}{f_m + 1}, \frac{f_m B_m + B_t}{f_m + 1})$
$tuple_m = \langle min\{I, \check{I}_m\}, max\{I, \hat{I}_m\}, f_m + 1, max\{\lambda_m, t - q_m\}, p_m, t \rangle$
else
$L = L + 1$
$RGB_L = (R_t, G_t, B_t)$
$tuple_L = \langle I, I, 1, t - 1, t, t \rangle$
end if
end for
Para cada codeword c_i, tendo $i = 1, ..., L$, atualize os valores de cada λ_i de cada CodeWord criado para $\lambda_i = max\{\lambda_i, (N - q_i + p_i - 1)\}$.

Ao final da fase de treinamento, é criada uma estrutura de dados contendo os píxeis

obtidos da cena de referência da fase de treinamento, resultando em uma matriz de CodeBooks.

Cada CodeBook corresponde a um píxel da tela treinado e são avaliados separadamente. Dessa forma, o comportamente de um píxel não afeta os resultados de detecção com qualquer outro píxel.

Como na cena de referência um píxel pode alcançar valores distintos, um CodeBook pode conter vários CodeWords, armazenando todos os valores de cor do píxel registrados na fase de treinamento.

2.3.1.2 Refinamento dos CodeBooks

Em alguns vídeos, pode ocorrer a aparição de objetos não estáticos na cena de treinamento por alguns frames, não correspondento a um elemento estático ou a algum evento periódico do *background*. Logo, é melhor retirar o CodeWord que foi registrado com o valor dos píxeis deste elemento para que não sejam guardadas informações incorretas a respeito do *background*, para desempenho de reconhecimento e, como consequência, para diminuir o uso total de memória.

A variável que guarda o maior intervalo em que um codeword não foi utilizado durante o treinamento, denotada por λ e pertencente a tupla de um CodeWord, tem como objetivo verificar quais CodeWords podem ser descartados da estrutura de dados final gerada. Dessa forma, pequenas aparições de novos elementos em um vídeo ou objetos que não seguem um padrão de movimento durante o treinamento são posteriormente descartados.

A equação de refinamento é dada através da medição de um limiar, que habitualmente é atribuído a ele a metade do número de frames treinados. Este limiar é denotado pela variável T_μ, e a decisão de deletar ou não um CodeWord de um CodeBook é dada a partir da comparação entre o valor do limiar T_μ e o valor da variável de maior intervalo em que um respectivo CodeWord não foi utilizado (λ). Sendo assim, os CodeWords que possuirem um valor alto de λ_m serão eliminados do CodeBook correspondente. O valor de λ é calculado em número de frames.

Logo, tendo N como o número de frames treinados, temos que:

$$T_\mu = \frac{N}{2}$$

$$\mu = \{c_m | c_m \in \vartheta \wedge \lambda_m \leqslant T_\mu\}$$

Desta forma, esta equação percorre todos os CodeWords de cada CodeBook e verifica

quais deles não possuem o valor de λ_m menor que o limiar T_μ. Estes serão deletados da estrutura ao final do treinamento.

Este refinamento é importante para não gerar uma estrutura de dados muito grande, onde muitos CodeWords que foram criados em algum frame do treinamento nunca voltarão a ser utilizados ou são criados a partir de um elemento não estático que não deve fazer parte do *background*, resultando apenas em um gasto de memória. Guardar CodeWords que não ocorrem numa frequência considerável numa cena prejudicaria a eficácia do algoritmo para testes com períodos longos de treinamento. A fase de refinamento do algoritmo ocorre posteriormente a fase de treinamento e antes da fase de detecção.

2.3.1.3 Algoritmo de Detecção

A fase de detecção consiste em capturar os píxeis de uma cena de entrada e calcular a distância entre cada píxel de entrada do modelo de *background* construído no treinamento, representado pelos CodeWords pertencentes ao CodeBook correspondente deste píxel. Os passos do algoritmo de detecção são ilustrados no algoritmo 2.

Algoritmo 2 Algoritmo de detecção: Segmentação em *Foreground* e *Background*(para um píxel)

Require: CodeBooks construídos.
while há um novo frame para ser capturado da cena de entrada **do**
$x = (R, G, B), I = \sqrt[2]{R^2 + G^2 + B^2}$
Procurar um CodeWord c_m em ϑ que corresponda/combine com o novo píxel capturado x baseado nas condições:
(a)*colordist*$(x, RGB_m) \leqslant \epsilon_2$
(b)*brightness*$(I, \langle \check{I}_m, \hat{I}_m \rangle) = true$
if $c_m! = null$ **then**
Background Detectado
Atualiza o CodeWord encontrado (mesmo processo do algoritmo de treinamento)
$RGB_m = (R_m, G_m, B_m) etuple_m = \langle \check{I}_m, \hat{I}_m, f_m, \lambda_m, p_m, q_m \rangle$
$RGB_m = (\frac{f_m R_m + R}{f_m + 1}, \frac{f_m G_m + G}{f_m + 1}, \frac{f_m B_m + B}{f_m + 1})$
$tuple_m = \langle min\{I, \check{I}_m\}, max\{I, \hat{I}_m\}, f_m + 1, max\{\lambda_m, t - q_m\}, p_m, t \rangle$
else
Foreground Detectado
end if
end while

Nessa fase, o algoritmo faz o uso de uma cena de entrada e, a partir do treinamento realizado, procura classificar os píxeis em *foreground* e *background*. Ao final, é gerada uma máscara onde os píxeis brancos correspondem ao *foreground* e os píxeis pretos correspondem

ao *background*, de acordo com a figura 2.4.

(a) (b) (c)

Figura 2.4: (a) Cena de Referência, (b) Cena de entrada e (c) máscara gerada da fase de detecção.

2.3.2 Distorção de cor e brilho

As funções de distorção de cor e brilho, utilizadas no cálculo de distância entre píxeis e CodeWords, foram propostas por Kyungnam Kim como uma maneira de avaliar as diferenças de cor e brilho de um píxel de maneira independente. Este modelo foi proposto porque algoritmos que geralmente utilizam a normalização de cores para os cálculos de distância entre píxeis, não geram um bom resultado em segmentação de *foreground* e *background* ao trabalhar com cenas que apresentam áreas escuras (KIM et al., 2005). Nas subseções seguintes serão apresentadas as características das funções de distorção de cor e distorção de brilho.

2.3.2.1 Distorção de cor

Para o cálculo de distorção de cor, o píxel contido no CodeWord é primeiramente reescalado para o mesmo valor de brilho do píxel de entrada e, em seguida, é medida a distância entre os dois pontos. Partindo da figura 2.3 onde tem-se a ilustração de um CodeWord, o valor de v_i é reescalado no gráfico e alinhado ao eixo do CodeWord e em uma distância *p* da origem, para ter o mesmo valor de brilho de x_t. Assim, tendo a reta em azul p, pode-se calcular a distância δ, que está ilustrada em vermelho. Dessa forma, tem-se a distorção de cor entre dois píxeis. A distorção de cor é calculada da seguinte forma:

- Tendo um píxel de entrada e um codeword com valores RGB:

$$||x_t||^2 = R^2 + G^2 + B^2$$

$$||v_t||^2 = R_i^2 + G_i^2 + B_i^2$$

$$\langle x_t, v_i \rangle^2 = (RR_i + GG_i + BB_i)^2$$

- O valor da distorção de cor é dado por:

$$p^2 = ||x_t||^2 \cos^2(\theta) = \frac{\langle x_t, v_i \rangle^2}{||v_t||^2}$$

$$\textit{colordist}(x_t, v_i) = \delta = \sqrt[2]{||x_t||^2 - p^2}$$

O valor da distorção de cor é relacionada à distância entre os valores do píxel e do CodeWord. A figura 2.3 ilustra o valor de δ , que se refere a distorção de cor e também ilustra todos os outros elementos utilizados nas equações construídas para verificar a distorção de cor entre um píxel e um CodeWord. Sendo assim, para verificar se um píxel faz parte de um CodeWord, deve-se encontrar um limiar que determine a maior distância aceitável entre um píxel de um CodeWord para que este possa ser considerado parte do mesmo pela distância resultante da distorção de cor.

2.3.2.2 Distorção de brilho

Para possibilitar a detecção de mudanças de brilho sem prejudicar segmentação final em *foreground* e *background* são armazenados valores de brilho mínimo e máximo em um CodeWord. Estes valores representam o brilho máximo e mínimo dos píxeis que foram classificado em um CodeWord. Para calcular o valor de distorção de brilho entre um píxel e um codeword são determinados dois limiares de brilho máximo e mínimo:

$$I_{low} = \alpha \hat{I}$$

$$I_{high} = min\{\beta \hat{I}, \frac{\check{I}}{\alpha}\}$$

Dessa forma, tendo x_t como um píxel de entrada e os dois limiares I_{low} e I_{high}, podemos determinar que a função de distorção de brilho é dada por:

$$brightness(I, \langle \check{I}, \hat{I} \rangle) = \begin{cases} Verdadeiro & \text{se } I_{low} \leqslant ||x_t|| \leqslant I_{high}, \\ Falso & \text{caso contrário.} \end{cases}$$

Os valores de α e β são constantes auxiliares para atribuir uma margem um pouco maior para os valores máximo de mínimo de brillho dos CodeWords, aumentando o intervalo de detecção de píxeis. A variável $||x_t||$ representa o valor de brilho do píxel que está sendo testado.

2.4 Espaços de cor

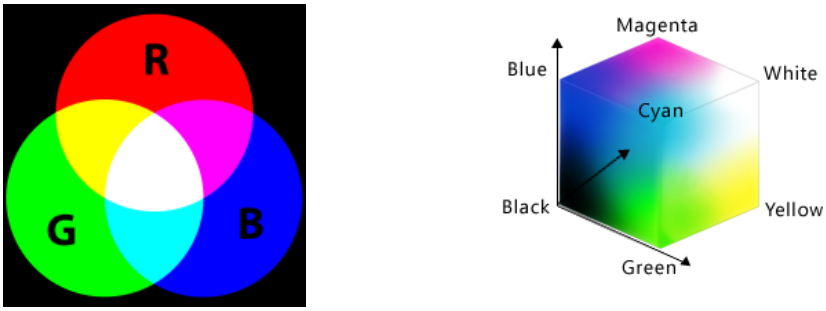
Os espaços de cor são modelos matemáticos que determinam como as cores são representadas, através da combinação de componentes. Então, em uma imagem colorida, cada píxel pode ser representado através das componentes de um espaço de cor.

Para este trabalho, serão estudados os seguintes espaços de cor: RGB, LAB, HSV, HLS e YCrCb. Cada espaço de cor possui características diferentes. Então, dependendo do espaço de cor utilizado, haverá maneiras diferentes de representar as cores.

2.4.1 Espaço de cor RGB

O espaço de cor RGB é um modelo de cor aditivo formato pelas primitivas Vermelho, Verde e Azul. Este espaço de cor é o mais utilizado entre os espaços de cores existentes e tem como qualidade a possibilidade de explorar combinações diferentes de três cores com três bandas. Desta forma, é possivel obter uma imagem colorida de melhor contraste, visto que é possivel manipular estas três diferentes bandas representando cada uma das cores primitivas (MENESES et al., 2012).

A representação do espaço de cor RGB é dada através de um cubo (MENESES et al., 2012), e ilustrada na figura 2.5. Esta representação se localiza em um espaço tridimensional, onde os vértices serão representados pelas cores: preto, verde, amarelo, vermelho, azul, turquesa, branco e magenta.



(a) (b)

Figura 2.5: Representação do espaço de cor RGB.

2.4.2 Espaço de cor LAB

Derivada do modelo padrão XYZ, o modelo LAB é composto pela luminância e dois canais de cores ‘a ’e ‘b ’. O canal ‘a ’corresponde nas extremidades às cores verde e vermelha. O canal ‘b ’corresponde nas extremidades às cores azul e amarela. Podem-se chamar esses dois canais de componentes cromáticas. A figura 2.6 ilustra a representação do espaço de cor LAB.

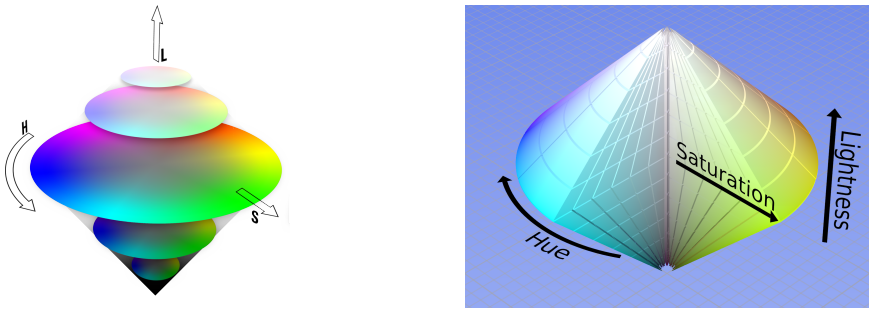
(a) (b)

Figura 2.6: Representação do espaço de cor LAB

2.4.3 Espaço de cor HLS

O modelo HLS é composto pelos parâmetros de matiz, luminosidade e saturação, definindo respectivamente a cor do ponto, a luminosidade e a diferença da cor em relação à cor cinza.

A representação gráfica desse modelo é um cone duplo, onde os vértices as extremidades verticais representam as cores branca e preta. Este sistema permite a classificação das cores entre as mais claras e as mais escuras. A figura 2.7 ilustra a representação gráfica deste espaço de cor.



(a) (b)

Figura 2.7: Representacao do espaco de cor HLS

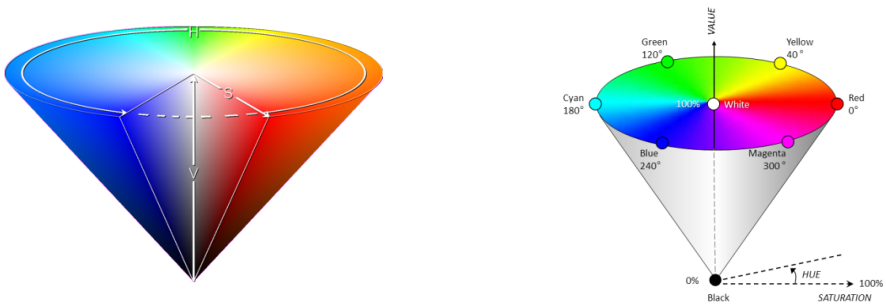
Neste sistema, as cores são obtidas a partir de um valor de ângulo da matiz, combinados com um valor de luminosidade e um valor de saturação. Este sistema descreve a percepção de forma mais acurada que no sistema RGB, uma vez que este modelo se aproxima mais da percepção de cor do ser humano (classificação em cores mais claras e mais escuras).

2.4.4 Espaço de cor HSV

O modelo HSV é composto pelos parâmetros de matiz, saturação e brilho, definindo respectivamente a cor do ponto, o quanto a respectiva cor se difere da cor cinza e a quantidade de brilho que o píxel contém.

No espaço HSV é possível manipular cada componente separadamente, obtendo dessa forma, maior controle sobre o realce de cor, através de deslocamento e operações lineares e não lineares sobre os três componentes de cor (MENESES et al., 2012).

A representação gráfica desse modelo é um cone, onde o vértice da extremidade vertical representa a cor preta, de acordo com a figura 2.8.



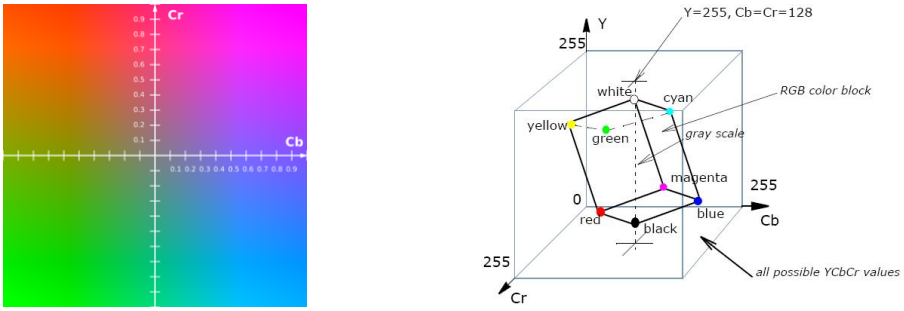
(a) (b)

Figura 2.8: Representacao do espaco de cor HSV

Este sistema tem uma semelhança muito grande com o modelo HLS, pois também baseia-se na percepção humana da cor. Porém, neste modelo não há um ponto de divisão entre cores claras e escuras. Tanto o modelo HLS quanto o HSV são modelos muito mais intuitivos do que o modelo RGB, visto que é fácil manusear as cores a partir de tons e sombras ao invés da adição de vermelho, verde e azul.

2.4.5 Espaço de cor YCrCb

O espaço de cor YCrCb é composto pelos parâmetros de luminância de um píxel e suas respectivas crominâncias de azul e vermelho. Este espaço de cor é muito utilizado para sinais de vídeos para televisores. A figura 2.9 ilustra a representação deste espaço de cor e a relação entre as suas respectivas crominâncias de azul e vermelho.



(a) (b)

Figura 2.9: Representacao do espaco de cor YCbCr

2.5 Funções de pós-processamento

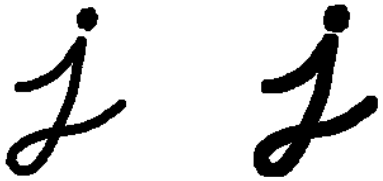
As subseções a seguir abordam algumas funções de processamento de imagens que serão utilizadas para refinar os resultados gerados pelo algoritmo *CodeBook Background Subtraction*. Estas subseções estão organizadas em: filtros morfológicos, detecção de bordas, equalização de histogramas e preenchimento de buracos.

2.5.1 Filtros morfológicos

Os filtros morfológicos exploram as propriedades geométricas dos sinais. Para filtros morfológicos, as máscaras utilizadas são denominadas elementos estruturantes e apresentam valores 0 ou 1 na matriz aplicada nos píxeis considerados. Os filtros básicos são: o filtro da mediana, erosão e dilatação (INPE, 2012). Para este trabalho, foram estudados os filtros de erosão e dilatação, além dos filtros provenientes da combinação destes dois filtros.

As características dos filtros de erosão e dilatação são:

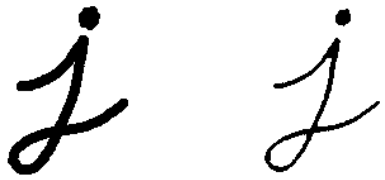
- Filtro morfológico de erosão: provoca efeitos de erosão das partes claras da imagem (e altos níveis de cinza), gerando imagens mais escuras, de acordo com a figura 2.10.



(a) (b)

Figura 2.10: (a) Imagem original e (b) imagem após a aplicação do filtro de erosão.

- Filtro morfológico de dilatação: provoca efeitos de dilatação das partes claras da imagem (e baixos níveis de cinza), gerando imagens com uma maior quantidade de píxeis com altos níveis de cinza, como na figura 2.11.

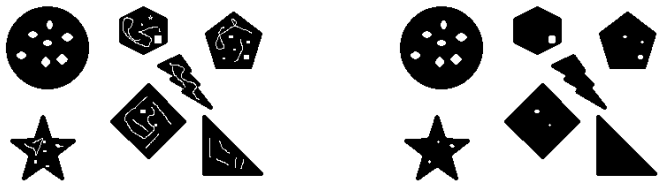


(a) (b)

Figura 2.11: (a) Imagem original e (b) imagem após a aplicação do filtro de dilatação.

Além dos filtros básicos, outros filtros morfológicos podem ser gerados a partir da combinação de dois filtros morfológicos, sendo eles:

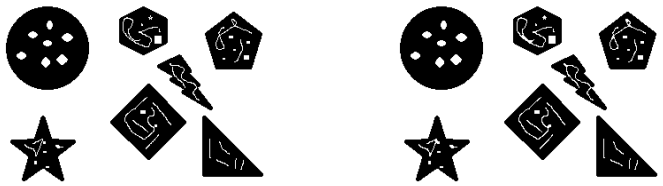
- Filtro de abertura: consiste na combinação de uma erosão seguida de uma dilatação, ilustrado na figura 2.12.



(a) (b)

Figura 2.12: (a) Imagem original e (b) imagem após a aplicação do filtro de abertura.

- Filtro de fechamento: consiste na combinação de uma diltação seguida de uma erosão, representado na figura 2.13.



(a) (b)

Figura 2.13: (a) Imagem original e (b) imagem após a aplicação do filtro de fechamento.

2.5.2 Reconhecimento de bordas

O reconhecimento de bordas em imagens é uma das operações mais utilizadas em processamento de imagens e de grande importância na área de visão computacional. As bordas definem os limites entre regiões de uma imagem, que auxiliam na segmentação e reconhecimento de objetos (NADERNEJAD; SHARIFZADEH; HASSANPOUR, 2008). Com o reconhecimento de bordas, é possível identificar os limites entre os objetos contidos em uma imagem.

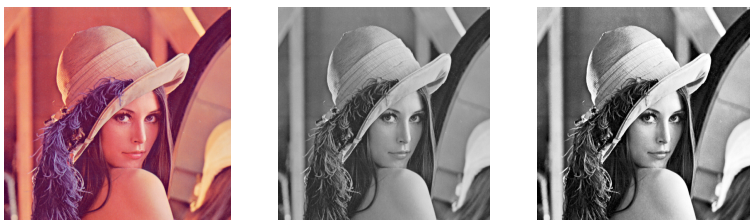
O algoritmo mais utilizado para o reconhecimento de bordas é chamado de Canny Edge Detection (CANNY, 1986). A figura 2.14 demonstra o funcionamento deste método.

(a) (b) (c)

Figura 2.14: (a) Imagem original, (b) imagem em tons de cinza e (c) imagem resultante do algoritmo de detecção de bordas.

2.5.3 Equalização de histogramas

A equalização de histogramas tem como objetivo expandir o contraste de uma imagem. Consequentemente, aumenta o alcance e a variação de intensidade entre píxeis vizinhos (GARG; MITTAL; GARG, 2011; OPENCV DOCUMENTATION, 2012). A figura 2.15 demonstra o aumento do contraste em uma imagem em tons de cinza através da aplicação da técnica de equalização de histogramas.

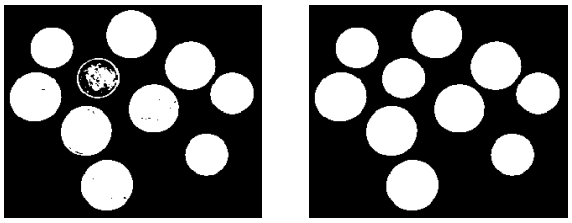


(a) (b) (c)

Figura 2.15: (a) Imagem original, (b) imagem em tons de cinza e (c) imagem em tons de cinza após a aplicação do método de equalização de histogramas.

2.5.4 Preenchimento de buracos

O preenchimento de buracos tem por objetivo tapar os "buracos"encontrados em uma região da imagem. No exemplo dado pela figura 2.16, são preenchidas as regiões de cor preta contornadas por regiões de cor branca.



(a) (b)

Figura 2.16: (a) Imagem original, (b) imagem após a aplicação do Preenchimento de buracos.

3 MÉTODOS E TÉCNICAS

A pesquisa desse trabalho tem um teor experimental e exploratória. Exploratória, pois são explorados os espaços de cor existentes e funções de pós-processamento com o objetivo de melhorar os resultados gerados inicialmente por um algoritmo escolhido como base da aplicação. Experimental, pois são realizados testes com diferentes abordagens visando encontrar a melhor solução para o problema. A eficácia de cada abordagem é comprovada a partir das taxas de acerto e erro na detecção correta de *foreground-background*. Ao final, é realizado um comparativo dos resultados gerados de cada implementação.

3.1 Material utilizado

Como material para a realização dos experimentos, são utilizados vídeos capturados de uma câmera parada, podendo ou não apresentar variações de iluminação no ambiente durante a filmagem. Para cada experimento feito, são gravados dois vídeos: o primeira é gravado com o objetivo apenas de capturar o *background* do ambiente, e o segundo é gravado com o objetivo de detectar as regiões deste vídeo que não estão de acordo com as cores observadas no primeiro vídeo.

3.2 Algoritmo utilizado

Pretende-se utilizar o algoritmo *CodeBook Background Subtraction* (KIM et al., 2005) para realizar a detecção do *foreground* e *background* dos vídeos. Este algoritmo foi escolhido pois se mostrou bastante robusto para a detecção de múltiplos *backgrounds* em ambientes com iluminação variante. Sendo assim, utilizando este algoritmo, pode-se inferir que são gerados bons resultados e, por isso, este algoritmo é utilizado como base desse trabalho.

Mesmo com a geração de bons resultados, foi observado no próprio artigo onde foi proposto o algoritmo (KIM et al., 2005) a presença contínua de ruídos e detecções parciais de elementos. A figura 3.1 ilustra o problema da ocorrência de ruídos em uma cena e detecções parciais de *foreground*.

Sendo assim, pretende-se utilizar funções de pós-processamento e portar o algoritmo *CBS* (*CodeBook Background Subtraction*) para outros espaços de cor (além do RGB que é utilizado no algoritmo base), com o objetivo de realizar uma comparação dos resultados gerados

(a) (b) (c)

Figura 3.1: (a) Cena de Referência, (b) Cena de entrada e (c) *Foreground* da cena com a presença de ruídos e detecções parciais.

em cada abordagem.

3.3 Fórmulas para calcular a diferença entre cores

Os diferentes espaços de cor existentes possuem características diferentes e, com isso, a classificação dos píxeis e a relação entre eles também possui um comportamento diferente. O estudo de outros espaços de cor e a implementação do algoritmo inicial para os diferentes espaços de cor escolhidos tem por objetivo verificar se é possível obter um melhor resultado utilizando um espaço de cor diferente do RGB.

Os espaços de cor escolhidos foram: LAB, HLS, HSV e YCbCr. Para cada uma destes espaços de cor escolhidos, foi proposto um cálculo de distância entre píxeis para conseguir classificá-los em *foreground* e *background* de acordo com a modelagem e posicionamento das cores nos diferentes espaços de cor, sendo estes variantes por causa das diferentes representações que cada um apresenta (como foi visto na seção 2.4). Os cálculos escolhidos para cada espaço de cor foram os seguintes:

- **LAB**: Para este espaço de cor foi utilizada a fórmula CIEDE2000 (LUO; CUI; RIGG, 2001). Ela foi utilizada pois possui algumas melhorias em relação a fórmula CIE94, que por sua vez é uma versão melhorada da fórmula CIE76.

  Tendo dois píxeis no espaço LAB $(L*_1, a*_1, b*_1)$ e $(L*_2, a*_2, b*_2)$ juntamente com 3 constantes $k_L, k_C e k_H$, a fórmula da diferença de cor CIEDE2000 ΔE_{00} pode ser dividida em três principais passos, de acordo com Sharma (SHARMA; DALAL, 2005):

  1. Calcular C'_1, C'_2, h'_1 e h'_2:

$$C*_1 = \sqrt{(a*_1)^2 + (b*_1)^2} \qquad C*_2 = \sqrt{(a*_2)^2 + (b*_2)^2}$$

$$\overline{C*} = \frac{C*_1 + C*_2}{2}$$

$$G = 0.5(1 - \sqrt{\frac{\overline{C*}^7}{\overline{C*}^7 + 25^7}})$$

$$a'_1 = (1+G)a*_1 \qquad a'_2 = (1+G)a*_2$$

$$C'_1 = \sqrt{(a'_1)^2 + (b*_1)^2} \qquad C'_2 = \sqrt{(a'_2)^2 + (b*_2)^2}$$

$$h'_1 = \begin{cases} 0 & b*_1 = a'_1 = 0 \\ tan^{-1}(b*_1, a'_1) & \textit{caso contrário} \end{cases} \qquad h'_2 = \begin{cases} 0 & b*_2 = a'_2 = 0 \\ tan^{-1}(b*_2, a'_2) & \textit{caso contrário} \end{cases}$$

2. Calcular $\Delta L'$, $\Delta C'$ e $\Delta H'$

$$\Delta L' = L*_2 - L*_1 \qquad \Delta C' = C'_2 - C'_1$$

$$h'_2 = \begin{cases} 0 & C'_1 C'_2 = 0 \\ h'_2 - h'_1 & C'_1 C'_2 \neq 0; |h'_2 - h'_1| \leq 180^o \\ (h'_2 - h'_1) - 360 & C'_1 C'_2 \neq 0; (h'_2 - h'_1) > 180^o \\ (h'_2 - h'_1) + 360 & C'_1 C'_2 \neq 0; (h'_2 - h'_1) < -180^o \end{cases}$$

$$\Delta H' = 2\sqrt{C'_1 C'_2} \sin \frac{\Delta h'}{2}$$

3. Calcular a diferença de cor CIEDE2000 denotada por ΔE_{00}:

$$\overline{L'} = (L*_2 + L*_1)/2 \qquad \overline{C'} = (C'_2 + C'_1)/2$$

$$\overline{h'} = \begin{cases} \frac{h'_1 + h'_2}{2} & |h'_1 - h'_2| \leq 180^o; C'_1 C'_2 \neq 0 \\ \frac{h'_1 + h'_2 + 360^o}{2} & |h'_1 - h'_2| > 180^o; (h'_1 + h'_2) < 360^o \\ \frac{h'_1 + h'_2 - 360^o}{2} & |h'_1 - h'_2| > 180^o; (h'_1 + h'_2) \geq 360^o \\ (h'_1 + h'_2) & C'_1 C'_2 = 0 \end{cases}$$

$$T = 1 - 0.17\cos(\overline{h'} - 30^o) + 0.24\cos(2\overline{h'}) + 0.32\cos(3\overline{h'} + 6^o) - 0.20\cos(4\overline{h'} - 63^o)$$

$$\Delta\theta = 30\exp\{-[\frac{\overline{h'} - 275^o}{25}]^2\} \qquad R_C = 2\sqrt{\frac{\overline{C'}^7}{\overline{C'}^7 + 25^7}}$$

$$S_L = 1 + \frac{0.015(\overline{L'} - 50)^2}{\sqrt{20 + (\overline{L'} - 50)^2}} \qquad S_C = 1 + 0.045\overline{C'}$$

$$S_H = 1 + 0.015\overline{C'}T \qquad R_T = -\sin(2\Delta\theta)R_C$$

$$\Delta E_{00} = \sqrt{(\frac{\Delta L'}{k_L S_L})^2 + (\frac{\Delta C'}{k_C S_C})^2 + (\frac{\Delta H'}{k_H S_H})^2 + R_T(\frac{\Delta C'}{k_C S_C})(\frac{\Delta H'}{k_H S_H})}$$

Os valores de (k_L, k_C, k_H) foram ajustados para $(1.0, 0.8, 0.8)$. Cada um desses valores modifica a influência de cada atributo (L, a, b) da cor no cálculo final da distância.

- **HLS**: para o cálculo de distância entre píxeis dentro do espaço de cor HLS foi utilizada a fórmula proposta por Grant (GRANT; CLARK; GREEN, 2008). A técnica em si é

utilizada para a detecção de bordas, porém como este é um cálculo de distância entre píxeis, foi possível utilizar essa fórmula para o propósito desse trabalho.

Tendo dois píxeis no espaço de cor HLS (h, l, s) e (h', l', s'), a distância entre duas cores no espaço HLS é calculada da seguinte forma:

$$l_{min} = \frac{min(128-|128-l|,128-|128-l'|)}{128} \qquad s_{min} = \frac{min(s,s')}{256}$$

$$d_h = min(|h-h'|, 360-|h-h'|)l_{min}s_{min}$$

$$d_l = l - l' \qquad d_s = (s-s')s_{min}$$

$$\Delta HLS_{hls} = \sqrt{d_h^2 + d_l^2 + d_s^2}$$

- **HSV**: para o cálculo de distância entre píxeis dentro do espaço de cor HSV foi utilizada a fórmula que calcula a distância euclidiana entre duas cores (FISHER, 1999). Dentre os cálculos de distância apresentados, foi escolhida a fórmula de distância euclidiana que utiliza as primitivas *Hue* e *Saturation* e a diferença entre píxeis da primitiva *Value* é calculada separadamente. A distância euclidiana utilizando apenas as primitivas *Hue* e *Saturation* representam a avaliação da cor independente do brilho. A distorção de brilho é calculada através da diferença da primitiva *Value* entre duas cores. Tendo dois píxeis no espaço de cor HSV (h, s, v) e (h', s', v'), a distância euclidiana entre duas cores nesse espaço de cor é calculada da seguinte forma:

$$\Delta HSV_{hs} = ||(scos(2\pi h), ssin(2\pi h)) - (s'cos(2\pi h'), s'sin(2\pi h'))||$$

$$\Delta HSV_v = |v - v'|$$

- **YCbCr**: dentro do espaço de cor YCbCr foi inicialmente testada a fórmula de distorção de cor utilizando a distância euclidiana (Kim et al., 2001). Porém, visto que os resultados utilizando esta fórmula se mostraram muito errôneos, foi verificada a possibilidade de avaliar a distância entre cores de maneira semelhante com o modelo de cor proposto por Kim para o espaço de cor RGB (KIM et al., 2005). Considerando que as cores no espaço de cor YCbCr já possuem dois atributos de crominância que são independentes do brilho, foi escolhido calcular a distância das cores em duas partes: calcular a distorção de tonalidade utilizando os atributos de crominância de azul e vermelho (Cb e Cr) e calcular a distorção de brilho através da diferença de Luminância (Y).

Tendo dois píxeis no espaço de cor YCbCr (Y, Cb, Cr) e (Y', Cb', Cr'), a distância euclidiana entre duas cores foi proposta da seguinte forma:

$$\Delta YCbCr_{CbCr} = ||(Cb, Cr) - (Cb', Cr')||$$

$$\Delta YCbCr_{Y} = |Y - Y'|$$

Uma das grandes vantagens de usar esta abordagem (calcular distorção de tonalidade e brilho) é poder trabalhar com dois valores de limiar para as distâncias: um para a distorção de cor referente à tonalidade das mesmas e outro para a distorção de brilho, possibilitando controlar o valor máximo de erro desejado de maneira independente, ou seja, para cada uma das duas distâncias terá um valor diferença de erro (distância máxima).

3.4 Funções de pós-processamento

A utilização de algoritmos de processamento de imagens, como equalização de histogramas, reconhecimento de bordas e operadores morfológicos, tem como propósito refinar a saída gerada, tentando eliminar os problemas observados na implementação inicial (ruídos e detecções parciais). Os algoritmos utilizados com o propósito de refinar a saída gerada foram: filtro morfológico de abertura, equalização de histogramas, reconhecimento de bordas e preenchimento de buracos. A figura 3.2 ilustra a imagem final almejada após a aplicação das funções de pós processamento.

(a) (b)

Figura 3.2: (a) Imagem gerada da fase de detecção e (b) imagem almejada após aplicadas as funções de pós processamento.

O filtro morfológico de abertura é utilizado para a remoção de ruído, ou seja, a remoção de pequenos píxeis presentes na imagem resultante que não estão de acordo com o resultado esperado. Dessa forma, o problema da presença de ruídos presentes no *background* da saída gerada na fase de detecção deverá ser solucionado. As funções de equalização de histogramas,

reconhecimento de bordas e preenchimento de buracos são utilizadas para buscar uma melhoria no problema de detecções parciais do *foreground*.

Após a implementação das abordagens propostas, são realizados testes com cada uma delas, e os resultados são comparados para verificar qual estratégia gerou melhores resultados.

3.5 Arquitetura

O algoritmo desenvolvido contém passos de treinamento, refinamento dos CodeBooks, detecção e refinamento do frame gerado. A figura 3.3 representa o diagrama de atividades da arquitetura proposta.

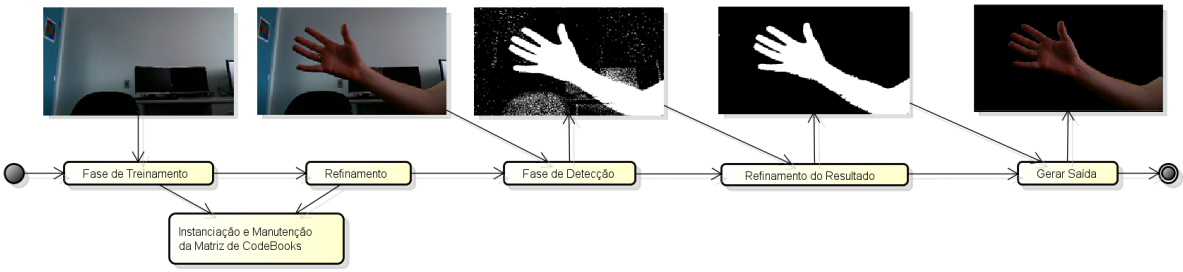


Figura 3.3: Passos da abordagem proposta.

A estrutura de classes é composta por um gerenciador que possui uma matriz de codebooks, onde cada codebook possui um vetor de codewords. A figura 3.4 representa o diagrama de classe proposto para a implementação geral do algoritmo.

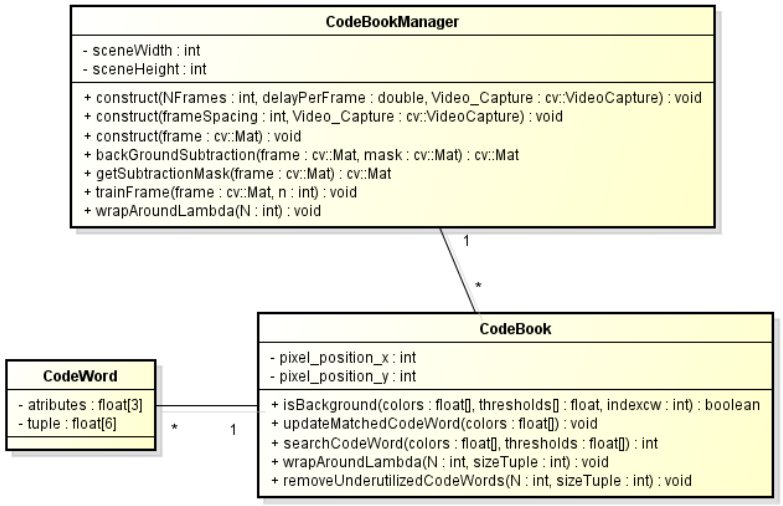


Figura 3.4: Diagrama de Classe da implementação do algoritmo *CodeBook Background Subtraction*.

A classe CodeBookManager foi criada para realizar toda a manipulação dos CodeBooks,

com o objetivo de abstrair todas as funções de treinamento, refinamento e detecção. A classe CodeBook foi criada para guardar os valores dos CodeWords, abstrair as funções de diferença entre cores e manipulação dos CodeWords guardados, além das funções para classificação de píxeis em conhecido ou desconhecido. A classe CodeWord contém dois vetores, sendo um para guardar valores de cor e outro para os valores de controle de brilho e frequência (como mencionado na seção 2.3 que trata o algoritmo *CodeBook Background Subtraction*).

O refinamento realizado antes de gerar a saída foi adicionado ao algoritmo após a fase de detecção. Neste passo são utilizadas funções de processamento de imagens para retirar os possíveis erros das máscaras geradas pelo algoritmo. Visando a implementação e evitando a necessidade de reimplementação das funções de pós-processamento em cada abordagem proposta pelos diferentes espaços de cor escolhidos, foi desenvolvida uma biblioteca contendo todas as funções utilizadas.

Para a manipulação de imagens e captura de vídeos, foi utilizada a biblioteca OpenCV (Open Source Computer Vision). Ela contém funções auxiliares para visão computacional em tempo real (OPENCV - OPEN SOURCE COMPUTER VISION, 2012). Ela possui funções que auxiliam na análise e processamento de imagens. Além disso, possui módulos que abstraem toda a parte de captura, leitura e escrita de dados, sendo estes de imagens, vídeos ou câmeras.

Esta biblioteca possui seu próprio gerenciamento de memória, tanto para leitura de vídeos ou imagens, quanto pra gerar arquivos de saída. Essa gerência é realizada através de contadores que verificam a quantidade de ocorrências para um certo objeto criado (AUTOMATIC MEMORY MANAGEMENT, OPENCV DOCUMENTATION, 2012). Quando esse número chegar a zero, o destrutor do objeto é chamado, desalocando a memória utilizada. Dessa forma, não é necessário que o programador se preocupe com gerenciamento de memória.

O OpenCV possui, por padrão, funções de processamento de imagens como mudança de espaço de cor e operadores morfológicos, que foram utilizadas nesse trabalho.

4 IMPLEMENTAÇÃO

Neste capítulo são discutidos os aspectos da implementação desenvolvida. Dessa forma, as seções estão divididas de acordo com as classes construídas: CodeWord, CodeBook e CodeBookManager. Além disso, a seção 4.5 relata as funções de pós processamento que foram implementadas para o trabalho.

Para cada seção serão explicitadas as funções que foram utilizadas em cada fase do algoritmo, sendo elas: fase de treinamento, fase de refinamento e fase de detecção. Todas as funções descritas nesse capítulo são apresentadas na seção de anexos.

4.1 CodeWord

A classe CodeWord, como foi dito no capítulo 3, contém dois vetores, um para guardar valores de cor e outro para controle de brilho e frequência. As funções implementadas nessa classe foram apenas para facilitar a manipulação dos dados. O código 4.1 demonstra o escopo da classe CodeWord.

```
#define MAX_SIZE_ATRIBUTES 3
#define MAX_SIZE_TUPLE 6

class CodeWord
{
public:
    CodeWord();
    ~CodeWord();
    void setAtributes(float c[], int n);
    void setAtributesValueAt(float value, int index);
    void setTuple(float t[], int n);
    void setTupleValueAt(float value, int index);
    float* getAtributes();
    float getAtributesValueAt(int index);
    float* getTuple();
    float getTupleValueAt(int index);

    float atributes[MAX_SIZE_ATRIBUTES];
    float tuple[MAX_SIZE_TUPLE];
};
```

Código 4.1: Escopo da classe CodeWord.

4.2 CodeBook

A classe CodeBook contém as características de um píxel ao longo de um certo intervalo de tempo. A classe é composta por CodeWords, funções de diferença de cor, detecção e manutenção dos CodeWords. O Código 4.2 contém o escopo da classe CodeBook.

```
class CodeBook
{
public:
    CodeBook();
    ~CodeBook();
    virtual bool isBackground(float colors[], float thresholds[], int
        indexToCodeWord) = 0;
    virtual void updateMatchedCodeWord(float colors[]) = 0;
    int searchCodeWord(float values[], float thresholds[]);
    // N = number of frames
    // sizeTuple = 6(RGB) or 4(HSV, HLS, LAB, YCbCr)
    void wrapAroundLambda(float N, int sizeTuple);
    void removeUnderutilizedCodeWords(int N, int sizeTuple);

    int x,y;
    std::vector<cbook::CodeWord> _codeWords;
};
```

Código 4.2: Escopo da classe CodeBook.

A função **isBackground()** contém o cálculo de diferença de cor entre píxeis. Como esta função foi implementada de maneira diferente para cada espaço de cor (RGB, LAB, HLS, HSV e YCbCr), elas são comentadas na seção 4.3.

A função **searchCodeWord()** tem como objetivo verificar se, a partir dos CodeWords existentes no respectivo CodeBook, há algum dos CodeWords que consiga classificar o respectivo píxel de entrada como *background*. Em caso afirmativo, é retornada a posição de referência do codeWord selecionado, caso contrário é retornado um valor de erro (-1) e um novo CodeWord será construído para o respectivo CodeBook.

A função **updateMatchedCodeWord()** tem o objetivo de atualizar o valor de um CodeWord cada vez que o mesmo consegue classificar um píxel como *foreground*. Dessa forma, tanto valores de cor quanto valores de controle de brilho e de frequência são atualizados, dependendo da implementação feita em cada versão do algoritmo para cada diferente espaço de cor.

Para o refinamento dos CodeBooks foi implementada a função **removeUnderutilizedCodeWords()**. Esta função estipula um valor de limiar T para verificar quais CodeWords devem ser removidos de cada CodeBook que compõem a estrutura com o modelo do *background*.

Dessa forma, CodeWords pouco utilizados serão descartados, diminuindo a quantidade de memória gasta para guardar o modelo do *background* e diminuindo o tempo gasto para classificar um píxel em *foreoground* e *background*, visto que a quantidade de CodeWords utilizados na comparação dos píxeis é reduzida. Esta função é relatada na subseção 2.3.1.2 que trata o algoritmo *CBS*.

Relacionando com a função **removeUnderutilizedCodeWords()** que se utiliza do valor de λ de cada CodeWord, correspondendo ao maior intervalo em que um CodeWord não foi utilizado, tem-se a função **wrapAroundLambda()**. Esta função tem por objetivo atualizar o valor de λ dos CodeWords pertencentes ao CodeBook.

4.3 Funções de diferença de cor

Como foi dito anteriormente, a função **isBackground()**, presente na classe CodeBook, foi implementada para verificar se um píxel pode ou não ser classificado como *background* de uma cena. Sendo assim, foram implementadas funções auxiliares para cada um dos cálculos nos diferentes espaços de cor propostos. As implementações das funções de diferença de cor entre píxeis estão presentes no apêndice A.

As funções de diferença de cor relatadas na presente seção foram separadas de acordo com o espaço de cor utilizado, sendo eles:

- **RGB**: Na versão do CodeBook para o espaço de cor RGB, o cálculo de para classificação dos píxeis é realizado a partir de duas funções:
  - **colordist()**: calcula a distorção de cor relatada na seção 2.3.2.1, computando a distância de tonalidade entre as cores.
  - **brightness()**: calcula a distorção de brilho, demonstrada na seção 2.3.2.2. Esta fórmula verifica se o valor de brilho de uma cor está entre dois limites superior e inferior.
- **LAB**: Na versão do CodeBook para o espaço de cor LAB, a função **CIEDE2000()** utilizada para o cálculo de diferença de cor se refere à fórmula CIEDE2000 para cálculo de distância entre duas cores no espaço de cor LAB. A fórmula foi demonstrada no capítulo 3.
- **HSV**: Na versão do CodeBook para o espaço de cor HSV, foi utilizada uma abordagem

semelhante com a implementação em RGB. Nessa implementação, as cores são avaliadas de acordo com as diferenças de tonalidade e de brilho, representadas respectivamente pelas funções **colorDistortion()** e **valueDistortion()**. As fórmulas das duas funções foram apresentadas no capítulo 3.

- **HLS**: Na versão do CodeBook para o espaço de cor HLS foi utiliza apenas uma função chamada **hlsDistorted()**. A fórmula utilizada nessa função foi descrita no capítulo 3.

- **YCbCr**: Na versão do CodeBook para o espaço de cor YCbCr, assim como no espaço de cor HSV e RGB, foi utilizada a abordagem de avaliar separadamente as cores, entre tonalidade e brilho. Então foram implementadas as funções **crominanceDistance()** e **luminanceDistance()**. As fórmulas implementadas foram descritas no capítulo 3.

4.4 CodeBookManager

A classe CodeBookManager tem a função de criar e manipular toda a estrutura de CodeBooks, utilizando-os para classificar os píxeis de um frame de um vídeo em *foreground* e *background*. As funções desta seção se dividem de acordo com as fases do algoritmo: treinamento, refinamento e detecção. O escopo da classe CodeBookManager é apresentado pelo código 4.3.

```
class CodeBook
{
public:
    CodeBook();
    ~CodeBook();
    virtual bool isBackground(float colors[], float thresholds[], int
        indexToCodeWord) = 0;
    virtual void updateMatchedCodeWord(float colors[]) = 0;
    int searchCodeWord(float values[], float thresholds[]);
    // N = number of frames
    // sizeTuple = 6(RGB) or 4(HSV, HLS, LAB, YCbCr)
    void wrapAroundLambda(float N, int sizeTuple);
    void removeUnderutilizedCodeWords(int N, int sizeTuple);

    int x,y;
    std::vector<cbook::CodeWord> _codeWords;
};
```

Código 4.3: Escopo da classe CodeBookManager.

Na fase de construção dos CodeBooks, as principais funções chamadas pela classe CodeBookManager são: **construct()**, **trainFrame()**. O pseudocódigo da construção dos CodeBooks foi ilustrado na seção 2.3.1.1.

A função **construct()**, realiza toda a construção e treinamento dos CodeBooks, recebendo apenas um frame ou uma fonte para captura de frames (arquivo de vídeo ou uma câmera). Ao final, tem-se a matriz de CodeBooks construída.

Dentro da função **construct()** são chamadas as funções **trainFrame()** e **wrapAroundLambda()**:

- **trainFrame()**: esta função tem por objetivo receber um frame e utilizá-lo para treinar os CodeBooks, que na fase de detecção serão utilizados para classificar os píxeis entre *foreground* e *background*. A função **trainFrame()** é chamada para cada um dos *N* frames presentes na cena de referência que contém o *background* da cena. Nas versões para os diferentes espaços de cor, a transformação das cores do espaço RGB da câmera para um dos outros espaços de cor é realizada nessa função, pois a câmera captura, por padrão, cada frame no espaço de cor RGB.

- **wrapAroundLambda()**: Esta função tem ligação direta com a fase de refinamento dos CodeBooks, pois esta tem o objetivo de chamar a função **wrapAroundLambda()** de cada um dos CodeBooks existentes da matriz de CodeBooks, criada e manipulada pela classe CodeBookManager.

A classe CodeBookManager possui a função **removeUnderutilizedCodeWords()** que, assim como a função **wrapAroundLambda()**, chama a função de refinamento **removeUnderutilizedCodeWords()** em cada um dos CodeBooks existentes na estrutura.

Na fase de detecção do algoritmo *CBS*, a principal função chamada pela classe CodeBookManager é a **getSubtractionMask()**. O pseudocódigo da fase de detecção do algoritmo *CBS* foi ilustrado na seção 2.3.1.3.

A função **getSubtractionMask()** recebe um frame e tem por objetivo retornar uma máscara contendo píxeis pretos e brancos. Os píxeis brancos são os píxeis que foram classificados como *foreground* e os píxeis pretos são os píxeis que foram classificados como *background*. Dentro da função **getSubtractionMask()**, assim como na função **trainFrame()** é chamada a função **searchCodeWord()**, que contém o cálculo de diferença de cor entre píxeis para cada espaço de cor.

Por fim, a função **backgroundSubtraction()** compreende na troca dos píxeis brancos da imagem final pelos píxeis do respectivo frame que foi avaliado.

4.5 Funções de pós-processamento

As funções de pós-processamento implementadas tem por objetivo refinar os resultados gerados da fase de detecção e são chamadas entre as funções **getSubtractionMask()** e **backgroundSubtraction()** da classe CodeBookManager, que foram demonstradas na seção 4.4.

- **Opening()**: função que aplica o filtro de abertura na imagem. A implementação deste método foi realizada através do uso das funções disponibilizadas pela biblioteca OpenCV, visto que ela possui suporte para filtros de erosão e dilatação. O código 4.4 contém a implementação da função **Opening()** e das funções **erosion()** e **dilation()**.

```
cv::Mat erosion(cv::Mat frame, int sSize)
{
    cv::Mat element;
    int erosion_type = cv::MORPH_RECT; //MORPH_ELLIPSE, MORPH_RECT
       MORPH_CROSS
    int erosion_size = sSize;
    element = cv::getStructuringElement(erosion_type, cv::Size( 2*
       erosion_size + 1, 2*erosion_size+1 ), cv::Point(erosion_size,
       erosion_size));
    cv::erode(frame, frame, element);
    return frame;
}

cv::Mat dilation(cv::Mat frame, int sSize)
{
    cv::Mat element;
    int dilation_type = cv::MORPH_RECT; //MORPH_ELLIPSE, MORPH_RECT
       MORPH_CROSS
    int dilation_size = sSize;
    element = cv::getStructuringElement(dilation_type, cv::Size( 2*
       dilation_size + 1, 2*dilation_size+1 ), cv::Point(dilation_size
       ,dilation_size));
    cv::dilate(frame, frame, element);
    return frame;
}

cv::Mat Opening(cv::Mat frame, int sSize)
{
    frame = erosion(frame, sSize);
    frame = dilation(frame, sSize);
    return frame;
}
```

Código 4.4: Função **Opening()**.

- **histogramEqualization()**: realiza a equalização de histogramas em uma imagem. Utilizou-se a implementação disponível da biblioteca OpenCV. O código 4.5 contém a implementação da função **histogramEqualization()**.

```
cv::Mat histogramEqualization(cv::Mat frame)
{
    cv::Mat out;
    cv::Mat frameGrayscale = frame.clone();
    cvtColor( frameGrayscale, frameGrayscale, CV_BGR2GRAY );
    equalizeHist( frameGrayscale, out );
    return out;
}
```

Código 4.5: Função **histogramEqualization()**.

- **CannyEdgeDetector()**: algoritmo que realiza a detecção de bordas em uma imagem em tons de cinza. Foi utilizada a função disponível pela biblioteca OpenCV. O código 4.6 contém a implementação da função **CannyEdgeDetector()**.

```
cv::Mat CannyEdgeDetector(cv::Mat frame, int lowThreshold = 0, int
    highThreshold = 30)
{
    cv::Mat edges = frame.clone();
    cv::cvtColor(edges, edges, CV_BGR2GRAY);
    GaussianBlur(edges, edges, cv::Size(7,7), 1.5, 1.5);
    Canny(edges, edges, lowThreshold, highThreshold, 3);
    return edges;
}
```

Código 4.6: Função **CannyEdgeDetector()**.

- **fillTheHolesMask()**: função que tem por objetivo o preenchimento de regiões pretas da imagem que estão dentro de uma região branca. Para a implementação deste método foi utilizado uma função do OpenCV que desenha um contorno em uma borda. Repetindo o passo N vezes, são preenchido os buracos da imagem. O código 4.7 contém a implementação da função **fillTheHolesMask()**.

```
//http://bsd-noobz.com/opencv-guide/60-2-color-based-object-
    segmentation
cv::Mat fillTheHolesMask(cv::Mat mask)
{
    std::vector<std::vector<cv::Point> > contours;
    findContours(mask.clone(), contours, CV_RETR_EXTERNAL,
        CV_CHAIN_APPROX_SIMPLE);
    cv::Mat dst = cv::Mat::zeros(mask.size(), mask.type());
    drawContours(dst, contours, -1, cv::Scalar::all(255), CV_FILLED);
    return dst;
}
```

Código 4.7: Função **fillTheHolesMask()**.

5 RESULTADOS OBTIDOS

Os resultados se mostram diferentes, dependendo das características de cada cena de referência e cena de entrada. Os resultados estão organizados em tabelas, com os frames gerados nos experimentos com cada espaço de cor, juntamente com as técnicas de processamento de imagens utilizadas em cada teste. Nesse trabalho, são apresentados os resultados de dois testes realizados.

Para o primeiro teste são utilizados dois vídeos contendo a cena de referência e cena de entrada. A figura 5.1 ilustra um frame retirado de cada vídeo. Os resultados do primeiro teste são apresentados na tabela 5.1.

(a) (b)

Figura 5.1: Respectivamente: (a) frame retirado da cena de Referência, (b) frame retirado da cena de entrada.

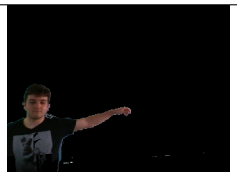
Os resultados constatados no primeiro teste são:

- HLS: Gerou um resultado com poucas imperfeições e parte do *background* não foi detectado devido a uma diferença muito grande de cor proveniente das sombras geradas na cena de entrada. A utilização do filtro morfológico de abertura auxiliou na diminuição de ruídos na imagem e a função de preenchimento de buracos melhorou os resultados de detecção do *foreground*.

- HSV: Gerou um pouco mais de ruídos se comparado com os resultados gerados pela abordagem utilizando o espaço de cor HLS. Porém, realizou uma detecção mais precisa do *background*. O filtro morfológico de abertura diminuiu a quantidade de ruídos existentes e o preenchimento de buracos melhorou a detecção do *foreground*.

- LAB: Gerou um resultado com problemas tanto no *background* quanto no *foreground* e as sombras existentes na cena de entrada tiveram grande influência na detecção do *background*. O filtro morfológico de abertura, ao contrário do que era previsto, piorou

Tabela 5.1: Resultados gerados a partir dos vídeos contendo os frames 5.1a (cena de referência) e 5.2b (cena de entrada).

Espaço de Cor	Algoritmo Base	Algoritmo Base + Filtro de Abertura	Algoritmo Base + Filtro de Abertura + Fill Holes
HLS			
HSV			
LAB			
RGB			
YCbCr			

a detecção do *foreground*, impedindo a função de preenchimento de buracos preencher certas regiões de interesse que foram marcadas como *backgorund* da cena.

- RGB: Gerou boas detecções, com a presença mediana de ruídos. O filtro morfológico de abertura corrigiu os ruídos presentes no *background* e o preenchimento de ruídos melhorou as imperfeições geradas no *foreground*.

- YCrCb: Gerou boas detecções, com uma quantidade muito baixa de ruídos. O filtro morfológico de abertura foi utilizado porém não modificou muito o resultado, mesmo retirando o resto de ruídos aparentes. O preenchimento de buracos não modificou a cena final gerada.

No segundo teste, assim como no primeiro, são utilizados dois vídeos contendo a cena de referência e cena de entrada. A figura 5.2 ilustra um frame retirado de cada vídeo e os resultados do segundo teste são apresentados na tabela 5.2.

(a) (b)

Figura 5.2: Respectivamente: (a) frame retirado da cena de Referência, (b) frame retirado da cena de entrada.

Os resultados constatados no segundo teste são:

- HLS: Resultou em regiões de *background* mal detectadas por causa das sombras que apareceram na cena de entrada. O uso do filtro morfológico de abertura diminuiu os ruídos gerados na saída da detecção, porém isso causou um erro perceptível no *foreground* que não pôde ser solucionado com o preenchimento de buracos.

- HSV: Os resultados gerados apresentam imperfeições no *background*, não sendo solucionadas pelo filtro morfológico de abertura. O preenchimento de buracos corrigiu alguns píxeis que haviam sido detectados como *background*.

- LAB: Gerou imperfeições perceptíveis no *foreground*, além de más detecções do *background*. O filtro morfológico de abertura impediu a correta detecção do *foreground*, po-

Tabela 5.2: Resultados gerados a partir dos vídeos contendo os frames 5.2a (cena de referência) e 5.2b (cena de entrada).

Espaço de Cor	Algoritmo Base	Algoritmo Base + Filtro de Abertura	Algoritmo Base + Filtro de Abertura + Fill Holes
HLS			
HSV			
LAB			
RGB			
YCbCr			

Tabela 5.3: Tabela comparativa entre os tempos (em segundos) de cada abordagem proposta.

Espaço de Cor	Algoritmo Base	Algoritmo Base + Filtro de Abertura	Algoritmo Base + Filtro de Abertura + Fill Holes
HLS	12,49	12,53	12,72
HSV	24,91	24,95	25,37
LAB	71,27	71,37	74,07
RGB	16,82	16,87	16,96
YCbCr	10,43	10,46	10,74

rém retirou parte do ruído no *background*. O preenchimento de buracos consertou algumas imperfeições do *foreground*, porém o resultado ainda foi ruim.

- RGB: Os resultados apresentam um pouco de ruído, solucionados com o filtro morfológico de abertura. Pelo fato de não haver fendas na região de interesse referente ao *foreground*, o preenchimento de buracos não melhorou o resultado.

- YCrCb: Gerou pouca quantidade de ruídos e, neste caso, o preenchimento de buracos prejudicou o resultado final. O filtro morfológico de abertura retirou parte do ruído presentes na detecção.

Utilizando a cena de referência e a cena de entrada do segundo teste (representadas pelas figuras 5.2a e 5.2b), foi realizado um comparativo entre os tempos que cada abordagem necessita para classificar os píxeis de cada frame de um vídeo em *foreground* e *background*.

A tabela 5.3 contém os tempos (em segundos) para cada abordagem utilizada, ressaltando que os tempos capturados medem, para cada estratégia proposta, as fases de treinamento, refinamento de CodeBooks, detecção e refinamento do resultado gerado. A cena de referência utilizada possui 02 segundos de duração e a cena de entrada utilizada possui 08 segundos de duração.

Os vídeos utilizados para o teste são de dimensão 640x424, e o vídeo não foi processado e mostrado ao mesmo tempo para o usuário durante a execução dos teste. Sendo assim, o processo consistiu em salvar o resultado gerado em um arquivo de vídeo para ser visualizado depois, pois o que interessava era o tempo que cada abordagem levava para processar todas as fases de treinamento, refinamento de CodeBooks, detecção e refinamento do resultado gerado.

Resumindo, para cada um dos espaços de cor testados foram verificados os seguintes pontos:

- HLS:
  - Gerou muitos ruídos e muito sensível às sombras que aparecem;
  - Não realizou uma boa detecção do *foreground* no segundo teste feito, prejudicando o resultado almejado.
- HSV:
  - Também percebe-se a grande quantidade de ruídos e a sensibilidade do modelo de fundo às sombras que aparecem;
  - O *foreground* detectado contém a presença de contornos mal definidos. Dessa forma, é possível perceber que nas bordas do *foreground* detectado, há a presença perceptível de píxeis do *background*.
- LAB:
  - Se mostrou como o pior resultado gerado, pois houve má detecção do *foreground*, como foi visto no segundo teste realizado;
  - Assim como nos espaços HLS e HSV, se mostrou muito sensível às mudanças de iluminação, provenientes das sombras geradas pelos novos elementos apresentados na cena de entrada.
- RGB:
  - Gerou boas detecções, como foi apresentado por Kim (KIM et al., 2005);
  - Mesmo sendo robusto para variações de iluminação, percebeu-se que as sombras presentes na cena, provenientes de novos elementos apresentados apenas na cena de entrada, geraram uma má detecção do *background* da cena.
- YCrCb:
  - Pouca presença de ruídos, dispensando o uso do filtro de abertura na imagem nos casos testados;
  - Contém algumas imperfeições no quesito de variação de iluminação e geração de sombras provenientes dos novos elementos apresentados na cena de entrada;
  - Abordagem mais rápida (demora menos tempo para realizar a classificação dos píxeis em *foreground* e *background*);

– O padrão YCbCr abrange uma gama maior de tonalidades, o que pode ter eliminado um pouco da sensibilidade.

Tanto as implementações em RGB quanto em YCbCr geraram resultados muito satisfatórios. Uma das observações feitas entre os resultados gerados por estas duas abordagens, foi que os resultados gerados pelo espaço YCbCr geraram contornos mais bem definidos na parte detectada. Sendo assim, os contornos do *foreground* no espaço YCbCr possuem menos píxeis do *background*.

Foi visto que a utilização do filtro morfológico de abertura nas imagens diminui consideravelmente o ruído gerado pela imagem após a fase de detecção, tornando a imagem mais limpa.

A equalização de histogramas juntamente com o algoritmo de detecção de bordas não geraram resultados satisfatórios. Dessa forma, preferiu-se abortar o uso destas funções. Os resultados obtidos acabaram por gerar má formação das regiões de interesse, muitas vezes com pequenas fendas presentes na imagem. A figura 5.3 ilustra as pequenas fendas encontradas nas regiões de interesse.

(a) (b) (c)

Figura 5.3: (a) Cena de referência (b) Cena de entrada (c) Imagem pós detecção de bordas e aplicação da função de preenchimento de buracos.

Dessa forma, o algoritmo de preenchimento de buracos que seria utilizado acaba por não preencher a região de interesse, ou pode preencher áreas que não fazem parte do *foreground*, gerando piores resultados.

Para buscar contornar o problema das detecções parciais com uma abordagem diferente da utilização da função de detecção de bordas, foi utilizada a função de preenchimento de buracos.

O preenchimento de buracos mostrou que em alguns casos a imagem é melhorada significativamente, porém quando os objetos de interesse possuem uma fenda ou u buraco e este buraco deve ser classificado em background, esta função acabar piorando o resultado final.

Mesmo se utilizando dessas técnicas e abordagens diferentes, não foi encontrada uma maneira totalmente eficaz de detectar as sombras geradas pelos novos elementos da cena de entrada, o que gerou regiões erradas nos *foregrounds* das imagens geradas.

Uma das limitações das abordagens utilizadas é a grande quantidade de cálculos para classificar os píxeis, visto que cada píxel de um frame é avaliado separadamente. Porém, pelo fato dos CodeBooks serem tratados independentemente, o algoritmo é altamente paralelizável.

6 CONCLUSÃO

Este trabalho teve como objetivo o estudo e implementação do algoritmo *CodeBook Background Subtraction* proposto por Kim (KIM et al., 2005), além da pesquisa sobre diferentes espaços de cor e algoritmos de processamento de imagens, visando a detecção de *foreground* e *background* de vídeos.

O objetivo de realizar a implementação do algoritmo *CBS* e a pesquisa de abordagens para melhorar os resultados inicialmente gerados foi alcançado. As diferentes situações geradas nos vídeos (diferenças de iluminação, tipos de elementos do *foreground* e distância destes elementos da câmera) mostraram que é complexo encontrar um método que consiga gerar um resultado satisfatório para todos os casos. No entanto, as estratégias propostas já demonstraram que houve melhorias no resultado final, comparadas com a implementação inicial feita.

Foram testados 5 espaços de cor, sendo eles: HSV, HLS, LAB, RGB e YCbCr. A proposta que gerou melhores resultados utilizou-se do espaço de cor YCbCr, gerando uma quantidade menor de imperfeições no *background* para os testes que foram utilizados.

Além disso, foi realizado um comparativo medindo os tempos de cada abordagem. Dentre elas, a abordagem utilizando o espaço de cor YCbCr mostrou ser mais rápida que as outras estratégias propostas, visto que foi utilizado um cálculo simples de distância entre píxeis.

As funções de pós processamento auxiliaram na detecção de *foreground* e *background*, retirando os ruídos existentes no *background* e preenchendo buracos de *background* envoltos pelo *foreground*.

6.1 Contribuições alcançadas

Neste trabalho foi possivel realizar uma discussão a respeito do algoritmo *CodeBook Background Subtraction*, testar o algoritmo em diferentes espaços de cor e verificar quais métodos são válidos de se utilizar para o refinamento dos resultados, visto que os pontos positivos e negativos de cada método adicionado no refinamento se fez importante para avaliar cada uma das propostas.

6.2 Trabalhos Futuros

A implementação do suporte para GPU's, assim como o trabalho proposto por Andreas Griesser (GRIESSER et al., 2005), seria um tópico de interesse, visto que tornanria o processo de simulação mais rápido.

A adição de um ou mais CodeBooks, como no trabalho proposto por Mohamad Hoseyn Sigari(SIGARI; FATHY, 1999), pode ser utilizada para aumentar o nível de camadas de detecção. Com esta abordagem pode-se criar novas regras para a classificação de uma cena, separadas por camadas.

Podem ser adicionados algoritmos para reoconhecimento de objetos ou para reconhecimento de cor de pele. Dessa forma, eles seriam utilizados para mostrar ao usuário qual o elemento que foi detectado no *foreground*.

REFERÊNCIAS

AUTOMATIC Memory Management, OpenCV documentation. Acessado em Novembro/2012, http://docs.opencv.org/modules/core/doc/intro.html.

BOUWMANS, T.; BAF, F. E.; VACHON, B. Background Modeling using Mixture of Gaussians for Foreground Detection - A Survey. In: **Anais...** Recent Patents on Computer Science, 2008.

BUTLER, D. E.; JR., V. M. B.; SRIDHARAN, S. Real-time Adaptive Foreground/Background Segmentation. **Journal on Applied Signal Processing**, [S.l.], v.14, 2005.

CANNY, J. A Computational Approach to Edge Detection. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, [S.l.], v.PAMI-8, Issue: 6, 1986.

CELENK, M. A Color Clustering Technique for Image Segmentation. In: RECENT PATENTS ON COMPUTER SCIENCE. **Anais...** Computer Vision: Graphics: and Image Processing, 1990. v.52, p.145 – 170.

CRISTANI, M.; BICEGO, M.; MURINO, V. Integrated Region-and Pixel-based Approach to Background Modelling. In: COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING. **Anais...** Motion and Video Computing: 2002. Proceedings. Workshop on, 2002. p.3 – 8.

FISHER, R. B. **Change Detection in Color Images**. 1999.

GARG, R.; MITTAL, B.; GARG, S. Histogram Equalization Techniques For Image Enhancement. **InternatIonal Journal of electronIcs and communIcatIon technology**, [S.l.], v.2, 2011.

GRANT, R.; CLARK, A.; GREEN, R. **HLS Distorted colour model for enhanced colour image segmentation**. 2008. 1-6,26-28p.

GRIESSER, A. et al. GPU-Based Foreground-Background Segmentation using an Extended Colinearity Criterion. **VMV**, [S.l.], 2005.

HORPRASERT, T.; HARWOOD, D.; DAVIS, L. S. A statistical approach for real-time robust background subtraction and shadow detection. In: **Anais...** [S.l.: s.n.], 1999. p.1–19.

INPE. **Filtragem - Spring**. 2012.

Kim, H.-S. et al. Digital watermarking based on color differences. In: SOCIETY OF PHOTO-OPTICAL INSTRUMENTATION ENGINEERS (SPIE) CONFERENCE SERIES. **Anais...** [S.l.: s.n.], 2001. p.10–17. (Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, v.4314).

KIM, K. et al. Real-time foreground-background segmentation using codebook model. **Journal Real-Time Imaging**, [S.l.], v.11, 2005.

LEE, D.-S.; HULL, J. J.; EROL, B. A Bayesian framework for Gaussian mixture background modeling. In: **Anais...** Image Processing: 2003. ICIP 2003. Proceedings. 2003 International Conference on, 2003. v.3, p.III – 973–6 vol.2.

LUO, M. R.; CUI, G.; RIGG, B. The Development of the CIE2000 Colour-Difference Formula: ciede2000. In: IMAGE PROCESSING, 2003. ICIP 2003. PROCEEDINGS. 2003 INTERNATIONAL CONFERENCE ON. **Anais...** Color Research and Application, 2001. v.26, n.5.

MENESES, P. R. et al. **Introdução ao Processamento de Imagens de Sensoriamento Remoto**. Acessado em Novembro/2012, http://www.cnpq.br/documents/10157/56b578c4-0fd5-4b9f-b82a-e9693e4f69d8.

NADERNEJAD, E.; SHARIFZADEH, S.; HASSANPOUR, H. Edge Detection Techniques: evaluations and comparisons. In: COLOR RESEARCH AND APPLICATION. **Anais...** Applied Mathematical Sciences, 2008. v.2.

OPENCV - Open Source Computer Vision. Acessado em Novembro/2012, http://opencv.willowgarage.com/wiki.

OPENCV documentation. Acessado em Novembro/2012, http://docs.opencv.org.

SHAPIRO, L. G.; STOCKMAN, G. C. **Computer Vision**. [S.l.: s.n.], 2001.

SHARMA, G.; DALAL, W. W. E. N. The CIEDE2000 Color-Difference Formula: implementation notes, supplementary test data, and mathematical observations. In: APPLIED MATHEMATICAL SCIENCES. **Anais...** Color Research and Application, 2005. v.30, n.1.

SIGARI, M. H.; FATHY, M. Real-time Background Modeling/Subtraction using Two-Layer Codebook Model. **Proceedings of the International MultiConference of Engineers and Computer Scientists**, [S.l.], v.1, 1999.

STAUFFER, C.; GRIMSON, W. Adaptive background mixture models for real-time tracking. In: **Anais...** [S.l.: s.n.], 1999. v.2.

TOYAMA, K. et al. Wallflower: principles and practice of background maintenance. In: **Anais...** International Conference on Computer Vision, 1999.

APÊNDICES

APÊNDICE A – Cálculos de diferença entre cores

```
float hlsDistorted(float values[], int indexToCodeWord)
{
    float l_min = std::min(128 - abs(128 - values[2]), 128 - abs(128 -
        _codeWords[indexToCodeWord].atributes[2]))/128;
    float s_min = std::min(values[1], _codeWords[indexToCodeWord].atributes
        [1])/256;

    float d_h = std::min(abs(values[0] - _codeWords[indexToCodeWord].
        atributes[0]), 360 - abs(values[0] - _codeWords[indexToCodeWord].
        atributes[0])) * l_min * s_min;
    float d_l = values[1] - _codeWords[indexToCodeWord].atributes[1];
    float d_s = (values[2] - _codeWords[indexToCodeWord].atributes[2]) *
        s_min;

    /***Constants***/
    float alfa = 1.f;
    float beta = 1.f;
    float gama = 1.f;
    /***Constants***/

    return sqrt(pow(d_h*alfa, 2) + pow(d_l*beta, 2) + pow(d_s*gama, 2));
}
```

Código A.1: Função de distorção de cor utilizada para o espaço de cor HLS.

```
float colorDistortion(float colors[], int indexToCodeWord)
{
    float x1 = colors[1] * cos(2.f * (float)M_PI * colors[0]);
    float y1 = colors[1] * sin(2.f * (float)M_PI * colors[0]);
    float x2 = _codeWords[indexToCodeWord].atributes[1] * cos(2.f * (float)
        M_PI * _codeWords[indexToCodeWord].atributes[0]);
    float y2 = _codeWords[indexToCodeWord].atributes[1] * sin(2.f * (float)
        M_PI * _codeWords[indexToCodeWord].atributes[0]);

    return sqrt( pow(x2 - x1, 2) + pow(y2 - y1, 2) );
}

float valueDistortion(float colors[], int indexToCodeWord)
{
    return abs(_codeWords[indexToCodeWord].atributes[2] - colors[2]);
}
```

Código A.2: Função de distorção de cor utilizada para o espaço de cor HSV.

```
/*****************************************************************
 *
 * CIEDE2000 Color Difference Calculator
 *
 * Copyright (c) 2007 yoneh (http://d.hatena.ne.jp/yoneh/)
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the
```

```
 * "Software"), to deal in the Software without restriction, including
 * without limitation the rights to use, copy, modify, merge, publish,
 * distribute, sublicense, and/or sell copies of the Software, and to
 * permit persons to whom the Software is furnished to do so,
 * subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 * OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
 * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
 * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
 * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 *
 ***********************************************************************
 *
 * References:
 * [1] Gaurav Sharma, Wencheng Wu, and Edul N. Dalal, ``The CIEDE2000
 * Color-Difference Formula: Implementation Notes, Supplementary Test Data,
 * and Mathematical Observations'', Color Research and Application,
 * Vol. 30, No. 1, Feb 2005.
 *
 ***********************************************************************/

inline bool tolerance_zero(const double x)
{
    return fabs(x)<1e-9;
}

inline double cosd(const double degree)
{
    return cos(degree*M_PI/180.0);
}

inline double sind(const double degree)
{
    return sin(degree*M_PI/180.0);
}

inline double fqatan(const double y,const double x)
{
    double  t=atan2(y,x)/M_PI*180.0;

    if(t<0.0)
        t+=360.0;

    return t;
}

inline double f7(const double x)
{
    // if x is small, using the following approx.
    if(x<1.0)
        return pow(x/25.0,3.5);
```

```
    return 1.0/sqrt(1.0+pow(25.0/x,7.0));
}

double CIEDE2000(const double L1,const double a1,const double b1,
                 const double L2,const double a2,const double b2)
{
    const double epsilon=1e-9;
    double  C1ab,C2ab;
    double  Cab,G;
    double  a1_,a2_;
    double  C1_,C2_;
    double  h1_,h2_;

    C1ab=sqrt(a1*a1+b1*b1);
    C2ab=sqrt(a2*a2+b2*b2);
    Cab=(C1ab+C2ab)/2.0;
    G=0.5*(1.0-f7(Cab));
    a1_=(1.0+G)*a1;
    a2_=(1.0+G)*a2;
    C1_=sqrt(a1_*a1_+b1*b1);
    C2_=sqrt(a2_*a2_+b2*b2);

    if(tolerance_zero(a1_) && tolerance_zero(b1))
        h1_=0.0;
    else
        h1_=fqatan(b1,a1_);
    if(tolerance_zero(a2_) && tolerance_zero(b2))
        h2_=0.0;
    else
        h2_=fqatan(b2,a2_);

    // Calculate dL', dC', and dH'
    double  dL_,dC_,dH_,dh_;
    double  C12;

    dL_=L2-L1;
    dC_=C2_-C1_;
    C12=C1_*C2_;
    if(tolerance_zero(C12))
    {
        dh_=0.0;
    }
    else
    {
        double  tmp=h2_-h1_;

        if(fabs(tmp)<=180.0+epsilon)
            dh_=tmp;
        else if(tmp>180.0)
            dh_=tmp-360.0;
        else if(tmp<-180.0)
            dh_=tmp+360.0;
    }
    dH_=2.0*sqrt(C12)*sind(dh_/2.0);


    // Calculate L', C', h', T, and dTh
    double  L_,C_,h_,T,dTh;
```

```
    L_=(L1+L2)/2.0;
    C_=(C1_+C2_)/2.0;
    if(tolerance_zero(C12))
    {
        h_=h1_+h2_;
    }
    else
    {
        double  tmp1=fabs(h1_-h2_);
        double  tmp2=h1_+h2_;

        if(tmp1<=180.0+epsilon)
            h_=tmp2/2.0;
        else if(tmp2<360.0)
            h_=(tmp2+360.0)/2.0;
        else if(tmp2>=360.0)
            h_=(tmp2-360.0)/2.0;
    }
    T=1.0-0.17*cosd(h_-30.0)+0.24*cosd(2.0*h_)
        +0.32*cosd(3.0*h_+6.0)-0.2*cosd(4.0*h_-63.0);
    dTh=30.0*exp(-pow((h_-275.0)/25.0,2.0));


    // Calculate RC, SL, SC, SH, and RT
    double  RC,SL,SC,SH,RT;
    double  L_2=(L_-50.0)*(L_-50.0);

    RC=2.0*f7(C_);
    SL=1.0+0.015*L_2/sqrt(20.0+L_2);
    SC=1.0+0.045*C_;
    SH=1.0+0.015*C_*T;
    RT=-sind(2.0*dTh)*RC;


    //Calculate dE00
    const double kL = 1.f; // These are proportionally coefficients
    const double kC = .8f; // and vary according to the condition.
    const double kH = .8f; // These mostly are 1.
    double LP=dL_/(kL*SL);
    double CP=dC_/(kC*SC);
    double HP=dH_/(kH*SH);

    return sqrt(LP*LP+CP*CP+HP*HP+RT*CP*HP);
}
```

Código A.3: Função de distorção de cor utilizada para o espaço de cor LAB.

```
#define ALFA 0.5f //0.4 ~ 0.7
#define BETA 1.3f //1.1 ~ 1.5

bool brightness(float I, float Imin, float Imax)
{
    float Ilow = ALFA*Imax;
    float Ihi = std::min(BETA*Imax,Imin/ALFA);

    return (Ilow <= I && I <= Ihi);
}
```

```
float colordist(float c[], float colors[])
{
    float xt2 = pow(c[0],2) + pow(c[1],2) + pow(c[2],2);
    float vm2 = pow(colors[0],2) + pow(colors[1],2) + pow(colors[2],2);
    float xt_vi2 = pow(c[0]*colors[0] + c[1]*colors[1] + c[2]*colors[2],2);

    float p2 = xt_vi2 / vm2;
    return (sqrt(xt2 - p2));
}
```

Código A.4: Função de distorção de cor utilizada para o espaço de cor RGB.

```
float luminanceDistance(float colors[], int indexToCodeWord)
{
    return abs(colors[0] - _codeWords[indexToCodeWord].atributes[0]);
}

float crominanceDistance(float colors[], int indexToCodeWord)
{
    return sqrt( pow(colors[1] - _codeWords[indexToCodeWord].atributes[1],
       2) + pow(colors[2] - _codeWords[indexToCodeWord].atributes[2], 2) );
}
```

Código A.5: Função de distorção de cor utilizada para o espaço de cor YCbCr.