

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

CAIO RIOS DE SOUZA
YGOR BARRETO RIBEIRO

VISÃO COMPUTACIONAL: UMA ABORDAGEM COM SOFTWARE
LIVRE

Campos dos Goytacazes/RJ
2012

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

CAIO RIOS DE SOUZA
YGOR BARRETO RIBEIRO

VISÃO COMPUTACIONAL: UMA ABORDAGEM COM SOFTWARE LIVRE

Trabalho de conclusão de curso apresentado
ao Instituto Federal Fluminense como requisito
parcial para conclusão do Bacharelado em Sis-
temas de Informação.

Orientador: Prof. MSc Fábio Duncan de Souza

Campos dos Goytacazes/RJ
2012

CAIO RIOS DE SOUZA
YGOR BARRETO RIBEIRO

VISÃO COMPUTACIONAL: UMA ABORDAGEM COM SOFTWARE
LIVRE

Trabalho de conclusão de curso apresentado ao
Instituto Federal Fluminense como requisito
parcial para conclusão do Bacharelado em
Sistemas de Informação.

Aprovada em 23 de Novembro de 2012

Banca avaliadora:

Prof. MSc Fábio Duncan de Souza (Orientador)
Mestre em Pesquisa Operacional e Inteligência Computacional / UCAM Campos
Instituto Federal de Educação, Ciência e Tecnologia Fluminense / Campus Campos
Centro

Prof. MSc Fernando Luiz de Carvalho e Silva
Mestre em Engenharia de Produção / UENF
Instituto Federal de Educação, Ciência e Tecnologia Fluminense / Campus Campos
Centro

Prof. MSc Philippe Leal Freire dos Santos
Mestre em Informática / UFES
Instituto Federal de Educação, Ciência e Tecnologia Fluminense / Campus Campos
Centro

Dedico este trabalho aos meus pais, Joel Matos de Souza e Ana Lúcia do Espírito Santo Rios Souza, e ao meu irmão, Victor Rios de Souza, exemplos de vida e dedicação a família.

Caio Rios de Souza

Dedico esse trabalho a meus pais Eduardo e Lívia, minha avó Maria Francisca e minha tia Váleria pelo exemplo de vida.

Ygor Barreto Ribeiro

AGRADECIMENTOS

Primeiramente, eu agradeço a Deus por guiar minha vida. Aos meus pais, Joel e Ana, agradeço por terem dado o melhor de si para que eu pudesse me tornar quem sou hoje. Ao meu irmão Victor e sua esposa Ludimila, obrigado pelos conselhos e apoio. Agradeço também a minha namorada Rebeca pela paciência, cobrança e por me ensinar a cada dia, através de amor e sinceridade, a me tornar uma pessoa melhor. E aos meus colegas de profissão da Petrobras, gostaria de agradecer pelos ensinamentos diários e incentivos para a conclusão deste trabalho.

Caio Rios de Souza

Muitas pessoas colaboraram de alguma forma para que fosse possível compartilhar meus conhecimentos adquiridos na forma desse trabalho, gostaria de agradecer a todos eles pelo apoio. Em especial, gostaria de agradecer a meus pais Eduardo e Lívia e minha irmã Pyllar pelo suporte e carinho durante toda a minha vida. Meus primos Daniel, André e Lucas pela cobrança, típica de irmãos mais velhos. Aos meus colegas de turma, foi muito benéfico o convívio com todos vocês. E por fim agradeço a Deus por ter me dado a oportunidade de chegar aqui.

Ygor Barreto Ribeiro

“Só termina uma vez. Tudo que acontece
antes é apenas progresso.”

Jacob, Lost

RESUMO

O sentido da visão nos seres humanos tem grande importância para a realização de tarefas do cotidiano. Devido a isso, desenvolver essa habilidade em sistemas de informação tem se tornado igualmente importante. Este campo de estudo denomina-se Visão Computacional. Nos últimos anos, o aumento de novas tecnologias tem gerado crescente avanço nesse campo de pesquisa. Apesar disso, a área de Visão Computacional ainda se encontra imatura, não tendo uma formulação padrão para resolução de problemas. Este trabalho propõe apresentar essa ciência, através de conceitos necessários para o seu entendimento. Também são mostradas as ferramentas que podem ser utilizadas para o desenvolvimento de aplicações utilizando a linguagem Python, apresentando técnicas comumente utilizadas no processo de implementação de sistemas na área de Visão Computacional. Ainda, através de um estudo de caso, é exposto o desenvolvimento de uma aplicação capaz de detectar e rastrear objetos circulares em vídeo. A proposta é criar um material de referência que possa ser utilizado para introdução aos estudos na área de Visão Computacional, através das ferramentas e práticas apresentadas neste trabalho.

PALAVRAS-CHAVE: Visão Computacional, Segmentação de imagem, Python, OpenCV

ABSTRACT

The sense of vision in humans have great importance to perform daily tasks. Because of this, the development of this skill in information systems has become equally important. This field of study is called Computer Vision. In recent years, the advance of new technologies has created an increasing breakthrough in this research field. Nevertheless, this area is still immature, lacking a standard formulation for troubleshooting. This paper aim to present this science through it's basic concepts. Tools that can be used to develop applications using the Python language are also shown, presenting techniques commonly used to implementing systems in the field of Computer Vision Systems. Yet, through a case study, the development of an application capable of detecting and tracking circular objects in video is exposed. The proposal is to create a reference material that can be used for introducing studies in this area, through the tools and practices presented in this work.

KEYWORDS: Computer Vision, Image Segmentation, Python, OpenCV

LISTA DE FIGURAS

2.1	Divisão das áreas da Visão Computacional, relacionadas de acordo com suas dependências. Adaptado de (Szeliski, 2010).	18
3.1	Representação do modelo RGB através do desenho de um cubo (Carvalho, 2006).	23
3.2	Representação do modelo HSV através do desenho de um cone (Carvalho, 2006).	23
3.3	Exemplo de gráficos de histogramas divididos nos canais de cores vermelho, verde e azul (Team, 2012).	24
4.1	Comparativo de desempenho entre a ferramenta OpenCV com e sem a biblioteca proprietária da Intel. Adaptado de (Bradski; Kaehler, 2008).	29
4.2	Estrutura da biblioteca OpenCV (Bradski; Kaehler, 2008).	30
4.3	Logotipo do Instituto Federal Fluminense.	36
4.4	Limiarização com diferentes <i>thresholds</i> no logotipo do IFF.	37
4.5	Contornos desenhados em seus respectivos elementos. Nos contornos da Figura 4.5(a), os dados foram obtidos com apenas um <i>threshold</i> , já na Figura 4.5(b) foram usados dois <i>thresholds</i> .	37
4.6	Rastreamento feito no círculo vermelho do logotipo do IFF.	39
4.7	Bibliotecas usadas pelo SimpleCV divididas em camadas.	41
5.1	Vídeo sendo reproduzido utilizando a biblioteca OpenCV.	50
5.2	Relacionamento entre as classes do sistema de rastreamento.	50
5.3	Tratamento nos <i>frames</i> do vídeo.	52
5.4	Resultado final do sistema, detectando uma bola laranja na tela e circulando-a.	55
5.5	Caso básico de detecção, com o objeto parado no vídeo.	56
5.6	Diversos testes feitos com o objeto em diferentes velocidades.	56
5.7	Diversos testes feitos com o objeto em um plano de fundo complexo, em diferentes proximidades com o vídeo.	57
5.8	Em alguns momentos, a aplicação falhou em detectar o objeto desejado.	58
5.9	Aproximando o objeto da luz artificial, a tendência é que a detecção também falhe.	58
5.10	Casos de sucesso na detecção do objeto em situações adversas.	59

LISTA DE CÓDIGOS

1	O tipo das variáveis se diferem devido o uso de <i>bindings</i> diferentes.	30
2	Remoção de pacotes instalados por padrão no sistema operacional Linux Ubuntu 11.10.	31
3	Instalação da ferramenta CMake.	31
4	Instalação de dependências do OpenCV.	31
5	Instalação das dependências do FFmpeg.	32
6	Instalação do FFmpeg.	32
7	Instalação do V4L.	32
8	Comandos iniciais para instalação do OpenCV.	33
9	Execução da instalação do OpenCV através do CMake.	33
10	Comandos finais para instalação do OpenCV.	33
11	Carregamento básico de imagem utilizando OpenCV.	34
12	Comando para visualização dos valores atribuidos a variável <i>img</i>	34
13	Saída do <i>array</i> , representando uma imagem carregada pela interface cv2.	34
14	Recorte na imagem pode ser realizado através de manipulações com <i>arrays</i>	35
15	Imagens são salvas utilizando o método do OpenCV que realiza essa operação.	35
16	Operação para inversão da ordem do espaço de cor BGR para RGB.	35
17	Recuperação de contornos dos objetos contidos na imagem com o logotipo do IFF.	36
18	Comando para imprimir saída, contendo o número de elementos da figura.	37
19	Toda a lógica para obtenção dos contornos e desenho dos mesmos.	38
20	Os momentos são recuperados ao passar um contorno que representa um objeto da imagem.	38
21	Os valores dos momentos do segundo elemento do logotipo do IFF são mantidos por um dicionário, um tipo de dados do Python.	38
22	Operações com os momentos do segundo elemento da lista de contornos.	39
23	Desenho feito na imagem de exemplo usando os parâmetros calculados a partir dos momentos do contorno.	39

24	Inicialização do vídeo através da biblioteca OpenCV.	39
25	Lógica para processamento e visualização dos <i>frames</i> do vídeo.	40
26	Instalação das dependências do SimpleCV.	42
27	Instalação da biblioteca SimpleCV através do GitHub.	42
28	Instalação do IPython.	43
29	Carregamento de uma imagem utilizando o <i>framework</i> SimpleCV.	43
30	Conversões de espaço de cor utilizando a ferramenta SimpleCV.	44
31	Recuperação dos <i>blobs</i> são feitos através da imagem, enquanto o responsável por desenhá-los é objeto <i>blob</i>	45
32	Operações com momentos dos <i>blobs</i> com o <i>framework</i> SimpleCV.	45
33	Lógica para inicialização de vídeo com o <i>framework</i> SimpleCV.	45
34	Chamada principal do aplicativo.	51
35	Chamada do método da classe <i>BallTracker</i> que recupera os valores do círculo baseando-se pelo <i>frame</i> atual.	51
36	Lógica do método <i>find_ball()</i> da classe <i>ColorTracker</i>	52
37	Recuperação dos valores pertinentes para identificar um círculo no <i>frame</i> de vídeo.	52
38	Método que recupera informações do clique esquerdo do mouse.	53
39	Converte o <i>frame</i> para espaço HSV.	53
40	Um objeto do tipo <i>ThresholdManager</i> é instanciado para que trate os valores HSV recuperados através do <i>pixel</i> do <i>frame</i>	54
41	Inicialização da classe <i>ThresholdManager</i>	54
42	Método responsável por ajustar os valores HSV do limiares usadas para binarizar o <i>frame</i> do vídeo.	55

SUMÁRIO

1 INTRODUÇÃO	13
1.1 Justificativa do trabalho	14
1.2 Objetivo	14
1.3 Estrutura do trabalho	14
2 VISÃO COMPUTACIONAL	15
2.1 Conceito	15
2.2 Histórico	16
2.3 Cenário atual	16
2.4 Estrutura da área de Visão Computacional	17
3 PRINCIPAIS PROCESSOS DE VISÃO COMPUTACIONAL	21
3.1 Processamento de Imagem	21
3.1.1 Representação de Cores	22
3.1.1.1 Modelo RGB	22
3.1.1.2 Modelo HSV	23
3.1.2 Histograma	24
3.2 Segmentação	24
3.2.1 Segmentação por região	25
3.2.1.1 <i>Thresholding</i>	25
3.2.1.2 Dividir e fundir	25
3.2.2 Segmentação por contornos	26
3.2.3 Segmentação por textura	26
3.3 Reconhecimento	26
4 BIBLIOTECAS DE VISÃO COMPUTACIONAL	28
4.1 OpenCV	29
4.1.1 Estrutura	29
4.1.2 Instalação	31

4.1.3	Uso prático	33
4.2	SimpleCV	40
4.2.1	Estrutura	40
4.2.2	Instalação	42
4.2.3	Uso prático	43
4.3	Conclusão	46
5	ESTUDO DE CASO	48
5.1	Objetivo	48
5.2	Tecnologias utilizadas	48
5.2.1	Python	48
5.2.2	Linux	49
5.3	Implementação	49
5.4	Resultados	55
6	CONCLUSÕES	60
REFERÊNCIAS BIBLIOGRÁFICAS		62

1 INTRODUÇÃO

Os seres humanos possuem sentidos fundamentais para a sua sobrevivência. Graças a isso, eles conseguem interagir e se comunicar com o mundo a sua volta. Nesse contexto, existe o sentido da visão, responsável por detectar imagens e transmití-las ao cérebro para que possam ser interpretadas. Essas imagens declaram a presença de objetos, cada um deles contendo cores e formas. Estes elementos possuem particularidades que os tornam únicos no universo em que estão inseridos. Por meio desses fatores, somados a outras habilidades, os seres humanos conseguem identificar e classificar esses elementos, criando um significado e atribuindo um grau de importância para cada um deles.

Pode-se exemplificar esse processo ao descrever como são identificados e classificados objetos vistos no dia-a-dia. Ao observar um carro, um dos aspectos mais facilmente identificados é a sua cor. Outra tarefa simples é dizer o quanto conservado o carro está, observando se o mesmo possui algum arranhão ou amassado. Pode-se ainda classificar esse carro ao ver se o mesmo é conversível, ou se possui um porta-malas maior que os demais. No fim desta análise, tem-se uma visão completa sobre o carro.

Por ser um dos sentidos mais importantes para os seres humanos, o interesse dos estudiosos em recriar o sistema visual em computadores se tornou uma tarefa importante. Alguns dos fatores responsáveis por tornar esse campo de estudo viável foram a diminuição do preço dos equipamentos e a evolução das tecnologias, possibilitando acesso para o desenvolvimento de pesquisas nesta área.

Apesar dessa ascensão e disseminação, estudos têm mostrado a área de Visão Computacional concernentes ao sistema de visão. Szeliski (2010) descreve que, apesar de haver técnicas confiáveis capazes de realizar tarefas desse sistema, ter um computador capaz de interpretar uma imagem no mesmo nível de percepção de uma criança é um objetivo que ainda apresenta vários desafios.

Como consequência das pesquisas e avanços tecnológicos, o amadurecimento em algoritmos de Visão Computacional tornou possível a criação de ferramentas capazes de auxiliar a implementação eficiente de sistemas nesta área. Com o uso dessas ferramentas, inúmeras contribuições científicas tem sido apresentadas para aprimorar o desenvolvimento desses sistemas.

1.1 Justificativa do trabalho

O tema de Visão Computacional foi escolhido devido ao aumento da disponibilidade de recursos que possibilitem sua aplicação para resolução de problemas atuais. Atualmente, aplicações tem sido desenvolvidas para apresentar soluções onde até então a intervenção humana era indispensável. Dessa forma, disponibilizar um material que reúne conceitos e práticas das principais técnicas da área de Visão Computacional torna o trabalho uma fonte acessível para os iniciantes na área que buscam conhecimento e aprimoramento nesse domínio de estudo.

1.2 Objetivo

Este trabalho tem como objetivo inicial apresentar os principais conceitos de Visão Computacional. A partir do entendimento desses conceitos, almeja-se explorar a utilização de algumas das principais ferramentas para o desenvolvimento de aplicações neste campo de estudo. Finalmente, através de um estudo de caso, este trabalho visa mostrar o desenvolvimento de uma aplicação capaz de detectar e rastrear objetos circulares monocromáticos¹, com base nos conceitos e ferramentas pesquisadas.

1.3 Estrutura do trabalho

Este trabalho é composto por seis Capítulos, organizados conforme descrito abaixo.

O Capítulo 2 apresenta a área de Visão Computacional, através de conceitos e histórico sobre o assunto. São apresentados também domínios onde a Visão Computacional vem se tornando vital. Ainda é exposta superficialmente sua estrutura, descrevendo suas disciplinas.

O Capítulo 3 descreve os conceitos e técnicas relevantes para o entendimento deste trabalho. Nesse tópico, são descritas as áreas de Processamento de Imagem, Segmentação e Reconhecimento.

O Capítulo 4 descreve as principais ferramentas da área de Visão Computacional, com base nas tecnologias escolhidas. Essas ferramentas são apresentadas através de guias para instalação e uso.

O Capítulo 5 apresenta a implementação do estudo de caso, contendo detalhes das tecnologias utilizadas e os principais trechos do código-fonte. Também são apresentados os resultados obtidos a partir da aplicação desenvolvida.

Por fim, no Capítulo 6, são apresentadas as conclusões deste trabalho.

¹É a radiação produzida por apenas uma cor.

2 VISÃO COMPUTACIONAL

É apresentado neste Capítulo uma introdução da área de Visão Computacional, mostrando seu conceito e como o assunto tem se desenvolvido ao longo dos anos. Ainda, são apresentadas áreas de aplicação prática do assunto. Por fim, é apresentada a estrutura desse domínio, separada hierarquicamente por subáreas de estudo.

2.1 Conceito

A área de Visão Computacional tem como principal função recriar o sistema de visão de um ser humano, de modo que seja possível descrever o cenário percebido por esse sistema. Trucco e Verri (1998) destacam que encontrar uma definiçãoicontroversa a respeito de Visão Computacional é uma tarefa difícil, por ser uma disciplina com diferentes perspectivas. Definir um conceito claro sobre o assunto se torna mais fácil ao buscar quais os problemas a Visão Computacional propõe resolver e como isso é feito. Sob a perspectiva de Stockman e Shapiro (2001), a Visão Computacional tem como objetivo tomar decisões úteis a respeito de objetos físicos reais e cenas com base em imagens sensoriais.

Trivedi e Rosenfeld (1989) contextualizam a área de Visão Computacional junto a dois outros campos de estudo: neurofisiologia e psicologia perceptual. A neurofisiologia tenta entender como mecanismos neurais dos sistemas biológicos e sensoriais funcionam. A psicologia perceptual tenta entender os casos psicológicos direcionando a tarefa de percepção. Já a Visão Computacional investiga os casos computacionais e algorítmicos associados a aquisição, processamento e compreensão da imagem.

Nesse contexto é entendido que o campo de estudo de Visão Computacional trata-se de uma ciência que busca tornar possível a compreensão e interpretação de imagens, e assim adquirir informações relevantes a seu respeito, utilizando métodos científicos, comprovando e documentando resultados encontrados. É importante destacar que a área está intimamente relacionada a outros campos de estudo como inteligência artificial, matemática, neurobiologia e física.

2.2 Histórico

As primeiras ideias relacionadas a Visão Computacional são de 1950 em trabalhos de Levialdi, que buscava analisar imagens provenientes de experimentos físicos em câmaras de bolhas através de técnicas de Processamento de Imagem (JOLION, 1994).

Ainda segundo Jolion (1994), muitos pesquisadores da área acreditavam que o problema da Visão Computacional seria resolvido rapidamente. Um dos problemas fundamentais é que uma imagem bidimensional de uma cena não possibilita que se construa uma representação tridimensional da cena em questão, pois não existem equações geométricas suficientes para encontrar todas as incógnitas necessárias à reconstrução.

Apesar de muitos estudos importantes terem sido realizados, poucos frutos foram colhidos. Os principais pesquisadores descobriram que, para simular a percepção na máquina, seriam necessárias mais informações de como o cérebro interpreta as imagens e ferramentas para melhor desempenho no processamento (BIANCHI, 1998).

A partir dos anos 60, o incentivo para pesquisas em novas tecnologias computacionais causado pelas disputas políticas da Guerra Fria, permitiu um avanço em Visão Computacional com maior foco nas áreas de restauração, seletividade e transmissão de imagens. Na década de 70, surgiram os moldes de como esta área é apresentada hoje. Nessa época aumentaram as pesquisas sobre Processamento de Imagem, focadas em ordenação, melhorias na qualidade ou restauração e análise de imagem (ANDREWS; PATTERSON, 1976). Tempos depois, Marr (1982) propôs uma investigação computacional para a representação humana e processamento da informação visual, sendo a primeira metodologia completa para a Visão Computacional (JOLION, 1994). A partir dessas ideias, novos paradigmas surgiram, buscando melhorar a forma como o problema da Visão Computacional era descrito (TARR; BLACK, 1994) (BAJCSY, 1988) (BALLARD; BROWN, 1992) (ALOIMONOS; ROSENFELD, 1991).

2.3 Cenário atual

Ainda hoje, a Visão Computacional é considerada uma ciência em desenvolvimento, pois não foi encontrado um modelo genérico de percepção visual para ser utilizado na prática. A solução encontrada ainda vem sendo a utilização de conjuntos de algoritmos específicos para determinados tipos de tarefas na interpretação de uma imagem. Szeliski (2010) afirma que para projetar algoritmos de Visão Computacional é necessário uma análise do problema proposto e das limitações na representação da imagem formada.

Com o crescimento significativo de estudos na área de Visão Computacional, encontrar soluções que auxiliam o trabalho em outras áreas vem se tornando cada vez mais comum. Hoje já é possível encontrar aplicações que utilizam sistemas de Visão Computacional para realizar

diversas tarefas, a fim de auxiliar ou substituir as que antes eram executadas por pessoas. Dentre essas aplicações encontram-se:

- Reconhecimento óptico de caracteres;
- Inspeção ou controle de qualidade de produtos;
- Construção de modelos 3D;
- Medicina (medicina remota, reconstrução 3D, identificação de padrões orgânicos);
- Direção autônoma;
- Captura de movimentos;
- Vigilância;
- Reconhecimento de biometria;
- Análise de imagens geográficas.

2.4 Estrutura da área de Visão Computacional

O diagrama da Figura 2.1 separa os processos da Visão Computacional hierarquicamente e divididos em três campos. De modo geral, cada um dos componentes apresentados tem uma finalidade dentro da Visão Computacional. As aplicações apresentadas na Seção 2.3 são resultados da união de vários desses componentes, somando ao uso de algoritmos que buscam o melhor resultado para cada área em questão.

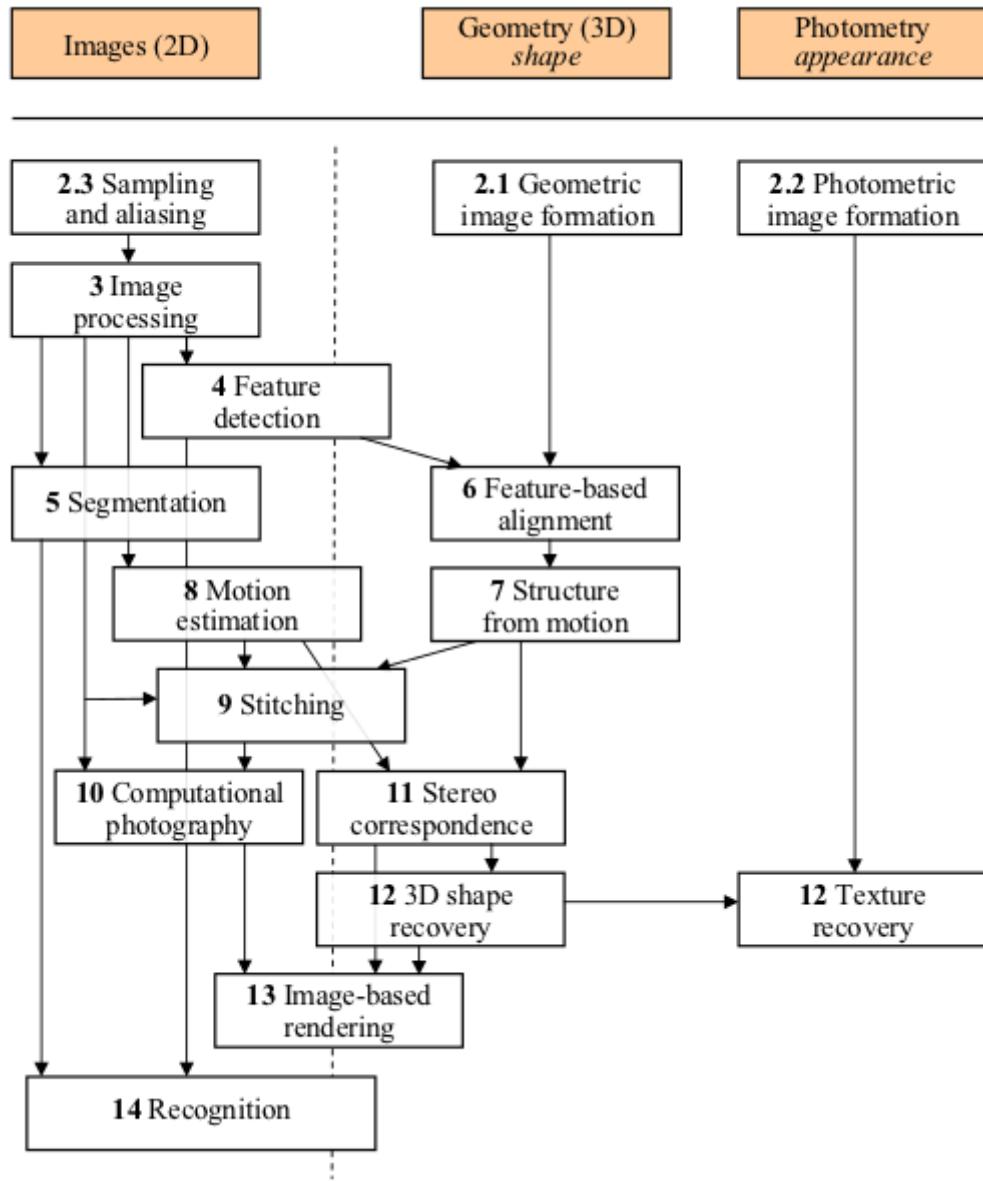


Figura 2.1: Divisão das áreas da Visão Computacional, relacionadas de acordo com suas dependências. Adaptado de (Szeliski, 2010).

Observando da esquerda para direita, podem ser vistos os três campos da Visão Computacional, divididos como: Imagens (*Images*), Geometria (*Geometry*) e Fotometria (*Photometry*). A primeira área (*Images*) cuida de problemas em um universo bidimensional. O tópico sobre Amostragem e Serrilhamento (*Sampling and aliasing*) trata de questões referentes a fotografia digital. A parte de Processamento de Imagem (*Image Processing*) foca na melhoria da qualidade da imagem para manipulações em sistemas de Visão Computacional. Já o processo de Detecção por Características (*Feature Detection*), como o próprio nome sugere, dispõe de técnicas para reconhecimento de características em imagens.

Ainda, existe o processo de Segmentação (*Segmentation*), que aborda técnicas de seg-

mentação de imagens. Em outras palavras, esse processo é responsável por separar regiões relevantes de uma imagem. A Estimativa de Movimento (*Motion Estimation*) é utilizada para alinhamento geométrico e calibração de câmeras e que em alguns momentos também pode ser encontrado em tarefas de um universo 3D (tridimensional). O processo de Junção de Imagens (*Stitching*) cuida da criação de novas imagens a partir de uma ou mais imagens de entrada. Esse procedimento é bastante usado na área de mapas digitais e fotos de satélite, e tal como o processo de Estimativa de movimento, pode também ser utilizado em ambientes 3D.

Diretamente ligado ao processo de Junção de Imagens, encontra-se a Fotografia Computacional (*Computational Photography*), que refere-se a captura, processamento e técnicas de manipulação da imagem computacional que melhoram ou aumentam a capacidade da fotografia digital. Como último processo de imagens 2D (bidimensional), temos o processo de Reconhecimento (*Recognition*), tendo com tarefa principal analisar uma cena e reconhecer objetos.

Na próxima coluna, encontra-se os processos que tratam, em sua maioria, de casos em universo 3D. A Formação da Imagem Geométrica (*Geometric image formation*) trata pontos, linhas e planos e como esses podem ser mapeados em imagens. A parte de Alinhamento Baseado em Características (*Feature-based alignment*) trata das características recuperadas de imagens em processos anteriormente utilizados e busca identificá-las em outras diferentes imagens. Feito o processo de Alinhamento Baseado em Características, pode-se então utilizar o conjunto de técnicas no processo de Estrutura a Partir do Movimento (*Structure from motion*), que cuida de encontrar estruturas tridimensionais de um objeto através de sinais de movimento continuamente.

Prosseguindo, está o processo Correspondência Estéreo (*Stereo correspondence*), que pode ser entendido como a parte da Visão Computacional que analisa objetos em uma cena, sendo tratados por diversos pontos de vista. Na mesma linha de reconstrução de modelos 3D, está o componente de recuperação de formas 3D (*3D shape recovery*). Diferente da Correspondência estéreo, esse componente funde diversas imagens de profundidade e de alcance. Por fim, está o componente da Renderização Baseada em Imagem (*Image-based rendering*), diretamente ligado aos dois universos. O conjunto de técnicas nesse tópico cuida da geração de modelos 3D a partir de imagens 2D.

Além dessas áreas núcleo dos sistemas de Visão Computacional, existe a Fotometria, classe de técnicas de medição, que realiza a medição de objetos do mundo real usando apenas imagens. Em seus processos, estão a Formação da Imagem Fotométrica (*Photometric image formation*) e a Recuperação de Texturas (*Texture recovery*). O primeiro, descreve como os sensores geram valores de cor e intensidade, e como esses são formados em uma imagem. O segundo, está ligado a aparência dos objetos da superfície, após terem sido gerados como modelo 3D.

No Capítulo seguinte, são detalhados os principais processos de Visão Computacional.

Por se tratar de disciplinas essenciais para o entendimento da área de Visão Computacional, foram escolhidos processos referentes ao universo bidimensional. A abordagem desses tópicos tem como finalidade servir de base conceitual para o desenvolvimento do estudo de caso, exposto no Capítulo 5.

3 PRINCIPAIS PROCESSOS DE VISÃO COMPUTACIONAL

Este Capítulo contextualiza os principais processos da área de Visão Computacional. São descritas as técnicas mais utilizadas em cada um desses processos visando atingir o entendimento necessário para esse trabalho.

3.1 Processamento de Imagem

Processamento de Imagem é um processo da Visão Computacional que atua na maior parte dos projetos do universo bidimensional. Esse processo consiste em tratar a imagem buscando obter o máximo de qualidade possível sem se importar com a informação em si. Segundo Jain (1989), o termo Processamento de Imagem refere-se ao processamento de uma imagem 2D através de um computador digital.

Embora essa área seja abordada como um processo de Visão Computacional, é normal encontrar algumas publicações que tratam esses dois processos como ciências isoladas. Uma dessas distinções é destacada por Aloimonos e Rosenfeld (1991), que atribuem como diferença básica o fato das tarefas envolvendo Processamento de Imagem serem de mais baixo nível que a Visão Computacional. Além disso, a Visão Computacional está relacionada com a percepção do ambiente, enquanto que o Processamento de Imagem é uma tarefa que possibilita essa percepção.

Freitas (1998) também relata que, apesar de ter em comum a imagem digital como fim ou como meio, ao processamento digital de imagem cabe a aquisição e a manipulação da informação adquirida na forma de *pixels*¹. Já Visão Computacional é uma área voltada à análise de imagens, interpretação de características e reconstituição do modelo de um objeto ou de uma cena.

Segundo Filho e Neto (1999), a etapa de aquisição de imagem tem como função converter uma imagem em uma representação numérica adequada para o processamento digital subsequente. Na etapa em que as imagens são tratadas, operações lógicas são realizadas em todos os *pixels*, normalmente expressas sob forma algorítmica.

¹Menor ponto que forma uma imagem digital.

3.1.1 Representação de Cores

A forma como os seres humanos interpretam as cores varia de acordo com os seus sistemas visuais. Para representar essas cores em sistemas artificiais, criou-se os modelos de representações das cores. Segundo Filho e Neto (1999), o objetivo principal desses modelos é permitir a especificação de cores em um formato aceito por todos. Conceitualmente, os modelos de cores são representados através de sistemas tridimensionais de coordenadas, onde cada eixo refere-se a uma cor primária (FOLEY et al., 1990).

3.1.1.1 Modelo RGB

O modelo de cores RGB (*red, green, blue*) é um sistema aditivo que possui a capacidade de representar a percepção humana com um alto grau de semelhança, além de conseguir iludir esta mesma percepção, fazendo com que as pessoas acreditem que possam enxergar várias cores em uma única reprodução (BUNTING, 1998). Por possuir essas características, este modelo foi adotado como padrão para reprodução de cores de diversos dispositivos eletrônicos, como câmeras digitais e monitores de vídeo.

Baseado em um sistema de coordenadas cartesianas, representado por um cubo, o modelo RGB possui como cores primárias o vermelho, o verde e o azul, que encontram-se localizadas uma em cada vértice do cubo (FILHO; NETO, 1999). Nos demais vértices, encontram-se combinações de cores a partir das cores primárias. A tonalidade das cores variam também de acordo com sua intensidade, normalmente representadas através de um número no intervalo de 0 a 255. Intensidades mínimas, apresentam cores mais escuras enquanto que intensidades máximas apresentam cores mais claras.

A Figura 3.1 apresenta essa forma de representação do modelo, utilizando o número 1 ao invés do 255 para representar a tonalidade máxima de uma cor. Além dos vértices representando as cores primárias, tem-se ainda mais dois vértices para representar as cores preta e branca, formados pela combinação das cores primárias em intensidades mínimas e máximas respectivamente. Além desses vértices, encontram-se também os vértices derivados da combinação de duas cores primárias. A cor magenta é resultado da combinação de vermelho com azul, enquanto que a cor amarela é combinação do vermelho com verde. Por fim tem-se o ciano, resultado da combinação da cor verde com azul.

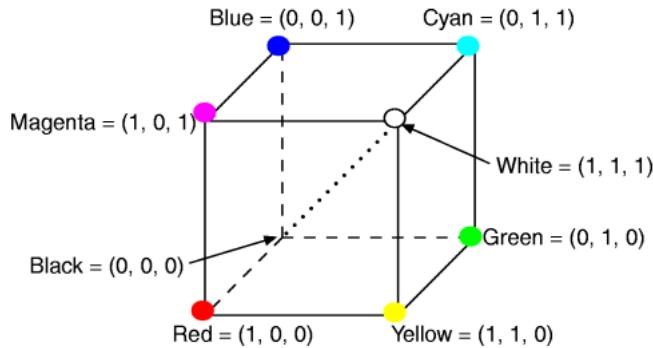


Figura 3.1: Representação do modelo RGB através do desenho de um cubo (CARVALHO, 2006).

3.1.1.2 Modelo HSV

O modelo HSV apresenta uma forma diferente de representar as cores, onde se tem a matiz (*hue*), saturação (*saturation*) e valor (*value*). A matiz é responsável por determinar a tonalidade dominante de uma área, ou seja, a família em que aquela cor está inserida. Esse componente é medido através de valores entre 0 e 360. A saturação mede a pureza da cor da área. Já o valor define a luminância da cor. Nas modalidades de saturação e valor, a variação pode ocorrer entre 0% e 100%.

Conforme visto na Figura 3.2, o espaço de cor HSV é representado por um cone. O lado circular do cone apresenta as tonalidades, normalmente representadas por um ângulo, onde cada ângulo corresponde a uma cor no cone. A saturação é representada pela distância desde a borda ao centro do círculo, de modo que, quanto menor o valor de saturação, menor a presença de tons de cinza na imagem e mais próxima da cor branca. O brilho é determinado pela posição vertical em cores do cone. Na extremidade pontiaguda do cone, não há nenhum brilho, portanto a tonalidade das cores se aproximam da cor preta. No final do cone, na base, estão as cores mais brilhantes (CARDANI, 2001).

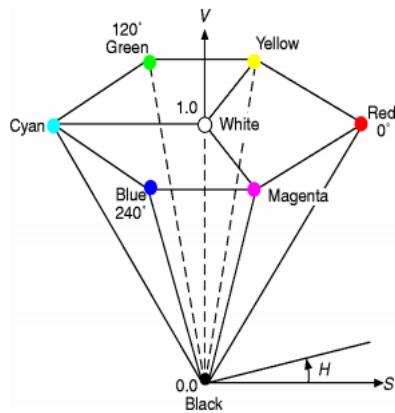


Figura 3.2: Representação do modelo HSV através do desenho de um cone (CARVALHO, 2006).

3.1.2 Histograma

Originalmente criados para melhor análise de dados estatísticos, os histogramas consistem em retângulos contíguos com base nas faixas de valores da variável e com área igual à frequência relativa da faixa. A altura de cada retângulo é denominada densidade de frequência definida pelo quociente da frequência relativa pela amplitude da faixa (CECIRE, 2002).

Em Processamento de Imagem a técnica de histograma é muito utilizada para indicar o percentual de diferentes tons de cinza dos *pixels* em uma imagem através de um gráfico de barras, possibilitando dessa forma uma interpretação visual da distribuição dos valores dos tons de cores da imagem (FILHO; NETO, 1999). Através da visualização do histograma de uma imagem é possível obter uma indicação de sua qualidade quanto ao nível de contraste e quanto ao seu brilho médio, determinados pela linha horizontal do gráfico. Entretanto, quando uma imagem é colorida, torna-se necessário a decomposição da mesma para que seja avaliado o histograma de cada um dos componentes da imagem separadamente (Figura 3.3).

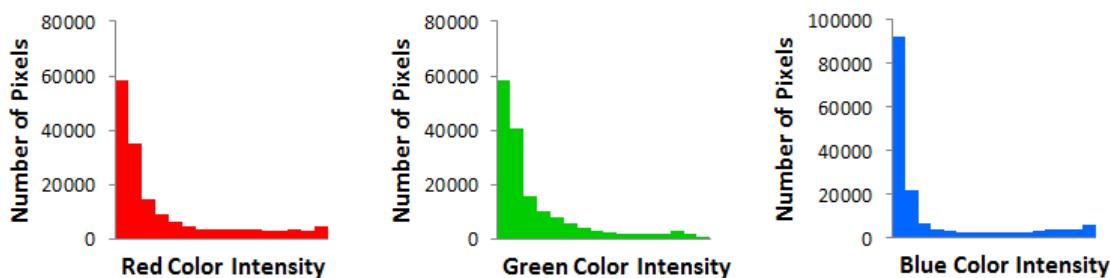


Figura 3.3: Exemplo de gráficos de histogramas divididos nos canais de cores vermelho, verde e azul (TEAM, 2012).

3.2 Segmentação

Além de ser um processo de Visão Computacional, a Segmentação possui uma ligação com a fase de Processamento de Imagem, visto que a qualidade da visualização interfere muito devido à dificuldade de se interpretar as características da imagem. O processo de Segmentação consiste em dividir a imagem em diferentes regiões, buscando separar áreas da imagem, que serão processados posteriormente no processo de Reconhecimento (PUZICHA; HOFMANN; BUHMANN, 1999).

Nesse processo, existe uma etapa denominada *clustering*, fase de identificação da imagem, que pode ser de dois tipos: supervisionado ou não-supervisionado (ZHOU; HUANG, 2003). Os processos supervisionados possuem conhecimento prévio sobre quais informações serão classificadas e agrupadas. Em contrapartida, os processos não-supervisionados não requerem nenhuma informação anterior sobre as informações que devem ser agrupadas.

A etapa de Segmentação pode ser realizada utilizando métodos baseados em histograma (ROTHER et al., 2006), onde são realizados cálculos utilizando um histograma a partir de todos os *pixels* da imagem, e os picos e vales do histograma podem ser utilizados para definir um número de agrupamentos a ser encontrado. Além disso, esse processo também pode ser realizado através de lógica espacial (ZABIH; KOLMOGOROV, 2004), dividindo ou fundindo imagens a partir de pontos semelhantes.

Devido à dificuldade em fazer com que o computador reconheça os padrões e agrupe as imagens significativas, a Segmentação se torna um dos processos mais complexos da área de Visão Computacional. Tomando como base uma imagem que possui diversos objetos distintos, fazer com que o computador interprete e consiga separar esses objetos se tornou um desafio para os técnicos. Consequentemente, surgiram várias técnicas de agrupamentos que buscam no fim o mesmo resultado, ou seja, uma coleção de segmentos que quando combinados formam uma imagem inteira.

Existem várias técnicas e métodos para segmentar uma imagem tanto estática quanto dinâmica. A seguir serão apresentados alguns dos principais métodos de agrupamentos utilizados pelo processo de Segmentação.

3.2.1 Segmentação por região

A detecção por região pode ser feita extraíndo uma região ou dividindo a imagem em várias regiões diferentes. As regiões normalmente apresentam como característica a continuidade do nível de cinza de seus pixels e podem ser detectadas pelos métodos de *thresholding* ou dividindo e fundindo segmentos da imagem.

3.2.1.1 Thresholding

Várias técnicas podem ser utilizadas para alcançar a seleção de segmentos que formam os objetos buscados pelo sistema. *Thresholding*, também conhecido como limiarização, é o método de Segmentação de fácil compreensão, se comparado com as demais. O método consiste em utilizar imagens pós-processadas e convertidas para tonalidades de cinza em sequência, criando um limiar onde é separada da imagem principal uma imagem segmentada.

O *thresholding* é um método baseado em histograma, onde o limiar de separação pode ser encontrado nos vales entre um pico e outro do histograma (PADILHA, 1999).

3.2.1.2 Dividir e fundir

Como mencionado no método de *thresholding*, a maneira mais simples possível para segmentar uma imagem é selecionando um limiar e então calcular os componentes ligados. Entretanto, um único limiar raramente é o suficiente para a imagem por causa da iluminação e variações estatísticas do objeto (Szeliski, 2010). O método de dividir e fundir agrupa em segmentos os *pixels* que se encontram em regiões de características semelhantes, dividindo ou fundindo a imagem se preciso.

3.2.2 Segmentação por contornos

O contorno pode ser definido pela variação dos níveis de cinza entre duas regiões semelhantes. O método de detecção de contornos da imagem permite encontrar regiões com uniformidades de valores. Nesse método de Segmentação é localizado a região da imagem que possui variações nos valores. O objetivo do método é localizar as bordas ou curvas dos objetos nas imagens em forma de contornos finos e sem interrupções (Szeliski, 2010).

Também é possível segmentar uma imagem por contornos através da detecção de pontos. Esse método é muito utilizado em objetos em movimento, avaliando a variação em suas direções principais (Antunes, 1999).

3.2.3 Segmentação por textura

A detecção por textura busca agrupar subconjuntos que possuam aproximadamente as mesmas características em qualquer lugar da imagem (Antunes, 1999). Essa técnica possui métodos como o estatístico, baseando-se na distribuição dos níveis de cinza da imagem para definir uma matriz que representa essa distribuição através da imagem, e então, a partir dessa matriz, identificar a probabilidade de um conjunto de *pixels* compor uma determinada textura.

3.3 Reconhecimento

Realizar a análise e reconhecimento de objetos em uma cena tem sido uma das tarefas mais desafiadoras no domínio da Visão Computacional. Apesar de ser uma tarefa facilmente executada por crianças, conseguir que um computador realize a mesma tarefa não é igualmente simples (Szeliski, 2010). A dificuldade no Reconhecimento se dá pela infinidade de classificações que os seres humanos empregam. Ainda é possível acontecer de objetos da mesma classe conterem atributos distintos, dificultando ainda mais a exatidão na fase de Reconhecimento e tornando improvável a análise através de uma base de dados de exemplos.

O objetivo da etapa de Reconhecimento é fazer com que o sistema classifique as informações adquiridas das etapas anteriores baseada em um conhecimento proposto ou com experiência em aprendizado.

Em muitos casos, o Reconhecimento depende do contexto onde o objeto está inserido e os elementos da cena. Se o objeto procurado for de conhecimento do computador, é possível procurar por pontos característicos e verificar se os mesmos se alinham de forma geométrica. Existem vários modelos utilizados para realizar um reconhecimento em uma imagem. Em seus estudos, Weber (2000) apresenta diversos modelos de Reconhecimento balanceados por funções probabilísticas.

Um dos modelos mais simples apresentados é caracterizado por um objeto constituído pela formação de várias peças que podem possuir características distintas, como aparência, forma e escala relativa. A partir da segmentação das peças do objeto é iniciada a fase de detecção, que em uma primeira etapa utiliza detectores especializados para cada uma das peças formadas. Após as peças serem reconhecidas e suas informações armazenadas, a segunda etapa é iniciada para a detecção do objeto formado através da união das informações de cada peça recolhida. Outra etapa importante no processo é a localização do objeto na imagem, que é realizada usando as peças detectadas que correspondem ao objeto em primeiro plano na imagem.

Através desse modelo, conseguiu-se um reconhecimento razoavelmente rápido e simples. Além disso, a partir desse modelo, Weber propôs outros modelos que utilizavam cálculos com funções buscando um custo de processamento menor.

4 BIBLIOTECAS DE VISÃO COMPUTACIONAL

No mercado de software atual, existem bibliotecas que auxiliam o desenvolvimento de sistemas de Visão Computacional, direcionadas em diversos focos. Muitas dessas ferramentas têm como objetivo suportar questões específicas do campo de estudo de Visão Computacional.

Apesar de não ser o foco deste trabalho, algumas dessas bibliotecas merecem destaque. Uma delas se trata do VLFeat¹, biblioteca que possui em sua implementação algoritmos populares de Visão Computacional. Outra biblioteca a ser destacada é o VIGRA², que enfatiza o uso de algoritmos flexíveis, construídos usando programação genérica (MUSSER; STEPANOV, 1989).

Outras bibliotecas por sua vez possuem recursos que ampliam seu uso em diversas áreas no desenvolvimento de sistemas de Visão Computacional. Essas bibliotecas são suportadas por diversas linguagens, tanto de baixo nível como de alto nível. Visando a boa produtividade e simples manuseio, optou-se pela abordagem de linguagens de programação de alto nível. Por se tratar de uma linguagem de simples utilização e aprendizado, foi escolhida a linguagem de programação Python neste trabalho. Essa linguagem conta com diversos recursos de uso simples e possui boa integração com as bibliotecas mais utilizadas na área de Visão Computacional.

No grupo de bibliotecas suportadas pela linguagem Python, a que mais se destaca é o OpenCV. Através do uso de interfaces, a utilização da ferramenta tem facilitado o desenvolvimento de aplicações de Visão Computacional, além de permitir o melhor entendimento da área. Existe também o *framework* SimpleCV, com a missão de simplificar ainda mais o uso das interfaces do OpenCV para Python.

Neste Capítulo, são mostradas a utilização e instalação dessas ferramentas de uso geral para o desenvolvimento de sistemas de Visão Computacional suportadas pela linguagem Python.

¹<http://github.com/mmmikael/vlfeat>

²<http://hci.iwr.uni-heidelberg.de/vigra>

4.1 OpenCV

Considerada como uma das mais completas bibliotecas na área de Visão Computacional (THORNE, 2009) (SEYDOUX, 2010), o OpenCV³ se destaca pelo seu uso abrangente, possuindo mais de 500 funções que implementam técnicas em diversas áreas de Visão Computacional. Dentre as áreas que a biblioteca atua estão: Processamento de Imagem, Segmentação, Transformação, Rastreamento, Calibração de Câmera, entre outras.

Desenvolvida pela Intel, a biblioteca do OpenCV foi criada para prover uma infraestrutura para criação rápida de aplicações de Visão Computacional. Por possuir o foco em aplicações em tempo real, seu código é escrito na linguagem C e C++. Essas linguagens são mais próximas das linguagens de máquina, desta forma, tendem a ter um desempenho superior às linguagens mais atuais, que necessitam ser interpretadas por uma máquina virtual. A utilização do OpenCV pode ainda ser flexibilizada através de interfaces, criadas para se comunicar com outras linguagens de programação como Java, Python e Ruby.

O uso do OpenCV pode ser otimizado com a utilização da biblioteca proprietária *multi-core ready IPP* (*Integrated Performance Primitives*), que possui funções altamente otimizadas para multimídia, processamento de dados e aplicações de comunicação (MOLENAAR, 2010). Em estudos realizados por Bradski e Kaehler (2008), a biblioteca é comparada trabalhando de forma independente e integrada com a biblioteca IPP. Essa comparação é apresentada na Figura 4.1, onde é possível observar o ganho razoável de performance da biblioteca ao ser utilizada em conjunto com a IPP.

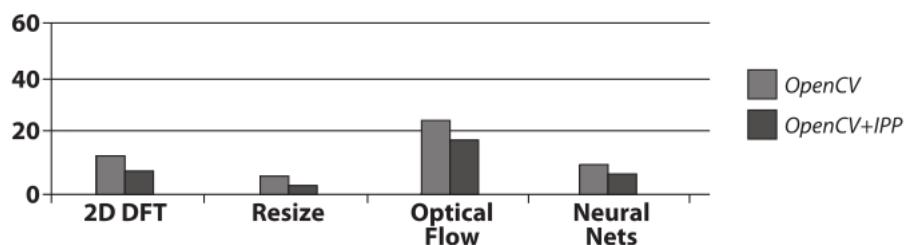


Figura 4.1: Comparativo de desempenho entre a ferramenta OpenCV com e sem a biblioteca proprietária da Intel. Adaptado de (BRADSKI; KAEHLER, 2008).

4.1.1 Estrutura

Baseando-se em relatos de Bradski e Kaehler (2008), o OpenCV é estruturado em cinco componentes principais. As partes de mais alto nível são divididas em: *CV*, *MLL* e *HighGUI*. Como é possível ver na Figura 4.2, no componente *CV* está toda a parte de Processamento de Imagem e algoritmos de Visão Computacional. O componente *MLL* contém ferramentas de *clustering* e métodos de classificadores estatísticos. A parte de *HighGUI*, como o nome sugere,

³<http://opencv.willowgarage.com>

contém componentes de interface com usuário, assim como rotinas de entrada e saída de vídeos e imagens. Da mesma forma, no *CXCORE* estão contidas todas as estruturas de dados usadas no OpenCV. Nesse componente também existem algoritmos para manipulação de arquivos XML, sendo possível realizar leitura e gravação de dados. Outro recurso reunido nesse componente bastante utilizado pelos desenvolvedores são as funções de desenho. Através delas, é possível, por exemplo, destacar em uma imagem ou vídeo uma região de interesse (ROI).

Além dos componentes descritos, o OpenCV ainda conta com um componente auxiliar, nomeado de *CvAux*, contendo métodos experimentais. A ideia desse módulo é conter funções de Visão Computacional em fases de testes que podem ser usadas pelos desenvolvedores, e com a aceitação ou não de cada função, a mesma é inserida nos módulos principais como uma função oficial do OpenCV.

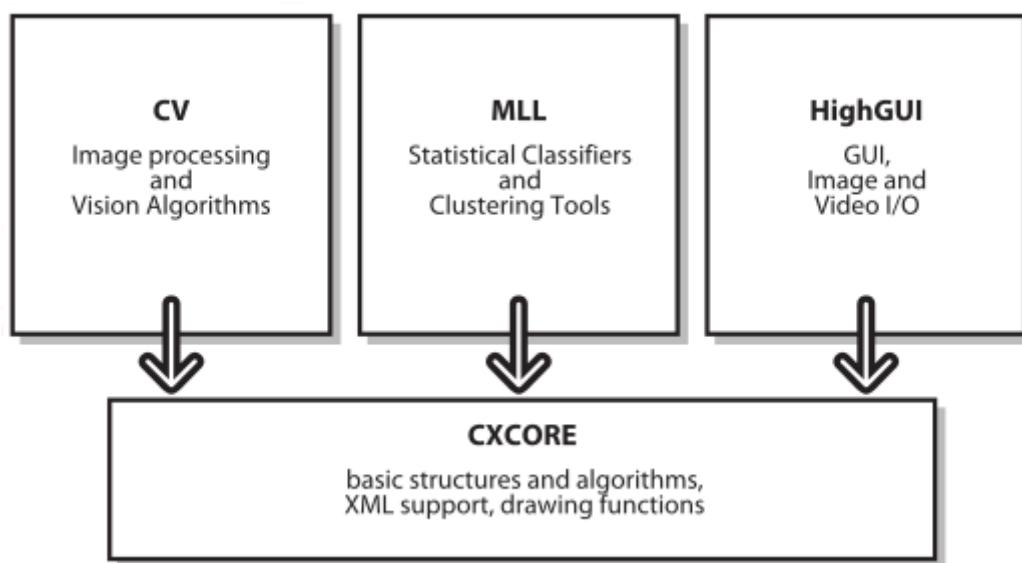


Figura 4.2: Estrutura da biblioteca OpenCV (BRADSKI; KAEHLER, 2008).

Como interfaces do OpenCV para linguagem Python, existem duas que cumprem esse papel. A primeira, declarada como *cv2.cv*, é uma implementação mais antiga e trabalha com classes da própria interface. A segunda, nomeada como *cv2*, possui mais dinamicidade em seu uso, permitindo atribuições a tipos de dados nativos do Python. Exemplificando a comparação das duas interfaces, ao carregar uma imagem em ambas, observa-se a diferença em como cada uma delas trata os dados (Código 1).

Código 1: O tipo das variáveis se diferem devido o uso de *bindings* diferentes.

```

1 In [1]: img = cv.LoadImage('logo_iff.png')
2 In [2]: img
3 Out[2]: <iplimage nChannels=3 width=291 height=385 widthStep=876 >
4
5 In [3]: img2 = cv2.imread('logo_iff.png')
  
```

```

6 In [4]: img2
7 Out[4]: array([[[255, 255, 255], ..., [255, 255, 255]]], dtype=uint8)
8
9 In [5]: type(img2)
10 Out[5]: <type 'numpy.ndarray'>

```

4.1.2 Instalação

Nas pesquisas realizadas, foi utilizado o sistema operacional Ubuntu em sua versão 11.10, em conjunto da biblioteca OpenCV em sua versão 2.4.2 e a linguagem de programação Python em sua versão 3.0. Buscando uma instalação mais completa e com a maioria dos recursos disponíveis, procurou-se reunir a maior parte das dependências da ferramenta. Contudo, não foi possível realizar a instalação da biblioteca IPP, por se tratar de uma biblioteca proprietária.

Antes de iniciar a instalação do OpenCV, recomenda-se a remoção das bibliotecas usadas pela ferramenta que já estejam instaladas no sistema operacional (ERALP, 2012). O Código 2 apresenta a remoção das bibliotecas comumente encontradas no sistema operacional Linux Ubuntu 11.10.

Código 2: Remoção de pacotes instalados por padrão no sistema operacional Linux Ubuntu 11.10.

```
sudo apt-get remove ffmpeg x264 libx264-dev
```

Para a instalação da biblioteca, é necessário baixar o pacote compactado em sua página oficial. A versão usada no trabalho foi a 2.4.2. A instalação da biblioteca é realizada através de uma ferramenta de construção de sistemas, o CMake (Código 3). Além disso, existem outras dependências que precisam ser instaladas antes da execução da instalação do OpenCV (Código 4).

Código 3: Instalação da ferramenta CMake.

```
sudo apt-get install cmake
```

Código 4: Instalação de dependências do OpenCV.

```

sudo apt-get install build-essential
sudo apt-get install python-dev python-numpy python-sphinx
sudo apt-get install libgtk2.0-0 libgtk2.0-dev
sudo apt-get install libjpeg8 libjpeg8-dev

```

```

sudo apt-get install libgstreamer0.10-0 libgstreamer0.10-dev gstreamer0.10-tools \
    gstreamer0.10-plugins-base libgstreamer-plugins-base0.10-dev \
    gstreamer0.10-plugins-good gstreamer0.10-plugins-ugly gstreamer0.10-plugins-bad \
    gstreamer0.10-ffmpeg
sudo apt-get install libjasper-dev
sudo apt-get install libavcodec-dev
sudo apt-get install libdc1394-22-dev
sudo apt-get install libavformat-dev
sudo apt-get install libv4l-dev
sudo apt-get install libswscale-dev

```

O FFmpeg, conjunto de ferramentas para gravação, conversão e criação de stream de áudio e vídeo, uma das dependências do OpenCV, necessita da instalação prévia de outras bibliotecas para seu funcionamento. Essas dependências são descritas no Código 5. Após a instalação destas dependências, instala-se o FFmpeg (Código 6).

Código 5: Instalação das dependências do FFmpeg.

```

sudo apt-get install libfaac-dev libjack-jackd2-dev libmp3lame-dev \
    libopencore-amrnb-dev libopencore-amrwb-dev libSDL1.2-dev libtheora-dev \
    libva-dev libvdpau-dev libvorbis-dev libx11-dev libxf86vidcore-dev \
    texi2html yasm zlib1g-dev libx264-dev
mkdir ~/src
cd ~/src
wget ftp://ftp.videolan.org/pub/videolan/x264/snapshots/ \
    x264-snapshot-20120820-2245-stable.tar.bz2
tar xvf x264-snapshot-20120820-2245-stable.tar.bz2
cd x264-snapshot-20120820-2245-stable/
./configure --enable-static
make
sudo make install

```

Código 6: Instalação do FFmpeg.

```

cd ~/src
wget http://ffmpeg.org/releases/ffmpeg-0.11.tar.bz2
tar xvf ffmpeg-0.11.tar.bz2
cd ffmpeg-0.11
./configure --enable-gpl --enable-libfaac --enable-libmp3lame \
    --enable-libopencore-amrnb --enable-libopencore-amrwb --enable-libtheora \
    --enable-libvorbis --enable-libx264 --enable-libxvid --enable-nonfree \
    --enable-postproc --enable-version3 --enable-x11grab
make
sudo make install

```

Outra dependência do OpenCV é a biblioteca V4L, uma interface de captura de vídeo para programação de aplicativos para Linux. Pode ser baixada e instalada através dos comandos mostrados no Código 7.

Código 7: Instalação do V4L.

```
cd ~/src
wget http://www.linuxtv.org/downloads/v4l-utils/v4l-utils-0.8.8.tar.bz2
tar xvf v4l-utils-0.8.8.tar.bz2
cd v4l-utils-0.8.8
make
sudo make install
```

Agora é possível executar a instalação do OpenCV. Após realizada a descompactação do pacote, na pasta do OpenCV, crie um diretório onde estarão contidos os arquivos (Código 8).

Código 8: Comandos iniciais para instalação do OpenCV.

```
mkdir release
cd release
```

O comando *cmake* irá construir os executáveis da instalação do OpenCV (Código 9).

Código 9: Execução da instalação do OpenCV através do CMake.

```
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local \
-D BUILD_NEW_PYTHON_SUPPORT=ON -D BUILD_EXAMPLES=ON ..
```

Ao executar o comando acima, é possível visualizar quais dependências foram reconhecidas. No geral, as mais importantes são: GTK+ 2.x, FFMPEG, GStreamer e V4L/V4L2. Por fim, é necessário executar os comandos que irão de fato instalar a biblioteca na máquina (Código 10).

Código 10: Comandos finais para instalação do OpenCV.

```
make
sudo make install
```

4.1.3 Uso prático

O OpenCV conta com diversos exemplos em várias linguagens em seu subdiretório (*OpenCV-2.4.2/samples*). Sua documentação também é bastante rica, e pode ser acessada através de seu site⁴, contendo descrição e exemplos dos recursos da biblioteca.

⁴<http://opencv.itseez.com>

Tomando como base esse material, foi selecionado um conjunto expressivo de comandos da biblioteca, para que seja possível uma rápida familiarização com a ferramenta. Começando com um exemplo básico, é possível carregar e mostrar uma imagem na tela através dos comandos do Código 11.

Código 11: Carregamento básico de imagem utilizando OpenCV.

```

1 import cv2
2 img = cv2.imread('caminho_da_imagem')
3 cv2.imshow('titulo', img)
4 cv2.waitKey(0)

```

A primeira linha importa a interface do OpenCV, que no caso é a *cv2*. Caso seja necessário o uso da interface alternativa, basta substituir *cv2* por *cv2.cv*. O Código 12 mostra o conteúdo do *array* da imagem carregada, com os valores RGB de cada *pixel*.

Código 12: Comando para visualização dos valores atribuidos a variável *img*.

```

1 print img

```

A saída será um *array* (Código 13), onde para cada grupo de valores RGB é montado um *array*, que por sua vez é agrupado e compõe uma linha vertical da imagem. Então cada *array* é agrupado e assim se forma um *array* que representa por completo a imagem carregada. Dessa forma, se torna flexível a realização de diferentes tipos de operações, como recortar e converter tipos (Código 14).

Código 13: Saída do *array*, representando uma imagem carregada pela interface *cv2*.

```

1 [[255 255 255]
2 [255 255 255]
3 [255 255 255]
4 ...,
5 [255 255 255]
6 [255 255 255]
7 [255 255 255]]
8 ...,
9 [[255 255 255]
10 [255 255 255]
11 [255 255 255]
12 ...,
13 [255 255 255]
14 [255 255 255]
15 [255 255 255]]]

```

Código 14: Recorte na imagem pode ser realizado através de manipulações com *arrays*.

```
1 parte_um = img[:len(img)/2]
2 parte_dois = img[len(img)/2:]
```

Pelo fato da variável *img* ser de um tipo conhecido pela linguagem Python, as manipulações com seus valores se tornam simples. Para salvar as variáveis, cada uma contendo metade da imagem, utiliza-se o que é exposto no Código 15.

Código 15: Imagens são salvas utilizando o método do OpenCV que realiza essa operação.

```
1 cv2.imwrite('DIR_PROJETO/images/img1.jpg', parte_um)
2 cv2.imwrite('DIR_PROJETO/images/img2.jpg', parte_dois)
```

Quanto as conversões de espaço de cor, a biblioteca possui funções e variáveis próprias para essa tarefa. Por padrão, a imagem carregada é armazenada na ordem BGR, que é contrária a mais conhecida, RGB. Através do comando *cvtColor()* é possível obter, por exemplo, essa ordem de cores (Código 16).

Código 16: Operação para inversão da ordem do espaço de cor BGR para RGB.

```
1 img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

Em algumas documentações da biblioteca, é comum encontrar os nomes das variáveis usadas para transformação do espaço de cor precedidos por CV, principalmente as que se referem a linguagem C. Contudo, na interface para Python, os nomes das variáveis referentes a esse tipo de operação se encontram com o prefixo COLOR. Uma lista completa de todas as variáveis usadas em conjunto com o método *cvtColor()* pode ser encontrada na documentação oficial da biblioteca, ou então no próprio ambiente de configuração, caso o mesmo tenha a capacidade de autocompletar o código a ser escrito.

Outras operações bastante usadas com a ferramenta, são relacionadas a análises estruturais e descritores de formas. Através de contornos, que segundo Bradski e Kaehler (2008) tratam-se de uma lista de pontos que representam, de uma forma ou outra, uma curva em uma imagem, o OpenCV consegue recuperar informações de objetos em uma imagem. Em outras

palavras, o seu limite. Essa técnica se torna muito poderosa em ocasiões em que é preciso detectar e reconhecer objetos em imagens e vídeo. O Código 17 mostra um exemplo de como obter os contornos de uma imagem.

Código 17: Recuperação de contornos dos objetos contidos na imagem com o logotipo do IFF.

```

1 img = cv2.imread('DIR_PROJETO/images/logo_iff.png')
2 img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
3 ret, thresh = cv2.threshold(img_gray, 127, 255, cv2.THRESH_BINARY)
4 contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

```

A primeira linha, como visto anteriormente, carrega uma imagem, que no exemplo, será o logotipo do Instituto Federal Fluminense (Figura 4.3). Então a imagem é convertida para escala de cinza, para que possa ser usada no próximo método, *threshold()*. Esse método delimita uma parte da imagem de acordo com os parâmetros passados. No exemplo, é especificado pelos parâmetros manter os *pixels* que estiverem entre 127 e 255, colocando-os em preto. Os que não forem deste grupo, são retirados. Esse método transforma uma imagem em escala de cinza em uma imagem binária⁵, mantendo apenas os *pixels* que sejam de interesse. Com esses parâmetros, obtém-se uma imagem com apenas parte do logotipo (Figura 4.4(a)). Para que seja possível recuperar todos os seus elementos, faz-se necessário alterar os parâmetros passados. Tendo a parte vermelha 234, 31, 42 como valores RGB, seu valor em escala de cinza é 93, baseando-se na fórmula de conversão de RGB para escala de cinza (KUMAR; VERMA, 2010). Já os quadrados verde, possuem valor 155 em escala de cinza e 115, 196 e 48 como valores RGB. Portanto, para recuperar todo o logotipo, altera-se os parâmetros para 155 e 255. Os resultados são vistos na Figura 4.4(b).

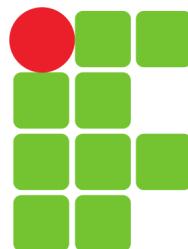
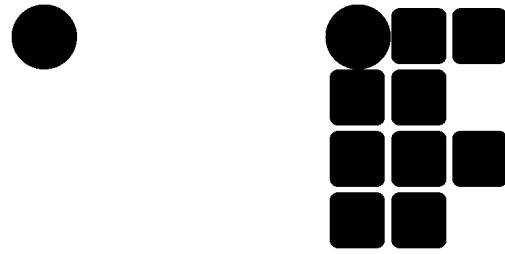


Figura 4.3: Logotipo do Instituto Federal Fluminense.

⁵Imagem digital com todos os valores de *pixel* 0 ou 1.



(a) Logotipo parcialmente binarizado.
(b) Logotipo totalmente binarizado.

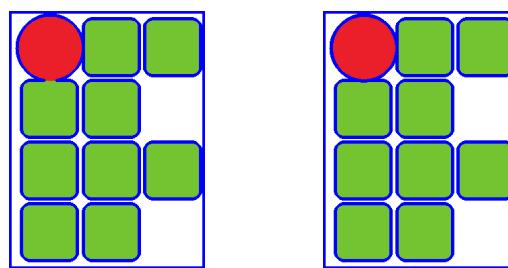
Figura 4.4: Limiarização com diferentes *thresholds* no logotipo do IFF.

Realizada a etapa de *threshold*, pode-se então recuperar os contornos dos elementos em preto na imagem. Aplicando o método *findContours()*, descrito na linha 4 do Código 17, é retornada uma lista com os limites de todos os elementos. Através do Código 18, é possível ver o comando utilizado para saber a quantidade de elementos que o OpenCV recuperou.

Código 18: Comando para imprimir saída, contendo o número de elementos da figura.

```
| print len(contours)
```

Como o exemplo utiliza uma imagem binarizada com todos os objetos da imagem, o resultado é dez, representando a quantidade de elementos do *array*. Apesar de haver dez elementos em preto identificáveis, a ferramenta uniu dois deles, e obteve o contorno do fundo branco, como é possível ver na Figura 4.5(a), após os contornos serem desenhados. Entretanto, pelo fato dos elementos unidos serem de cores diferentes, é possível separá-los, aplicando dois *thresholds* em momentos diferentes do código. Realizada a etapa de recuperação dos contornos, os mesmos são desenhados na imagem. O resultado final é exposto na Figura 4.5(b), enquanto o Código 19 contém todas as operações necessárias para se chegar a este resultado.



(a) Contornos obtidos com um *threshold*.
(b) Contornos obtidos com dois *thresholds*.

Figura 4.5: Contornos desenhados em seus respectivos elementos. Nos contornos da Figura 4.5(a), os dados foram obtidos com apenas um *threshold*, já na Figura 4.5(b) foram usados dois *thresholds*.

Código 19: Toda a lógica para obtenção dos contornos e desenho dos mesmos.

```

1 img = cv2.imread('DIR_PROJETO/images/logo_iff.png')
2 img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
3 ret, thresh_ball = cv2.threshold(img_gray, 93, 255, cv2.THRESH_BINARY)
4 contours, hierarchy = cv2.findContours(thresh_ball, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
5 cv2.drawContours(img, contours[1], -1, (255,0,0),3)
6 ret, thresh_squares = cv2.threshold(img_gray, 155, 255, cv2.THRESH_BINARY)
7 contours2, hierarchy = cv2.findContours(thresh_squares, cv2.RETR_TREE, \
8         cv2.CHAIN_APPROX_NONE)
9 cv2.drawContours(img, contours2, -1, (255,0,0),3)
10 cv2.imwrite('DIR_PROJETO/images/saida.png', img)

```

Ainda, através dos contornos, é possível recuperar os momentos da imagem. A biblioteca realiza essa tarefa através da função *moments()*. O Código 20 recupera os momentos do segundo contorno da lista, que contém o círculo vermelho do logotipo. Esse método retorna um dicionário, que é um tipo de dados da linguagem Python. Ao exibir as informações contidas nessa lista, é possível visualizar todos os pares de chave e valor do dicionário (Código 21).

Código 20: Os momentos são recuperados ao passar um contorno que representa um objeto da imagem.

```

1 moments = cv2.moments(contours[1])

```

Código 21: Os valores dos momentos do segundo elemento do logotipo do IFF são mantidos por um dicionário, um tipo de dados do Python.

```

1 {'mu2': 4466250.579995997, 'mu3': 2.099651949538806e-06, 'm11': 24999544.25,
2 'nu2': 0.07959091389346826, 'm12': 1642430867.25, 'mu21': -6560.77209803462,
3 'mu20': 4465517.828794554, 'nu20': 0.07957785588502587, 'm30': 2505168506.5,
4 'nu21': -1.350844812113205e-06, 'mu11': -4485.94523428008,
5 'mu12': -19483.180956840515, 'nu11': -7.994188290096345e-05,
6 'nu12': -4.011533021684291e-06, 'm02': 26989947.0, 'm03': 1969781748.75,
7 'm00': 7491.0, 'm01': 410761.5, 'mu30': 23255.18666744232,
8 'nu30': 4.788178555058973e-06, 'm10': 455995.0, 'm20': 32223018.833333332,
9 'm21': 1766364339.4166667}

```

Essas chaves são de três tipos: *mu* (momentos centrais), *nu* (momentos centrais normalizados) e *m* (momentos espaciais). Ainda existem os momentos invariantes, também conhecidos como momentos *Hu*, que podem ser recuperados através do método *HuMoments()*. Mais sobre momentos no OpenCV podem ser encontrados no trabalho realizado por Kilian (2001).

Através desses momentos, é possível obter características dos objetos, como sua área e centróide. A área do objeto é contida na chave *m00*. Tendo esse valor, encontra-se então as coordenadas x e y do centro de massa do objeto (Código 22).

Código 22: Operações com os momentos do segundo elemento da lista de contornos.

```
1 area = moments['m00']
2 x = moments['m10'] / area
3 y = moments['m01'] / area
```

Caso esteja definido a forma do objeto a ser encontrado, como um círculo, se torna interessante a recuperação de outros dados específicos dessa forma. Com isso, o resultado final tende a ser mais preciso, devido ao aumento de informações recuperadas do objeto a ser detectado. Como no exemplo apresentado é utilizado um objeto em forma circular, uma das informações relevantes a ser recuperada a respeito desse objeto é o seu raio. Isso é feito usando a fórmula da geometria, junto ao valor da área do contorno. Então, desenhando os valores encontrados, determina-se exatamente o centro e contorno do círculo vermelho em relação a figura em que ele está inserido (Código 23). O resultado é exposto na Figura 4.6.

Código 23: Desenho feito na imagem de exemplo usando os parâmetros calculados a partir do momentos do contorno.

```
1 raio = sqrt(area/3.14)
2 cv2.circle(img, (int(x),int(y)), int(raio), (255,0,0), 3)
3 cv2.circle(img, (int(x),int(y)), 3, (255,0,0), -1)
```

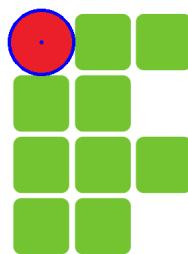


Figura 4.6: Rastreamento feito no círculo vermelho do logotipo do IFF.

A biblioteca ainda conta com recursos para manipulações com vídeo, onde pode-se utilizar arquivos de vídeo ou câmeras, como a de um notebook. Para o caso de uma câmera de um notebook usa-se o comando mostrado no Código 24.

Código 24: Inicialização do vídeo através da biblioteca OpenCV.

```
1 video = cv2.VideoCapture(0)
```

Um número inteiro é passado pela função que recupera o vídeo, representando o identificador do dispositivo de vídeo. A partir desse objeto, cria-se então uma rotina para captura e apresentação dos *frames* do vídeo em uma nova janela (Código 25).

Código 25: Lógica para processamento e visualização dos *frames* do vídeo.

```
1 while video.grab():
2     flag, frame = video.retrieve()
3     if flag:
4         cv2.imshow('Video', frame)
5         cv2.waitKey(1)
```

Tendo em mente que um vídeo é composto por diversas imagens em sequência, os mesmos tratamentos feitos em imagens também podem ser utilizados em vídeos, como aplicação de filtros e desenhos. Isso abrange as possibilidades em inúmeras atividades, que podem evoluir até sistemas completos de Visão Computacional.

4.2 SimpleCV

SimpleCV⁶ é um *framework*⁷ de código aberto que foi desenvolvido pelos engenheiros da Sight Machine, e está sob a licença BSD (*Berkeley Software Distribution*). O *framework* é portável nas plataformas Mac, Windows e distribuições Linux Ubuntu e Arch Linux.

O objetivo do *framework* é tornar mais fácil aos programadores o desenvolvimento de sistemas de Visão Computacional, simplificando muitas das tarefas mais comuns. Demaagd et al. (2012) destacam que, por ser uma biblioteca simples, não é necessário possuir muitos conhecimentos em Python e especialização na área da computação, sendo preciso apenas o interesse em Visão Computacional.

4.2.1 Estrutura

Em sua implementação, o SimpleCV utiliza bibliotecas existentes para Python na área de Visão Computacional, reunindo os recursos mais importantes de cada uma delas. A Figura

⁶<http://simplecv.org>

⁷Coleção abstrata de classes, interfaces e padrões dedicados a resolver um conjunto de problemas através de uma arquitetura flexível e extensível.

4.7 expõe a relação do SimpleCV com as bibliotecas existentes no mercado. É destacado pelo diagrama a nova camada criada pelo SimpleCV, que acessa os recursos necessários das bibliotecas para Python. Dessa forma, grande parte das tarefas de Visão Computacional comumente realizadas, tornam-se mais simples.

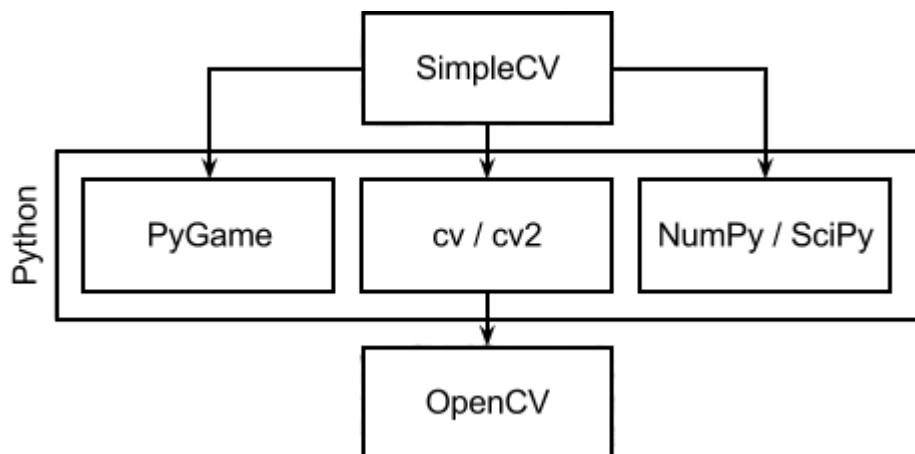


Figura 4.7: Bibliotecas usadas pelo SimpleCV divididas em camadas.

Segundo sua documentação oficial⁸, o SimpleCV se encontra em sua versão 1.3 dividido em conjuntos de classes. Essas classes são separadas da seguinte forma: *Features*, *MachineLearning*, *Segmentation* e *Shell*. O pacote *Features*, conta com métodos para classificação de objetos em imagens, baseando-se em suas características. O conjunto de classes contidos no pacote *MachineLearning* implementam alguns algoritmos de classificação, como o *k-nearest neighbor* e *Naive Bayes*. O pacote *Segmentation* contém classes para segmentação de imagens, podendo ser diferenciada por cor ou comparando diferenças entre dois *frames*. A classe *Shell* permite que a biblioteca seja usada via terminal, tendo dessa forma, resposta rápida para dúvidas que surgirem durante o desenvolvimento, realização de testes rápidos e acesso a última documentação dos objetos do SimpleCV. Essa implementação é baseada no IPython.

Além desses pacotes, existem ainda as classes básicas da ferramenta, sendo elas: *Camera*, *Color*, *ColorModel*, *Display*, *DrawingLayer*, *EXIF*, *Font*, *ImageClass* e *Stream*. Cada uma delas possui uma finalidade distinta. Com a classe *Camera*, por exemplo, é possível trabalhar com vídeo de diversos dispositivos, como a de um notebook, de câmeras que enviam sinais pela Internet (*stream*) ou até mesmo da câmera do Kinect⁹, dispositivo desenvolvido pela Microsoft. Outras classes fornecem manipulações com interface, mostrando resultados de imagens e vídeo em janelas do sistema operacional e eventos com o mouse, e desenhos básicos de formas e texto.

⁸<http://simplecv.org/docs>

⁹Sensor de movimentos de última geração, desenvolvimento inicialmente para jogos eletrônicos.

4.2.2 Instalação

Existem diversas formas de realizar a instalação do SimpleCV no Linux Ubuntu 11.10. Independente do método escolhido para a instalação do *framework*, existem dependências que precisam ser instaladas previamente. São elas: Python (2.7 ou superior), Python Setup Tools, PyGame, NumPy, SciPy, Easy Install e o OpenCV. Essas dependências podem ser instaladas através do comando mostrado no Código 26.

Código 26: Instalação das dependências do SimpleCV.

```
sudo apt-get install python-setuptools python-pygame python-scipy
```

Por padrão as dependências restantes já vem instaladas na versão 11.10 do Linux Ubuntu, sendo um caso isolado o OpenCV, que possui um processo de instalação, como mostrado na Seção 4.1.2. Para a instalação do SimpleCV em si, é necessário realizar o download do pacote .deb em seu site oficial¹⁰. Caso esteja tudo devidamente configurado, basta executar o .deb e a instalação será feita. No entanto, nos testes realizados, essa instalação apresentou problemas ao executar um *script* simples de uso da ferramenta. Como solução, foi necessário baixar a última versão da biblioteca do repositório de controle de versão do projeto, o GitHub. Realizada esta operação, bastou entrar na pasta que foi criada e executar o arquivo de instalação, utilizando o próprio interpretador do Python. Essas operações são mostradas no Código 27, que inclui a instalação do Git, necessário para realizar o download do código fonte do repositório de controle de versão.

Código 27: Instalação da biblioteca SimpleCV através do GitHub.

```
sudo apt-get install git
git clone git://github.com/ingenuitas/SimpleCV.git SimpleCV
sudo apt-get install python-setuptools
sudo python setup.py install
```

Além de possuir bibliotecas essenciais para o seu funcionamento, o SimpleCV também conta com algumas bibliotecas opcionais para uso exclusivo em determinados projetos, como PIP, BeautifulSoup, webm, freenect, entre outros.

Também é possível fazer com que o SimpleCV seja utilizado em interatividade com o shell. Para isso, é necessário a instalação do IPython, uma ferramenta da linguagem Python para

¹⁰<http://simplecv.org/download>

shell. O download dessa ferramenta pode ser feito pelo seu site¹¹, ou ser baixada e instalada automaticamente através do terminal (Código 28).

Código 28: Instalação do IPython.

```
sudo apt-get install ipython
```

4.2.3 Uso prático

Para modo comparativo, serão mostradas nesta Seção funções equivalentes às utilizadas na Seção sobre OpenCV.

Diferente da interface *cv2* do OpenCV, o *framework* SimpleCV conta com classes próprias para manipulação de imagens, assemelhando-se a interface *cv2.cv*. Ao ler sua documentação, percebe-se que a proposta da ferramenta é facilitar o trabalho do programador, evitando que manipulações manuais sejam feitas. Neste caso, por possuir a maioria das operações necessárias, o uso do *framework* evita trabalhos repetitivos normalmente realizados no desenvolvimento de sistemas de Visão Computacional. Por outro lado, essa filosofia de uso da ferramenta torna o aprendizado um pouco lento, sendo necessário constantes consultas a descrições de métodos e estruturas de dados em sua documentação.

A seguir, são mostrados exemplos de operações que podem ser feitas com a ferramenta. Como exemplo inicial, é exposto no Código 29 o carregamento de uma imagem utilizando o *framework* SimpleCV.

Código 29: Carregamento de uma imagem utilizando o *framework* SimpleCV.

```
1 from SimpleCV import Image
2 img = Image('DIR_PROJETO/images/logo_iff.png')
```

A imagem carregada é criada como uma classe do próprio *framework*, *SimpleCV.Image*. Nela, estão implementadas diversas funções para ações que podem ser feitas com a imagem, como mostrá-la na tela ou acessar seus *pixels*. Para acessar um pixel da imagem, por exemplo, utiliza-se o método *getPixel()*. Caso opte-se por manipular manualmente seus *pixels*, é possível recuperar o mesmo tipo de objeto retornado por padrão pela interface *cv2* do OpenCV, através do método *getNumpy()*.

¹¹<http://ipython.org>

Seguindo a mesma ideia, outras manipulações podem ser realizadas, como transformar uma imagem em binária, alterar seu espaço de cor ou aplicar filtros (Código 30).

Código 30: Conversões de espaço de cor utilizando a ferramenta SimpleCV.

```

1 # Binarizacao em preto e branco
2 img.binarize()
3 # Binarizacao em branco e preto
4 img.binarize().invert()
5
6 # Conversao para RGB
7 img.toBGR()
8 # Conversao para HSV
9 img.toHSV()
10 # Conversao para escala de cinza
11 img.toGray()
12
13 # Aplicacao do filtro dilatar
14 img.dilate()
15 # Aplicacao do filtro corroer
16 img.erode()

```

O método *binarize()* aplica um limiar na imagem, tornando os valores acima desse limiar em preto, e os valores abaixo em branco. Ao utilizar o método *invert()*, esses valores são invertidos. O método *threshold()* também pode ser utilizado para se ter o mesmo resultado, contudo, por padrão são colocados os *pixels* abaixo desse limiar em preto, e os acima em branco. Pode-se também passar valores como parâmetros para determinar este limiar.

Quanto as transformações nos espaços de cores da imagem usados no exemplo acima, os métodos utilizados são bastante intuitivos. O método *toBGR()* transforma os valores dos *pixels* da imagem seguindo a ordem azul, verde e vermelho, mesmo padrão utilizado pelo OpenCV. Já o método *toHSV()* realiza essa transformação para os componentes matiz, saturação e valor. Ainda, o método *toGray()* transforma a imagem para escala de cinza.

Por fim, os filtros utilizados no exemplo são filtros morfológicos que alteram a forma dos objetos na imagem. Através do método *dilate()*, os limites dos objetos com mais brilho são expandidos, e os que são mais escuros reduzidos. No método *erode()* ocorre o inverso, os limites dos objetos com mais brilho são reduzidos, e os mais escuros são expandidos.

Com os recursos disponibilizados pelo *framework*, também é possível a recuperação de *blobs*¹² através de máscaras, que são imagens binárias possuindo apenas a região de interesse a ser detectada em branco, e o resto em preto (RAHMAN, 2010). A máscara passada como no exemplo exposto no Código 31, deve possuir as mesmas dimensões da imagem original,

¹²Ponto ou uma região na imagem que difere em propriedades como brilho ou cor se comparado ao ambiente a sua volta.

além de ser necessário ter sido passada previamente por uma etapa de processamento, como utilização de métodos como *binarize()* ou *toGray()*. Esses *blobs*, então são desenhados usando o método *draw()*, tendo ainda a possibilidade de passagem de parâmetros para determinar a cor e o tamanho da linha a ser desenhada.

Código 31: Recuperação dos *blobs* são feitos através da imagem, enquanto o responsável por desenhá-los é objeto *blob*.

```
1 from SimpleCV import Color
2 blobs = img.findBlobsFromMask(img.binarize(155))
3 blobs.draw(Color.BLUE, 3)
```

Por sua vez, os momentos podem ser acessados após a recuperação dos *blobs* da imagem. Ao utilizar o método *findBlobsFromMask()*, é retornado um *FeatureSet* contendo todos os *blobs* da imagem, com base na máscara passada. Ao executar o método *show()*, esses *blobs* são contornados. Ainda, pode-se acessar cada momento em particular, simplesmente chamando a variável de um *blob* no vetor em que está contido. Essas operações podem ser melhor entendidas através do Código 32.

Código 32: Operações com momentos dos *blobs* com o *framework* SimpleCV.

```
1 # Visualizando todos os blobs
2 blobs.show()
3
4 # Visualizando o blob na posicao 8 do vetor
5 blobs[8].show()
6
7 # Acessando os momentos do blob na posicao 0 do vetor
8 blobs[0].m00
9 blobs[0].m10
10 blobs[0].m01
```

Assim como o OpenCV, o SimpleCV suporta diversos dispositivos de câmera diferentes. Para habilitar essa funcionalidade no *framework*, são utilizados os comandos no Código 33. As linhas do exemplo mostram como inicializar uma câmera local, podendo ser utilizada uma embutida em um notebook ou uma instalada no próprio computador pessoal, através, por exemplo, de uma conexão USB.

Código 33: Lógica para inicialização de vídeo com o *framework* SimpleCV.

```

1 from SimpleCV import Camera
2 cam = Camera()
3 while True:
4     cam.getImage().show()

```

4.3 Conclusão

Apesar de haver diversas bibliotecas para auxiliar o desenvolvimento de sistemas de Visão Computacional, existe apenas um conjunto limitado com suporte para linguagem Python. Nas pesquisas realizadas, a biblioteca OpenCV se mostrou predominante em se tratando de ferramentas para uso geral. Assuntos relacionados a ferramenta são facilmente encontrados, facilitando a aproximação dos iniciantes da área de Visão Computacional com a biblioteca. Sua documentação também é bastante detalhada, onde são mostrados exemplos de uso da maioria das funções disponíveis.

Em se tratando do suporte para linguagem Python, as diferenças entre as interfaces do OpenCV disponíveis não são claramente apresentadas. No trabalho realizado, o impacto dessa falta de clareza atrasou o desenvolvimento do estudo de caso. Os estudos foram iniciados utilizando a interface *cv* e, percebendo a existência e vantagens da interface *cv2*, mudou-se o foco nas pesquisas.

Complementando os estudos, o *framework* SimpleCV mostrou-se promissor. A ferramenta reúne diversos recursos comumente utilizados, minimizando o esforço repetitivo no desenvolvimento de sistemas de Visão Computacional. Percebe-se que o *framework* não foi criado para substituir o OpenCV, pelo contrário, é uma ferramenta complementar que pode ser utilizada em conjunto com o OpenCV, caso esse seja o desejo do desenvolvedor.

Oliver (2012) descreve um comparativo entre a biblioteca OpenCV e o *framework* SimpleCV levando em conta diversos aspectos: facilidade no uso, velocidade, recursos necessários para execução, custo, ambiente de desenvolvimento, gerenciamento de memória, portabilidade, entre outros. Nos quesitos facilidade no uso e gerenciamento de memória, o SimpleCV se mostra bastante superior em comparação com o OpenCV. Por outro lado, nos quesitos velocidade e portabilidade, o OpenCV se torna uma melhor opção.

Na perspectiva de iniciantes da área de Visão Computacional, que desconhecem quaisquer ferramentas disponíveis para uso no desenvolvimento de aplicações, o acervo de materiais encontrados na Internet são em sua grande maioria voltado para o OpenCV. Conclui-se, então, que apesar do *framework* SimpleCV facilitar o processo de desenvolvimento em diversos aspectos, o OpenCV proporciona uma curva de aprendizado melhor, por trazer na literatura assuntos

práticos relacionados ao desenvolvimento de sistemas de Visão Computacional.

Com base nisso, utilizou-se a biblioteca OpenCV em conjunto com a interface *cv2* para Python para desenvolvimento da aplicação apresentada como estudo de caso deste trabalho.

5 ESTUDO DE CASO

Nesse capítulo será apresentada a implementação de uma aplicação capaz de detectar objetos circulares em vídeo, de forma a exemplificar os conceitos de Visão Computacional abordados neste trabalho, utilizando a linguagem Python em conjunto com a biblioteca do OpenCV.

Por fim, são mostrados resultados da aplicação desenvolvida sendo executada em diferentes situações. Foram realizados testes em locais que apresentam tipos de luminosidade distintos, além de forçar um deslocamento do objeto no vídeo, com a finalidade de encontrar possíveis erros de detecção.

5.1 Objetivo

O estudo de caso tem como objetivo implementar um sistema utilizando técnicas de Visão Computacional, com o auxílio de uma das ferramentas estudadas, a fim de proporcionar um melhor entendimento da área. Através de algoritmos de simples entendimento, a aplicação busca realizar a detecção e rastreamento de objetos circulares monocromáticos em um vídeo projetado através de uma câmera conectada ao computador.

5.2 Tecnologias utilizadas

Como tecnologias para a elaboração do estudo de caso, foi escolhido Python como linguagem de programação. Além disso, como ambiente de desenvolvimento optou-se pela distribuição Linux Ubuntu. Linux foi entendida como uma opção mais acessível por ser livremente distribuída e possuir a linguagem Python instalada na maioria de suas distribuições.

5.2.1 Python

Criada em 1991 pelo programador de computadores Guido van Rossum, Python é uma linguagem de programação de alto nível, interpretada e orientada a objeto. Além dessas características, a tecnologia conta com tipagem dinâmica, onde não é preciso declarar explicitamente o tipo de cada variável. Mantida pela Python Software Foundation, organização sem fins lu-

crativos, o uso dessa ferramenta vem se tornando cada vez mais popular nos últimos anos, e se encontra em constante desenvolvimento, estando atualmente na versão 3.

5.2.2 Linux

São reconhecidos como sistemas operacionais Linux aqueles que são criados utilizando o *kernel*¹ criado por Linus Torvalds. Outra característica desse sistema operacional é o seu código aberto (*open source*). Dessa forma, desenvolvedores podem colaborar para o aprimoramento do sistema, levando em conta o feedback dos usuários.

Por se tratar de um software colaborativo e personalizável, com o passar do tempo desenvolveu-se diversos sistemas para complementar o uso do sistema operacional Linux. Assim, diversas versões do sistema operacional foram criadas, sendo denominadas distribuições Linux.

Dentre as principais distribuições Linux estão: Debian, Slackware e Red Hat. A partir delas, novas distribuições tem sido criadas e vem sendo utilizadas para finalidades distintas. Uma que vem se destacando é a distribuição Linux Ubuntu, desenvolvida pela Canonical. Sua interface amigável tem atraído diversos usuários ao redor do mundo e tem popularizado o Linux em geral. A distribuição conta com facilidades na instalação, realizada por um CD ou até mesmo através de um pendrive. Ainda, aplicações são facilmente instaladas utilizando o gerenciador de pacotes Apt.

5.3 Implementação

A etapa de implementação envolveu utilizar o conhecimento adquirido para criação de uma aplicação capaz de rastrear objetos circulares em vídeo. Para alcançar esse objetivo, a primeira etapa do processo de desenvolvimento da aplicação foi destinada a captura de uma cena através de uma câmera e reprodução em vídeo. O equipamento utilizado foi uma câmera integrada de um notebook Dell Vostro 1320, que também possui um processador Intel Core 2 Duo T6670 e 4 GB de memória RAM DDR2.

Como exemplo, é apresentada na Figura 5.1 a inicialização simples de uma cena em vídeo utilizando o OpenCV.

¹Programa que constitui o núcleo principal do sistema operacional.



Figura 5.1: Vídeo sendo reproduzido utilizando a biblioteca OpenCV.

Outra necessidade importante da aplicação desenvolvida refere-se a interação com o usuário, de modo a permitir que o mesmo seja capaz de selecionar no vídeo reproduzido pela câmera o objeto desejado para detecção e rastreamento. Para possibilitar esse recurso, desenvolveu-se uma funcionalidade onde, ao clicar com um dos botões do mouse no vídeo, a aplicação seja capaz de distinguir o objeto a ser detectado e rastreado.

A aplicação desenvolvida é composta por quatro classes, relacionadas de forma a separar a responsabilidade de cada uma delas, a fim de facilitar o entendimento do código. Esse relacionamento pode ser visto na Figura 5.2.

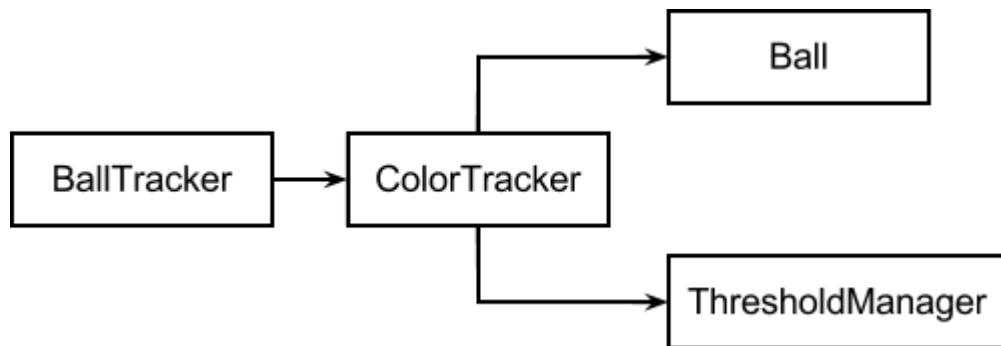


Figura 5.2: Relacionamento entre as classes do sistema de rastreamento.

O Código 34 mostra a inicialização do sistema. Na primeira linha de sua execução, é criado um dicionário com algumas informações pertinentes da janela, sendo elas: nome, largura e altura. Em seguida, um método do OpenCV é usado para nomear a janela a ser criada, utilizando a chave do dicionário que contém o nome desta janela. É instanciado então um vídeo, através do método do OpenCV, passando como parâmetro seu identificador.

Outras instruções no código são relacionadas a lógica para a realização do rastreamento. A criação do objeto *ball_tracker* (linha 5) tem como responsabilidade a descoberta das coordenadas do círculo, utilizando os valores do objeto a ser detectado e rastreado obtidos pelo clique do mouse. Realizada essa lógica, o objetivo final do objeto *ball_tracker* é retornar uma instância da classe *Ball* (linha 10), que será utilizada pelo método *circle()* do OpenCV para destacar o objeto no vídeo (linhas 12 e 13). Na linha 6, é utilizado outro método da biblioteca, que gerencia

os eventos do mouse na tela. Uma repetição, então, é criada com o objeto do vídeo, de forma a obter seus *frames* (linha 7).

Essa implementação permite que, dentro dessa repetição, cada *frame* do vídeo seja capturado e manipulado. O *frame* então é recuperado (linha 8), e, caso seja válido, é usado por outros métodos. No último bloco, é feita a recuperação do objeto *Ball*, contendo as coordenadas necessárias para que um círculo seja desenhado na tela. Esse objeto, então, é usado pelo método do OpenCV que circula a esfera encontrada com uma linha a sua volta e um ponto no meio, todos em azul. Por fim, o *frame* desenhado é inserido na janela criada, de forma a ser visto a partir da interface com o usuário (linha 14).

Código 34: Chamada principal do aplicativo.

```

1 if __name__ == '__main__':
2     WINDOW_PROPERTIES = {'name': 'Visualisation', 'width': 640, 'height': 480}
3     cv2.namedWindow(WINDOW_PROPERTIES['name'])
4     video = cv2.VideoCapture(0)
5     ball_tracker = BallTracker(video)
6     cv2.setMouseCallback(WINDOW_PROPERTIES['name'], on_mouse, [ball_tracker])
7     while video.grab():
8         flag, frame = video.retrieve()
9         if flag:
10             ball = ball_tracker.get_ball(frame)
11             if ball is not None:
12                 cv2.circle(frame, (ball.x, ball.y), ball.radius, \
13                             (255, 0, 0), 3)
14                 cv2.circle(frame, (ball.x, ball.y), 3, (255, 0, 0), -1)
15             cv2.imshow(WINDOW_PROPERTIES['name'], frame)
16             cv2.waitKey(1)

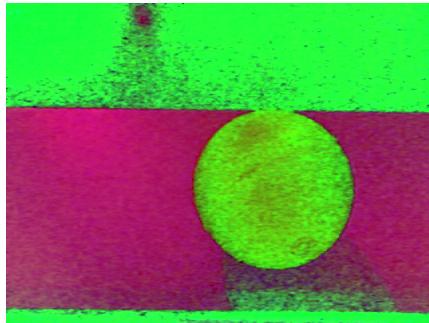
```

A classe *BallTracker*, utilizada na chamada principal, está contida no arquivo *tracker.py*, que também possui a classe *ColorTracker*. A primeira, utiliza a segunda para a recuperação do objeto no vídeo baseando-se nos valores HSV do *frame*.

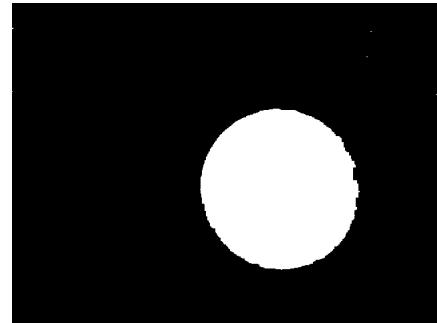
Como ponto de partida, tem-se a chamada do método *get_ball()*, da classe *BallTracker*, realizada na repetição criada pelo objeto do vídeo (Código 35). Através desse método, é passado para a classe *ColorTracker* o *frame* atual do vídeo, utilizando a função *find_ball()*. O espaço de cores desse *frame* é convertido para valores HSV (Figura 5.3(a)) e é devidamente binarizado (Figura 5.3(b)). Como pode ser visto no Código 36, que contém a lógica do método *find_ball()*, são aplicados alguns filtros para melhor detecção dos contornos (linha 5). Após realizados esses tratamentos no *frame*, os contornos são obtidos e utilizados para a recuperação dos parâmetros que compõem o círculo, sendo esses o centróide e raio (linha 6).

Código 35: Chamada do método da classe *BallTracker* que recupera os valores do círculo baseando-se pelo *frame* atual.

```
1 def get_ball(self, frame):
2     return self.color_tracker.find_ball(frame)
```



(a) *Frame* do vídeo em espaço de cor HSV.



(b) *Frame* após aplicação da técnica de binarização.

Figura 5.3: Tratamento nos *frames* do vídeo.

Código 36: Lógica do método *find_ball()* da classe *ColorTracker*.

```
1 def find_ball(self, frame):
2     if self.manager is not None:
3         hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
4         self.thresholded_frame = self.manager.threshold_frame(hsv_frame)
5         self.thresholded_frame = self.apply_filters(self.thresholded_frame)
6         return self.get_ball(self.thresholded_frame)
7
8 def get_ball(self, thresholded):
9     contours, hierarchy = cv2.findContours(thresholded, cv2.RETR_TREE, \
10                                             cv2.CHAIN_APPROX_SIMPLE)
11     x, y, radius = self.get_ball_parameters(contours)
12     ball = Ball(x, y, radius)
13     return ball
```

Como pode ser visto no Código 37, para a recuperação dos parâmetros do centróide e raio do círculo, é chamada a função que recupera os momentos do contorno da imagem binarizada (linha 6). Recuperado o momento do contorno, uma condição é feita para que sejam buscados os parâmetros apenas dos círculos com área maior que 1000 (linha 8). Dessa forma, evita-se que áreas do vídeo com valores HSV parecidos com o objeto desejado para detecção sejam desenhados, alcançando assim resultados mais próximos do esperado pelo usuário ao clicar com o mouse no vídeo.

Código 37: Recuperação dos valores pertinentes para identificar um círculo no *frame* de vídeo.

```

1 def get_ball_parameters(self, contours):
2     (... )
3     SMALLEST_OBJECT = 1000
4     for contour in contours:
5         try:
6             moments = cv2.moments(contour)
7             area = moments['m00']
8             if area > SMALLEST_OBJECT:
9                 x = moments['m10']/area
10                y = moments['m01']/area
11                radius = sqrt(area/3.14)
12            except:
13                pass
14        return x, y, radius

```

Para gerenciar o processo de binarização dos *frames*, implementou-se a classe *ThresholdManager*, tornando possível tratar os parâmetros do círculo e aplicá-los quando necessário. Essa classe possui em sua implementação diversas condições de mudança dos valores HSV, para que se ajustem não só a nova condição informada pelo mouse, mas também levando em conta o ambiente.

Quanto aos eventos do mouse, o OpenCV os trata através do método *setMouseCallback()*, como visto anteriormente no Código 34, responsável por declarar um método que é chamado a cada vez que um evento do mouse é disparado. Este método deve possuir a assinatura especificada pela documentação do OpenCV, contendo o evento a ser disparado, as coordenadas x e y do mouse no vídeo, e um vetor para recuperação dos parâmetros passados pelo método *setMouseCallback()*. Como pode ser visto no Código 38, esse valores são usados para que, ao clicar com o botão esquerdo do mouse no vídeo, os parâmetros x e y sejam passados para o método *set_hsv_values()*. O Código 39 mostra a lógica desse método, que converte o *frame* do vídeo atual para o espaço de cor HSV, e através do *frame* convertido, recupera o valor HSV do *pixel* referente as coordenadas x e y passadas como parâmetro para o método.

Código 38: Método que recupera informações do clique esquerdo do mouse.

```

1 def on_mouse(event, x, y, flag, param):
2     ball_tracker = param[0]
3     if event == cv2.EVENT_FLAG_LBUTTON:
4         ball_tracker.set_hsv_values(x, y)

```

Código 39: Converte o *frame* para espaço HSV.

```

1 def set_hsv_values(self, x, y):
2     flag, frame = self.video.retrieve()
3     if flag:
4         hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
5         self.color_tracker.new_value(hsv_frame[y][x][0], hsv_frame[y][x][1], \
6             hsv_frame[y][x][2])

```

Dessa forma, os novos valores HSV são passados para o método *new_value()* da classe *ColorTracker* (Código 40). Em seguida, o método *new_value()* invoca a classe *ThresholdManager* para que gerencie os novos valores do *pixel*, e inclua-os na binarização dos próximos *frames* do vídeo.

Código 40: Um objeto do tipo *ThresholdManager* é instanciado para que trate os valores HSV recuperados através do *pixel* do *frame*.

```

1 def new_value(self, h, s, v):
2     if self.manager is not None:
3         self.manager.add_values(h, s, v)
4     else:
5         self.manager = ThresholdManager(h, s, v)

```

A chamada da classe *ThresholdManager* é feita em dois momentos da execução do aplicativo. Na primeira vez em que a ação do clique com o mouse é realizada pelo usuário, o objeto é instanciado passando os valores HSV. Esses valores são então inicializados para criação de dois limiares, contendo valores HSV mínimos e máximos, que serão utilizados para a binarização dos *frames* (Código 41). Para os próximos eventos disparados pelo mouse, ao invés de instanciar novamente um novo objeto do tipo *ThresholdManager*, o método *add_values()* é invocado pelo objeto anteriormente criado (Código 42).

Código 41: Inicialização da classe *ThresholdManager*.

```

1 def __init__(self, h, s, v):
2     self.min_h = max(0, h-1)
3     self.max_h = min(179, h+1)
4     self.min_s = max(0, s-1)
5     self.max_s = min(255, s+1)
6     self.min_v = 0
7     self.max_v = 220

```

Código 42: Método responsável por ajustar os valores HSV do limiares usadas para binarizar o *frame* do vídeo.

```

1 def add_values(self, h, s, v):
2     HUE_MARGIN = 40
3     if h < (self.min_h - HUE_MARGIN):
4         self.min_h -= 180
5         self.max_h -= 180
6     elif h > (self.max_h + HUE_MARGIN):
7         h -= 180
8
9     if h < self.min_h:
10        self.min_h = h-1
11    if h > self.max_h:
12        self.max_h = min(179, h+1)
13    if s < self.min_s:
14        self.min_s = max(0, s-1)
15    if s > self.max_s:
16        self.max_s = min(255, s+1)

```

Através dessa estrutura lógica, ao clicar com o mouse no objeto desejado para detecção, o mesmo é contornado por um círculo azul e marcado com um ponto, indicando seu centróide. Devido as variações das cores do objeto a ser detectado, causadas pela luminosidade do local, procura-se escolher áreas do objeto contendo cores mais homogêneas. Dessa forma, resultados mais precisos são alcançados, como visto na Figura 5.4.



Figura 5.4: Resultado final do sistema, detectando uma bola laranja na tela e circulando-a.

5.4 Resultados

Como modelo de objeto para detecção e rastreamento, utilizou-se uma bola laranja. Esse objeto foi utilizado em dois planos de fundo, um monocromático e outro policromático. Nesses dois casos, situações foram testadas com o objeto parado e em movimento na cena.

Em um plano de fundo básico, os ajustes dos limiares puderam ser feitos baseando-se em diferentes contrastes de cores referentes ao objeto. Dessa forma, a detecção se mostrou precisa,

como mostra a Figura 5.5, estando o objeto completamente parado. Expondo o mesmo objeto em diferentes velocidades de movimentação no vídeo, os resultados mostraram-se satisfatórios (Figura 5.6). Em casos onde o objeto se desloca lentamente, o círculo que denota a detecção não possuiu a mesma precisão, porém apresentou um raio de dimensão próxima a original (Figura 5.6(a)) (Figura 5.6(b)). Nos casos onde o objeto se desloca mais rapidamente, a tendência é que a detecção seja feita acompanhando a trajetória do objeto, porém com precisão menor, fazendo com que o círculo representativo da detecção diminua (Figura 5.6(c)).



Figura 5.5: Caso básico de detecção, com o objeto parado no vídeo.

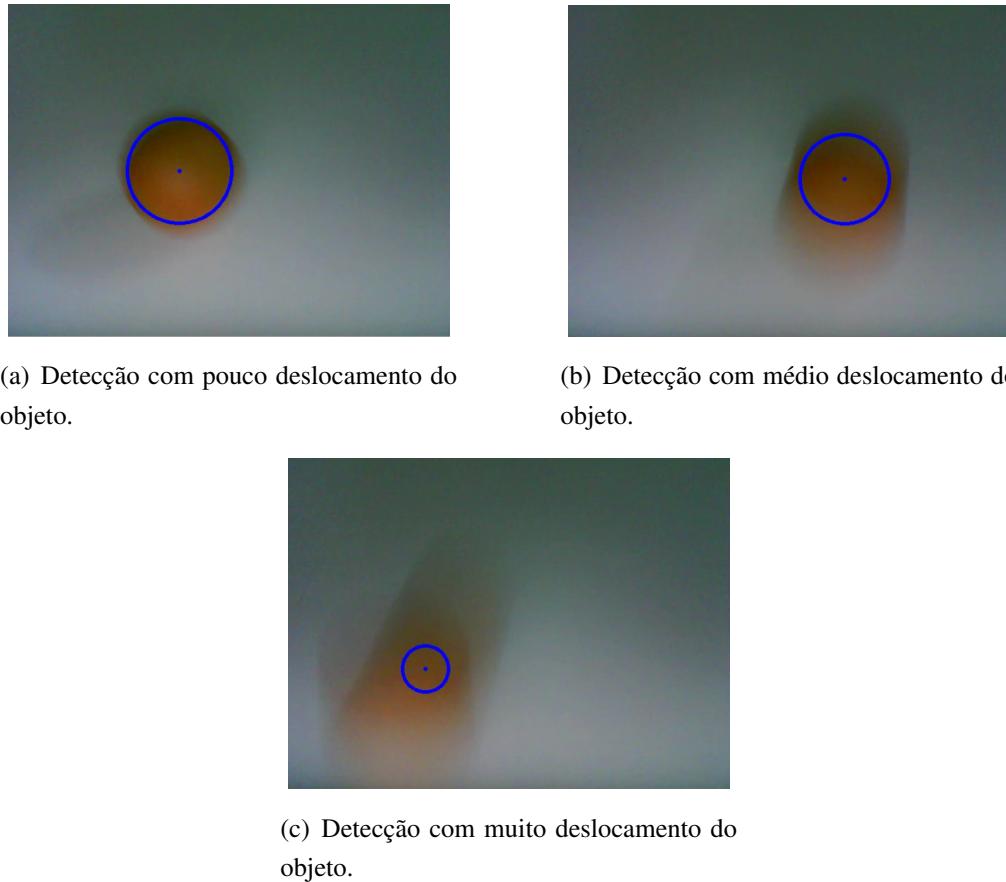


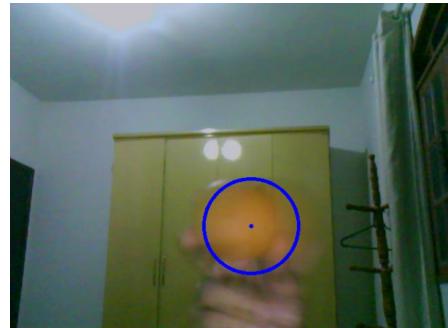
Figura 5.6: Diversos testes feitos com o objeto em diferentes velocidades.

Nos casos em que foi utilizado um plano de fundo policromático, a detecção se manteve

precisa. Ajustando os limiares com base no novo plano de fundo, o objeto foi rastreado, porém com intermitências em sua detecção. A Figura 5.7 apresenta casos em que o objeto foi exposto em diferentes distâncias em relação ao vídeo, estando em movimento. Apesar do objeto se apresentar em contraste parecido com o fundo, a aplicação mostrou qualidade na precisão, separando o objeto do plano de fundo complexo.



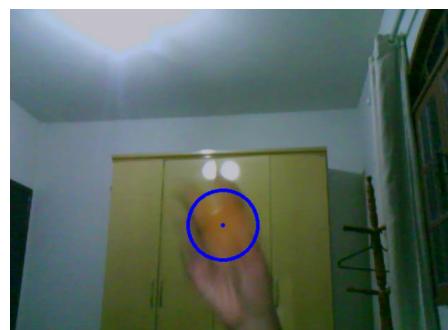
(a) Detecção com o objeto parado em um plano de fundo complexo.



(b) Detecção com o objeto em movimento próximo ao vídeo.



(c) Detecção com o objeto em movimento afastado do vídeo.



(d) Detecção com o objeto em distância média do vídeo.

Figura 5.7: Diversos testes feitos com o objeto em um plano de fundo complexo, em diferentes proximidades com o vídeo.

No entanto, em alguns casos com o objeto sendo deslocado rapidamente, a detecção não foi feita, assim como nos casos testados com plano de fundo básico (Figura 5.8). Apesar de não terem sido realizados testes mais aprofundados, a utilização de equipamentos mais sofisticados para a captura do vídeo e processamento da aplicação podem ocasionar em resultados mais satisfatórios na detecção do objeto.

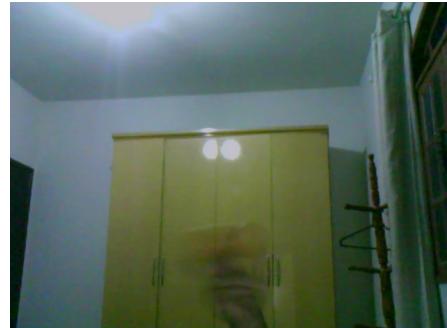


Figura 5.8: Em alguns momentos, a aplicação falhou em detectar o objeto desejado.

Outros casos ainda foram testados considerando outros fatores como iluminação e o objeto não sendo mostrado por completo no vídeo. Os casos em que o objeto é colocado próximo a luz, a aplicação passou a não detectá-lo (Figura 5.9). Por terem sido utilizados limiares com base nos tons laranja do objeto, ao aproximar o objeto da luz, o tom de cor do mesmo é alterado. Nesses casos, para a correta detecção, é necessário reajustar os limiares de modo a considerar esses novos tons de cores do objeto próximos a luz. Contudo, ao realizar esses ajustes, corre-se o risco da aplicação considerar outros objetos na cena, por possuírem o mesmo tom de cor do objeto.



Figura 5.9: Aproximando o objeto da luz artificial, a tendência é que a detecção também falhe.

Nas situações onde o objeto foi apresentado parcialmente no vídeo, os resultados foram positivos. A Figura 5.10 apresenta alguns casos onde o objeto foi apresentado com alguns obstáculos.



(a) Caso em que detecção foi bem sucedida mesmo sendo mostrada parcialmente no vídeo.

(b) Segundo caso onde o resultado foi positivo para detecção com obstáculos.

Figura 5.10: Casos de sucesso na detecção do objeto em situações adversas.

6 CONCLUSÕES

Este trabalho possibilitou a criação de uma documentação sucinta da área de Visão Computacional, reunindo as principais técnicas utilizadas para o desenvolvimento de aplicações na área. As pesquisas relacionadas ao desenvolvimento desses sistemas tiveram como objetivo agregar as principais ferramentas de linguagem Python utilizadas atualmente. Como resultado dessas pesquisas, descobriu-se que o OpenCV é uma ótima opção para o desenvolvimento de aplicações na área de Visão Computacional em linguagem Python, contendo diversos materiais e livros descrevendo suas funcionalidades. Do mesmo modo, as pesquisas voltadas para o *framework* SimpleCV mostraram a ferramenta como um complemento que simplifica o desenvolvimento de sistemas de Visão Computacional.

Com a utilização de exemplos práticos utilizando essas ferramentas, foi possível apresentar conceitos da área de Visão Computacional normalmente vistos de forma abstrata nos principais materiais desse domínio de estudo. Com isso, o trabalho também permitiu a aproximação entre o domínio da Visão Computacional e profissionais da computação, compondo uma base de consulta para o estudo e desenvolvimento de trabalhos futuros.

Através das pesquisas para o desenvolvimento da aplicação mostrada como estudo de caso, percebeu-se que as ferramentas da área de Visão Computacional para Python estão maduras o bastante para que pessoas com interesse, na área da computação, possam utilizá-las para o desenvolvimento de aplicações.

O estudo de caso obteve sucesso na criação de uma aplicação para detecção e rastreamento de objetos circulares monocromáticos, possibilitando a recuperação de informações pertinentes do mesmo. Dessa forma, possíveis funcionalidades podem ser criadas, como por exemplo estimar o quão distante o objeto está em relação ao vídeo, baseando-se no raio do objeto circular, obtido através dos cálculos feitos pela aplicação.

A base do que foi desenvolvido para a criação do estudo de caso pode ser aplicada em projetos de maior complexidade que visam resolver problemas cotidianos. Exemplos de projetos que podem utilizar a mesma base do que foi desenvolvido são aplicados em áreas de segurança e esportes. Através da detecção de bolas em esportes como futebol e tênis por exemplo, é possível determinar quanto tempo a bola esteve em jogo, ou então concluir se a bola tocou o lado de dentro ou fora da quadra. Em áreas da segurança, mais especificamente a área

de autenticação, a utilização da lógica de detecção de objetos circulares pode ser aplicada para reconhecimento do globo ocular, visando avaliá-lo biometricamente para credenciar a entrada de pessoas em áreas restritas.

Este projeto permite também que outros trabalhos futuros possam ser elaborados, como por exemplo, utilizar o Arduino¹ visando integrar a aplicação com robôs para que possam se guiar a partir do objeto detectado pela aplicação. Ainda, estimando a distância do objeto em relação ao vídeo, pode-se criar uma inteligência capaz de permitir que o robô ande mais rápido se o objeto estiver distante do vídeo, e de mesma forma, ande mais devagar se o objeto estiver próximo.

Conclui-se, então, que o desenvolvimento de aplicações da Visão Computacional pode ser otimizado com a utilização de tecnologias atuais, possibilitando a criação de aplicações na área de Visão Computacional com maior rapidez. Dessa forma, projetos são mais facilmente desenvolvidos, popularizando ainda mais a área de Visão Computacional.

¹Plataforma de hardware e software livre que simplifica a criação e prototipagem de projetos de eletrônica.

REFERÊNCIAS BIBLIOGRÁFICAS

- ALOIMONOS, Y.; ROSENFELD, A. A computer vision. *Science*, 1991.
- ANDREWS, H. C.; PATTERSON, C. L. Outer product expansions and their uses in digital image processing. *IEEE Transactions on Computers*, IEEE Computer Society, Washington, DC, USA, v. 25, n. 2, p. 140–148, February 1976.
- ANTUNES, E. J. *Processamento de Imagens: Uma abordagem interdisciplinar aplicada a correção de prognósticos meteorológicos*. 85 p. Tese (Monografia) — Universidade Federal de Pelotas, Pelotas, RS, Brasil, 1999.
- BAJCSY, R. Active perception. *Proceedings of the IEEE*, v. 76, n. 8, p. 996–1005, August 1988.
- BALLARD, D. H.; BROWN, C. M. Principles of animate vision. *CVGIP: Image Understanding*, Orlando, FL, USA, v. 56, n. 1, p. 3–21, July 1992.
- BIANCHI, R. A. da C. *Uma arquitetura de controle distribuída para um sistema de visão computacional propositada*. Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo, São Paulo, SP, Brasil, 1998.
- BRADSKI, G.; KAEHLER, A. *Learning OpenCV: Computer Vision with the OpenCV Library*. 1st. ed. Cambridge, MA: O'Reilly Media, 2008.
- BUNTING, F. *The Colorshop Color Primer An Introduction to the History of Color, Color Theory, and Color Measurement*. 1 st. ed. [S.l.]: Light Source Computer Images, Inc. An X-Rite Company, 1998.
- CARDANI, D. *Adventures in HSV Space*. 2001. Disponível em: <<http://132.68.58.138/labs/anat/hsvspace.pdf>>. Acesso em: 18 nov. 2012.
- CARVALHO, B. M. de. *Representações de Cores*. 2006. Disponível em: <<http://www.dimap.ufrn.br/~motta/dim102/Cores.pdf>>. Acesso em: 7 nov. 2012.
- CECIRE, K. *Histograms: Construction, Analysis and Understanding*. 2002. Disponível em: <<http://quarknet.fnal.gov/toolkits/new/histograms.html>>. Acesso em: 25 set. 2012.
- DEMAAGD, K. et al. *Practical Computer Vision with SimpleCV: The Simple Way to Make Technology See*. [S.l.]: O'Reilly Media, 2012. (Oreilly and Associate Series).
- ERALP, O. *A Comprehensive Guide to Installing and Configuring OpenCV 2.4.2 on Ubuntu*. 2012. Disponível em: <<http://www.ozbotz.org/opencv-installation/>>. Acesso em: 23 set. 2012.
- FILHO, O. M.; NETO, H. V. *Processamento Digital de Imagens*. 1nd. ed. Rio de Janeiro, RJ, Brasil: Editora Brasport, 1999.
- FOLEY, J. D. et al. *Computer graphics: principles and practice (2nd ed.)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1990.

- FREITAS, C. M. D. S. *Computação Gráfica*. 1998. Apresentação na VI Escola de Informática da SBC Regional Sul, realizado nas cidades de Pelotas (RS), Blumenau(SC) e Curitiba (PR).
- JAIN, A. K. *Fundamentals of digital image processing*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.
- JOLION, J.-M. Computer vision methodologies. *CVGIP: Image Understading*, Academic Press, Inc., Orlando, FL, USA, v. 59, n. 1, p. 53–71, January 1994.
- KILIAN, J. *Simple Image Analysis By Moments*. 2001. Disponível em: <<http://public.cranfield.ac.uk/c5354/teaching/dip/opencv/SimpleImageAnalysisbyMoments.pdf>>. Acesso em: 18 nov. 2012.
- KUMAR, T.; VERMA, K. Article: A theory based on conversion of rgb image to gray image. *International Journal of Computer Applications*, v. 7, n. 2, p. 7–10, September 2010. Published By Foundation of Computer Science.
- MARR, D. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. New York, NY, USA: Henry Holt and Co., Inc., 1982.
- MOLENAAR, G. *What is Integrated Performance Primitives (IPP)*? 2010. Disponível em: <<http://opencv.willowgarage.com/wiki/IPP>>. Acesso em: 22 set. 2012.
- MUSSER, D. R.; STEPANOV, A. A. Generic programming. In: *Proceedings of the International Symposium ISSAC'88 on Symbolic and Algebraic Computation*. London, UK, UK: Springer-Verlag, 1989. (Lecture Notes In Computer Science 358, v. 358), p. 13–25.
- OLIVER, A. *OpenCV vs. Matlab vs. SimpleCV*. 2012. Disponível em: <<http://simplecv.tumblr.com/post/19307835766/opencv-vs-matlab-vs-simplecv>>. Acesso em: 19 out. 2012.
- PADILHA, A. J. *Processamento e Análise de Imagem*. 1999. Disponível em: <<http://paginas-fe.up.pt/~padilha/PAI%2ficheiros/Cap4-ac.pdf>>. Acesso em: 1 out. 2012.
- PUZICHA, J.; HOFMANN, T.; BUHMANN, J. M. Histogram clustering for unsupervised image segmentation. In: *Proceedings of CVPR '99*. Ft. Collins, CO, USA: IEEE Computer Society, 1999. v. 2, p. 2602–2608.
- RAHMAN, A. *Contours - 1 : Getting Started*. 2010. Disponível em: <<http://opencvpython.blogspot.com.br/2012/06/hi-this-article-is-tutorial-which-try.html>>. Acesso em: 1 out. 2012.
- ROTHER, C. et al. Cosegmentation of image pairs by histogram matching - incorporating a global constraint into mrf's. In: *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*. Washington, DC, USA: IEEE Computer Society, 2006. (CVPR '06), p. 993–1000.
- SEYDOUX, R. Deepening project - bazar - vision based tracking library. University of applied sciences Fribourg, 2010.
- STOCKMAN, G.; SHAPIRO, L. G. *Computer Vision*. 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.
- SZELISKI, R. *Computer Vision: Algorithms and Applications*. 1st. ed. New York, NY, USA: Springer-Verlag New York, Inc., 2010.

- TARR, M. J.; BLACK, M. J. A computational and evolutionary perspective on the role of representation in vision. *CVGIP: Image Understanding*, Academic Press, Inc., Orlando, FL, USA, v. 60, n. 1, p. 65–73, July 1994.
- TEAM, T. C. *RGB Heatmap Demo*. 2012. Disponível em: <<http://cloudnsci.fi/wiki/index.php?n=HeatMiner.RgbHeatmapDemo>>. Acesso em: 7 nov. 2012.
- THORNE, B. R. Introduction to computer vision in python. *The Python Papers Monograph*, v. 1, 2009. Disponível em: <<http://ojs.pythonpapers.org/index.php/tppm/article/view/94/111>>. Acesso em: 28 set. 2012.
- TRIVEDI, M. M.; ROSENFELD, A. On making computers see. *IEEE Transaction on Systems, Man and Cybernetics*, v. 19, n. 6, p. 1333–1335, November 1989.
- TRUCCO, E.; VERRI, A. *Introductory Techniques for 3-D Computer Vision*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.
- WEBER, M. *Unsupervised learning of models for object recognition*. Tese (Doutorado) — California Institute of Technology, Pasadena, CA, USA, 2000.
- ZABIH, R.; KOLMOGOROV, V. Spatially coherent clustering using graph cuts. In: *Proceedings of the 2004 IEEE computer society conference on Computer vision and pattern recognition*. Washington, DC, USA: IEEE Computer Society, 2004. (CVPR’04, v. 2), p. 437–444.
- ZHOU, X. S.; HUANG, T. S. Relevance feedback in image retrieval: A comprehensive review. *Multimedia Systems*, v. 8, n. 6, p. 536–544, April 2003.