UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

UMA BÚSSOLA VISUAL PARA ORIENTAÇÃO DE ROBÔS MÓVEIS AUTÔNOMOS BASEADA EM VISÃO COMPUTACIONAL

DISSERTAÇÃO DE MESTRADO

Leonardo de Oliveira Nicorena

Santa Maria, RS, Brasil

2016

UMA BÚSSOLA VISUAL PARA ORIENTAÇÃO DE ROBÔS MÓVEIS AUTÔNOMOS BASEADA EM VISÃO COMPUTACIONAL

Leonardo de Oliveira Nicorena

Dissertação apresentada ao Curso de Mestrado Programa de Pós-Graduação em Informática (PPGI), Área de Concentração em Computação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Mestre em Ciência da Computação

Orientador: Profª. Dr. Giovani Rubert Librelotto

Santa Maria, RS, Brasil

2016

Nicorena, Leonardo de Oliveira

Uma Bússola Visual para Orientação de Robôs Móveis Autônomos baseada em Visão Computacional / por Leonardo de Oliveira Nicorena. – 2016.

77 f.: il.; 30 cm.

Orientador: Giovani Rubert Librelotto

Dissertação (Mestrado) - Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Informática, RS, 2016.

1. Visão computacional. 2. Bússola visual. 3. Robótica. I. Librelotto, Giovani Rubert. II. Título.

E-mail: lnicorena@inf.ufsm.br

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

UMA BÚSSOLA VISUAL PARA ORIENTAÇÃO DE ROBÔS MÓVEIS AUTÔNOMOS BASEADA EM VISÃO COMPUTACIONAL

elaborada por
Leonardo de Oliveira Nicorena

como requisito parcial para obtenção do grau de
Mestre em Ciência da Computação

COMISSÃO EXAMINADORA:

Giovani Rubert Librelotto, Dr.
(Presidente/Orientador)

Cesar Tadeu Pozzer, Dr. (UFSM)

Reiner Franthesco Perozzo, Dr. (UNIFRA)

Santa Maria, 31 de Agosto de 2016.

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

UMA BÚSSOLA VISUAL PARA ORIENTAÇÃO DE ROBÔS MÓVEIS AUTÔNOMOS BASEADA EM VISÃO COMPUTACIONAL
AUTOR: LEONARDO DE OLIVEIRA NICORENA
ORIENTADOR: GIOVANI RUBERT LIBRELOTTO
Local da Defesa e Data: Santa Maria, 31 de Agosto de 2016.

Este trabalho apresenta uma abordagem para obter a orientação de um robô humanóide autônomo, com base em informações provenientes da visão computacional. O método proposto é o de bússola visual, que combina técnicas de processamento de imagens e análise de dados, de modo a estimar a orientação de forma precisa e eficaz. A bússola visual consiste em um modelo de dados circular no qual características visuais extraídas do ambiente são processadas, armazenadas e, posteriormente, comparadas com novas leituras do ambiente de modo a inferir a orientação do robô.

Palavras-chave: Visão computacional. Bússola visual. Robótica.

ABSTRACT

Master's Dissertation
Post-Graduate Program in Informatics
Federal University of Santa Maria

A VISUAL COMPASS FOR ORIENTATION OF AUTONOMOUS MOBILE ROBOTS BASED ON COMPUTER VISION

AUTHOR: LEONARDO DE OLIVEIRA NICORENA
ADVISOR: GIOVANI RUBERT LIBRELOTTO
Defense Place and Date: Santa Maria, August 31th, 2016.

This paper presents an approach for the orientation of a autonomous humanoid robot, based on information from computer vision. The method proposed is a visual compass, which combines image processing techniques and data analysis, in order to estimate the orientation in a accurately and effective way. The visual compass consists of a circular data model in which visual features are processed and extracted from the environment, stored and then compared with new readings of the environment in order to infer the orientation of the robot.

Keywords: Visual Compass. Computer Vision. Robotics.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O trabalho desenvolvido ao longo desta dissertação faz parte de um amplo projeto de pesquisa desempenhado na Universidade Federal de Santa Maria, onde alunos e professores de cursos de graduação das áreas de Engenharias e Tecnologia desenvolvem estudos nas áreas de Robótica, Visão Computacional e Inteligência Artificial. O grupo possui uma equipe de competição, chamada TauraBots, que participa de competições de robótica organizadas pela RoboCup Federation, onde são fomentadas pesquisas nas áreas citadas através do aumento na dificuldade dos desafios das competições a cada ano. A equipe TauraBots, até o momento da publicação deste trabalho, participa das modalidades de futebol de robô organizadas pela RoboCup (ANDERS et al., 2016) nas categorias Teen Size e Kid Size.

A capacidade de auto localizar-se é uma característica básica que espera-se de um agente autônomo e também um dos principais desafios nos campos da Robótica e da Inteligência Artificial (ALTEROVITZ; KOENIG; LIKHACHEV, 2014). Um dos requisitos fundamentais para que se obtenha a localização precisa de um agente é a inferência da orientação com base nos recursos de *hardware* disponíveis. Comumente são utilizados para esta função sensores, tais como *Laser Scanners* ou GPS, entretanto, estes tipos de sensores apresentam alguns inconvenientes, como medições imprecisas em ambientes não artificiais ou dependência de um acesso claro à satélites, por exemplo, o que pode se tornar um problema em ambientes *indoor* ou domiciliares. A localização baseada em odometria também apresenta falhas devido ao acúmulo de erros na leitura das distâncias percorridas e a suscetibilidade à quedas e mudanças arbitrárias de posição (KARLSSON et al., 2005).

Embora abordagens que exploram a utilização destes sensores especiais ou de ambientes projetados artificialmente, na tentativa de obter êxito ao estimar sua orientação, existem diversos cenários onde há limitações que impedem a utilização dos mesmos. Diante disso, espera-se de um sistema confiável a capacidade de inferir ou estimar a localização com base nos recursos disponíveis, que em alguns casos costumam ser apenas dados oriundos de sensores óticos.

Informações provenientes da Visão Computacional se inserem nesse contexto como um fator chave e vêm sendo utilizadas na resolução destes problemas. Do ponto de vista conceitual, é mais desejável que um robô faça uso das mesmas modalidades de sensores que os humanos utilizam (STURM; VISSER, 2009). O mundo real é repleto de indicações visuais que foram especialmente projetados para facilitar a localização, tais como sinalização pública, setas e

outros pontos de referência importantes, como edifícios altos, por exemplo.

Abordagens para localização baseada em informações oriundas da Visão Computacional podem ser vistas em (GIACHETTI; CAMPANI; TORRE, 1998), onde há a utilização de fluxos ópticos e em (KOK; GEORGIOS K METHENITIS; STEENBERGEN, 2013), que se utiliza da identificação de pontos de referência (*landmarks*) em seu propósito. Entretanto, em ambientes onde não há marcadores visuais previamente conhecidos ou onde existam pontos de referência simétricos, por exemplo, a tarefa de obter a orientação do robô torna-se um desafio mais complexo. Logo, existe a necessidade de desenvolver técnicas para tratar deste problema, e é neste contexto que se insere a proposta de bússola visual apresentada no presente trabalho.

Uma Bússola Visual estima a direção do robô comparando pontos distantes, gerados estrategicamente, na imagem ao longo do tempo pela câmera (METHENITIS et al., 2013). A orientação estimada pela bússola e passada ao robô torna-se uma informação essencial na tarefa de inferir a posição do robô em um ambiente que não é previamente conhecido. Bússolas visuais podem ser classificadas em dois tipos: *model-free* e *model-based*. Métodos não baseados em modelos, como o apresentado em (LABROSSE, 2006), estimam a orientação com base nas imagens recentes oriundas da câmera. Em situações específicas, uma posição absoluta pode ser calculada, quando, por exemplo, o robô está alinhado de uma maneira conhecida em relação a outro objeto.

Em métodos de bússola visual baseados em modelos, como o apresentado por (STURM; VISSER, 2009) e estendido por (METHENITIS et al., 2013), um mapa cilíndrico das características visuais da imagem é criado e armazenado no robô para que então se possa obter a orientação absoluta. Este mapa é normalmente gerado em uma fase de inicialização, onde o robô realiza uma volta completa e as imagens são coletadas. Dependendo da abordagem, estes dados podem (ou não) ser atualizados posteriormente, de forma a alterar a precisão na inferência da orientação. Em casos onde as imagens são coletadas apenas na fase de inicialização, a probabilidade de ocorrer algum erro ao estimar a posição, aumenta conforme o robô se movimenta ao longo do ambiente.

Um dos problemas de pesquisa motivados pela RoboCup na preparação de um robô para jogar futebol é a localização do robô em relação ao mapa do ambiente. Uma bússola visual eficaz e sem dependência de *landmarks* no ambiente se faz necessária devido à escassez de recursos computacionais do *hardware* do robô e às dimensões simétricas do campo. Ela pode ser utilizada pelo robô de forma isolada, informando a orientação, ou como uma ferramenta para

auxiliar a intensificar a precisão da localização no campo de jogo. Um exemplo da importância de se possuir uma orientação correta durante o jogo é a necessidade de distinguir o seu próprio gol do gol da equipe adversária, evitando que ocorra situações como gols contra.

Tendo em vista que a compreensão fundamental de uma bússola visual consiste em possibilitar que um robô possa orientar-se em um dado ambiente, previamente conhecido ou não, com base em informações oriundas da câmera. Optou-se por definir um modelo de dados circular, onde informações de uma imagem panorâmica gerada do ambiente possam ser confrontadas com novas imagens recebidas, a fim de obter a orientação estimada do robô.

Dessa forma, têm-se como objetivo principal desta dissertação o desenvolvimento de um sistema de bússola visual, baseado em técnicas de processamento de imagens e visão computacional, capaz de obter a orientação de um robô móvel. Os objetivos específicos são os elencados a seguir:

- Obtenção da orientação do robô com base em informações visuais genéricas, sem a dependência de marcadores ou objetos específicos no cenário;
- Definição de um modelo de dados para análise dos dados obtidos a partir do processamento das imagens obtidas pela câmera;
- Aplicação de técnicas de análise de similaridade de dados para obtenção da orientação estimada do robô.
- Simulação do funcionamento da bússola visual em um cenário de futebol de robôs humanóides;

O presente estudo está organizado da seguinte forma: neste capítulo foi apresentada uma introdução do trabalho com a motivação e suas principais contribuições. O segundo capítulo apresenta uma visão geral sobre Robótica Móvel e o contexto onde este projeto está inserido. No capítulo 3 é apresentado um embasamento teórico para o entendimento das técnicas de Visão Computacional utilizadas e aplicadas no desenvolvimento do trabalho. No capítulo 4 são discutidos alguns trabalhos que se relacionam com o tema desta dissertação, mostrando o que tem sido feito pela comunidade científica nesta mesma linha. O capítulo 5 apresenta a metodologia que propiciou alcançar os objetivos desta dissertação e mostra o processo de desenvolvimento da bússola visual proposta. No capítulo 6 são demonstrados os experimentos e resultados obtidos com a implementação do sistema proposto. Por fim, o último capítulo

traz considerações finais e conclusões sobre o que foi discutido no decorrer deste documento, bem como trabalhos futuros, destacando os temas relacionados na abordagem proposta, nos resultados esperados e obtidos.

2 ROBÓTICA MÓVEL

Há muitas décadas o homem tem vislumbrado a possibilidade de construir máquinas que reproduzam ações humanas e auxiliem na realização das mais diversas tarefas. Desde ações simples como automatizar pequenas incumbências em uma linha de montagem industrial, por exemplo, até ações mais complexas como a realização de missões exploratórias com sondas espaciais. Muito se avançou na área de robótica desde então e esta evolução acompanha os avanços da informática e da eletrônica. Entretanto, está bastante distante a possibilidade de se construir um robô totalmente autônomo que realize ações semelhantes às de um ser humano.

De acordo com (BEKEY, 2005), um robô móvel pode ser definido como um dispositivo mecânico montado sobre uma base não fixa, que age sob o controle de um sistema computacional, equipado com sensores e atuadores que o permitem interagir com o ambiente.

Ainda em outra definição, Bekey afirma que: "Um robô móvel é um dispositivo mecânico montado sobre uma base móvel, que pode ser utilizado em ambientes terrestres, aéreos ou aquáticos. Ele se move através de comandos de um sistema computacional, equipado com sensores e atuadores que permitem interagir com o ambiente."

Segundo (MARTINS-FILHO et al., 2005), a robótica móvel, após décadas de importantes desenvolvimentos, permanece como um interessante assunto de pesquisa devido ao constante crescimento da aplicabilidade em diferentes domínios e seu aspecto relevante na economia e na tecnologia.

A aplicação prática de robôs móveis, associada a diferentes atividades na sociedade vem demonstrando o quanto promissor é o futuro desta área. Por exemplo, seu uso em aplicações domésticas (aspiradores de pó e cortadores de grama robóticos), industriais (transporte automatizado e veículos de carga autônomos), urbanas (transporte público, cadeiras com rodas robotizadas), militares (sistemas de monitoramento aéreo remoto, transporte de suprimentos e de armamento em zonas de guerra, sistemas táticos e de combate) e de segurança e defesa civil e militar (controle e patrulhamento de ambientes, resgate e exploração em ambientes hostis), demonstram grande gama de aplicações atuais dos robôs móveis e os interesses econômicos envolvidos em relação ao seu desenvolvimento e aplicação (WOLF et al., 2009).

2.1 RoboCup

A RoboCup Federation é uma iniciativa global que busca promover pesquisas nas áreas de robótica e inteligência artificial através da proposição de plataformas estimulantes fundamentadas em problemas do mundo real que sejam capazes de atrair a atenção do grande público (ROBOCUP, 2015a).

O objetivo principal da RoboCup é: "Em meados de século 21, uma equipe totalmente autônoma de robôs humanóides jogadores de futebol deve vencer um jogo contra o time de humanos campeão da última Copa do Mundo da FIFA utilizando as regras da FIFA"(ROBOCUP, 2015a). A proposta de vencer o time de humanos campeão do mundo deve ser um dos grandes desafios da comunidade de pesquisadores de robótica e inteligência artificial nas próximas décadas.

2.1.1 Áreas de pesquisa

Além da Inteligência Artificial (IA), desenvolve-se também pesquisa nas áreas de: eletrônica, computação, sistemas embarcados, visão computacional, sistemas de comunicação, mapeamento, sistemas em tempo real, aprendizado de máquinas, educação, arte, tecnologia, locomoção, etc. Além das aplicações diretas, as tecnologias aqui desenvolvidas podem ser aplicadas em áreas como: prótese, órtese, vigilância, controle, dentre muitas outras.

A plataforma mais conhecida da RoboCup é a de futebol de robôs (RoboCup Soccer), que busca desenvolver robôs capazes de cooperar em sistemas multi-agentes para atingir objetivos comuns. Existem diferentes modalidades de futebol. Elas podem variar com a tecnologia de construção dos robôs (por exemplo de acordo com o tipo de locomoção: rodas ou bípedes) e com o tamanho dos robôs e sua tecnologia embarcada (ex: kid size, teen size, adult size).

Contudo, o futebol não é o único foco de desenvolvimento da RoboCup. A RoboCup possui também uma liga especial para o desenvolvimento de robôs para resgate (RoboCup Rescue). Nesta liga, cenários de desastres naturais (tais como áreas de terremotos, por exemplo) são utilizados visando desenvolver sistemas de mapeamento e localização de vítimas.

Uma outra liga da RoboCup, a RoboCup@home, busca desenvolver robôs focados nas tecnologias assistivas, com o objetivo de acelerar o desenvolvimento de tecnologias que tornem os robôs autônomos capazes de auxiliar os seres humanos - particularmente aqueles com algum tipo de necessidade especial - em suas tarefas cotidianas nas residências, tais como: reconheci-

mento e transporte de móveis objetos, auxílio na preparação de refeições, entre outras.

As ligas acima descritas são destinadas prioritariamente a universidades e pesquisadores. Contudo, a RoboCup fomenta também uma liga especialmente desenvolvida para as crianças, a RoboCup Junior, que busca aproximar o jovem do universo da tecnologia através da robótica educacional. (ROBOCUP, 2015b)

2.1.2 RoboCup Soccer - Humanoid League

Na liga de humanóides da RoboCup, robôs bípedes autônomos com características humanas participam de um torneio de futebol contra outras equipes de robôs. A liga fomenta a pesquisa em robótica e computação buscando melhorar questões como caminhada dinâmica, corrida, chute na bola, mantendo o equilíbrio, percepção visual de outros jogadores, bola, elementos do campo e jogo em equipe. A liga é subdividida em 3 subligas, de acordo com o tamanho dos robôs: Teen Size, Kid Size and Adult Size.

A Universidade Federal de Santa Maria (UFSM) possui uma equipe de competição nesta liga, a TauraBots, criada em parceria com alunos e professores do Grupo de Automação e Robótica da Universidade Federal do Rio Grande do Sul (GCAR/UFRGS) e do Grupo de Automação e Robótica Aplicada da UFSM (GARRA/UFSM).

A equipe brasileira dispõe de robôs que utilizam a plataforma open-source DARwIn-OP (*Dynamic Anthropomorphic Robot with Intelligence - Open Platform*) para compor seu time (Figura 2.1). Por ser um projeto open-source, é possível personalizar o *hardware* e o *software* do robô, criando e alterando funcionalidades de modo a obter um melhor desempenho nas atividades pretendidas.

Nas ligas de futebol da RoboCup não é permitido utilizar sensores como bússolas magnéticas ou GPS, pois os robôs devem possuir características próximas às de humanos. Por este motivo a inferência da localização com base na visão torna-se essencial. A abordagem também é valida para robôs que dispõem de tais sensores, seja como forma de redundância ou para aprimorar a localização estimada.

2.2 A Plataforma Robótica - DARwIn-OP

A plataforma aberta DARwIn-OP é desenvolvida pela *Robotics and Mechanisms Laboratory (RoMeLa)*, em parceria com a *Virginia Tech, Purdue, University of Pennsylvania*, e foi

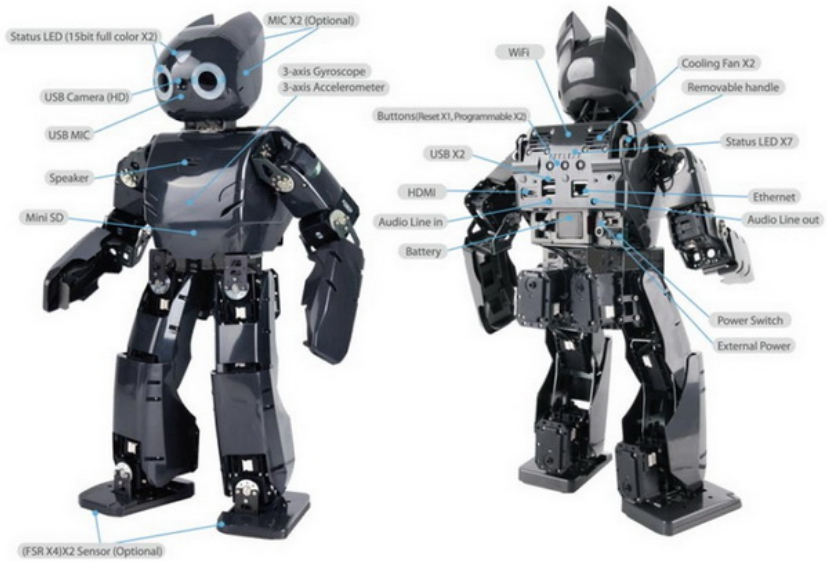


Figura 2.1 – Plataforma aberta DARwIn-OP

apresentada em 2011 por (HA et al., 2011).

Consiste em um projeto totalmente aberto, incluindo códigos-fonte, diagramas de circuitos, arquivos CAD (*Computer Aided Design*) de mecânica e demais partes da plataforma. Foi pensado para possuir uma estrutura expansível, alta performance, manutenção simples, ambiente de desenvolvimento amigável e preços acessíveis.

Os robôs da família DARwIn-OP podem desempenhar movimentos semelhantes aos realizados por humanos, como mover os braços, pernas, cabeça, caminhar e levantar-se ao cair. Também podem executar ações mais complexas, como quando programados para jogar futebol.

A arquitetura de *hardware* consiste em uma placa-mãe padrão de PC, com conexões de rede (*Ethernet* e *Wi-Fi*), portas USB, HDMI, entradas e saídas de áudio, microfone e câmera. Possui um subcontrolador CM-730, com giroscópio, acelerômetro e que gerencia dispositivos como atuadores, sensores, LEDs, botões e demais dispositivos externos de I/O. Fonte de alimentação e baterias para o suprimento de energia também estão presentes, além de alguns itens opcionais, como modulos FSR (*Force Sensing Registers*), para medir a força exercida em partes específicas do conjunto.

No diagrama da Figura 2.2 são mostradas as conexões entre as partes da estrutura de *hardware* do projeto.

A camada de *software* é formada por um *framework* hierárquico, modular e independente. Utiliza a linguagem de programação C++ e é independente do sistema operacional, ou seja, pode ser migrado para outros sistemas operacionais no futuro, inclusive para o Robot

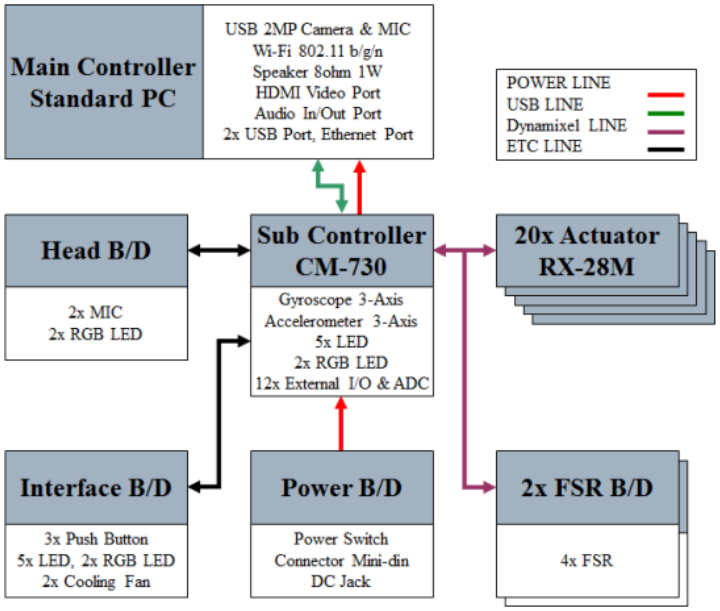


Figura 2.2 – Arquitetura de *hardware* do DARwIn-OP

Operating System (ROS). De maneira geral, o *framework* consiste em módulos de comunicação, movimento, caminhada, percepção, comportamento, visão e diagnóstico. Na Figura 2.3 é mostrado o diagrama de classes do *framework*.

É possível integrar de maneira relativamente simples novas funcionalidades ao *framework*, como algoritmos de comportamento ou visão, pois não há a necessidade de implementar funcionalidades básicas, as quais já estão presentes. Outra vantagem, é que em razão da natureza *open-source* do sistema, os desenvolvedores são encorajados a compartilhar com outros usuários seus códigos.

2.3 Considerações do capítulo

Este capítulo apresentou uma contextualização sobre o meio onde o trabalho está inserido. Inicialmente foi retratada uma visão geral sobre Robótica e suas linhas de pesquisa, bem como sobre a RoboCup, que busca fomentar estudos nesta linha. Falou-se a respeito do grupo de pesquisa, no qual este trabalho está inserido, e por fim, foi apresentada a plataforma robótica para a qual o sistema desenvolvido neste trabalho visa integrar.

No capítulo seguinte é apresentado um embasamento teórico para o entendimento das técnicas de Visão Computacional utilizadas e aplicadas no desenvolvimento deste trabalho.

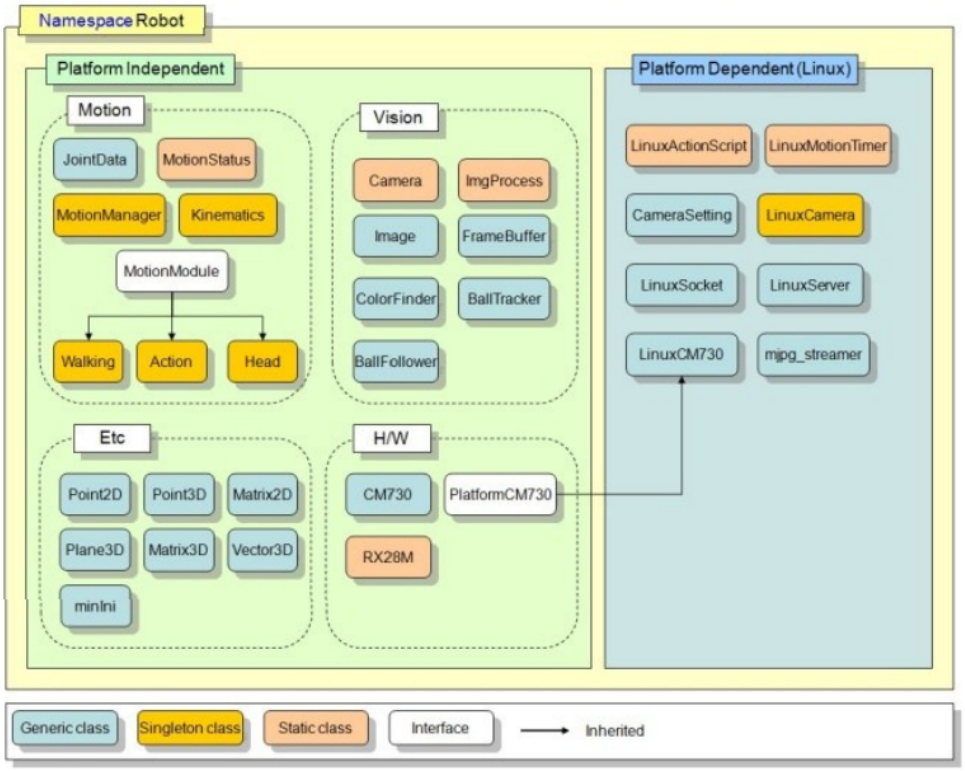


Figura 2.3 – Software *framework* do DARwIn-OP (HA et al., 2011)

3 VISÃO COMPUTACIONAL

Visão computacional é definida por (CONCI; AZEVEDO; LETA, 2007) como o domínio da ciência da computação que estuda e aplica métodos que permitem aos computadores interpretarem o conteúdo de imagens. A área de visão computacional estuda técnicas para extração de informações a partir de elementos de uma imagem. Apesar de ter sido teorizada décadas atrás, foi apenas recentemente que passou a ser empregada de forma mais ampla, devido à evolução da capacidade de processamento dos sistemas computacionais.

Visão Computacional pode ser definida como o estudo da extração de informação de uma imagem; mais especificamente, é a construção de descrições explícitas e claras dos objetos em uma imagem (BALLARD; BROWN, 1982). Difere-se do processamento de imagens na medida que, este implica obter novas imagens, através de transformações aplicadas em uma imagem original. Enquanto, que técnicas de Visão Computacional tratam especificamente da obtenção e manipulação de dados de uma imagem e do uso deles para diferentes finalidades.

Uma abordagem de visão computacional para análise e solução de um problema visual pode ser organizado em etapas sequenciais de aplicação, como mostrado na Figura 3.1.

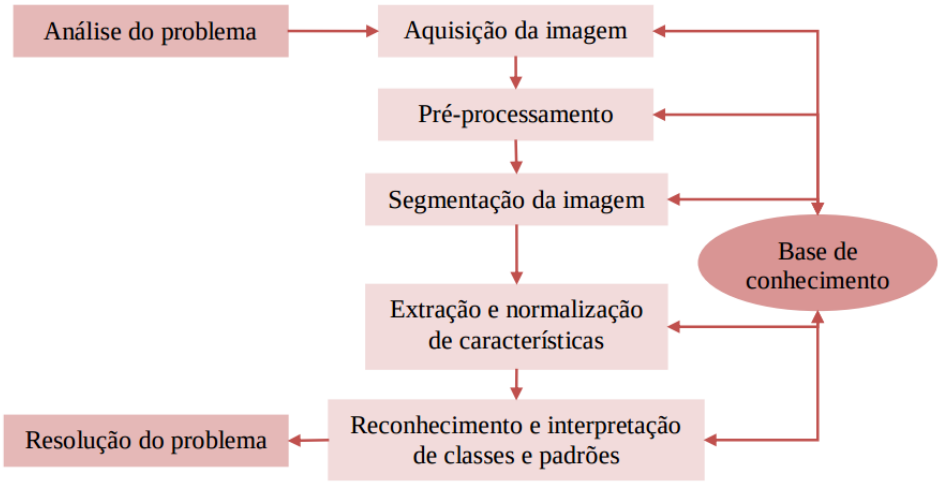


Figura 3.1 – Estrutura sequencial de um sistema de visão computacional (LULIO, 2011)

A estrutura de um sistema de visão computacional contempla estas etapas sequenciais, mediante a descrição de um problema visual: aquisição, pré-processamento, segmentação, normalização de características de interesse, classificação e reconhecimento de padrões, à resolução do problema. Todas interligadas a uma base de conhecimento dos resultados obtidos.

Na próxima seção deste capítulo serão detalhadas as etapas de processamento de ima-

gem que foram aplicadas no sistema proposto e desenvolvido no presente trabalho.

3.1 Processamento digital de imagens

Uma das áreas que mais tem crescido recentemente é o processamento digital de imagens. Esta área vem sendo objeto de crescente interesse por permitir viabilizar um grande número de aplicações que evolvem o aprimoramento de informações pictóricas para interpretação humana e também a análise automática por computador de informações extraídas de uma cena (MARQUES FILHO; NETO, 1999).

Processar uma imagem consiste em transformá-la sucessivamente com o objetivo facilitar a extração da **informação** nela presente (ALBUQUERQUE; ALBUQUERQUE, 2000). Este processo é composto pelas etapas de aquisição, armazenamento, processamento e exibição. Na Figura 5.1 é exibida uma representação visual destas etapas.

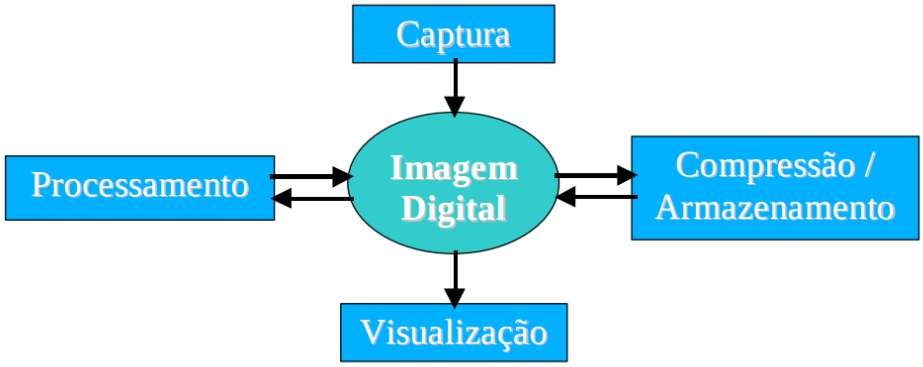


Figura 3.2 – Etapas do processamento de imagem (SCURI, 1999)

Com exceção das etapas de aquisição e exibição, a maioria das funções de processamento de imagens pode ser implementada via *software*, sem a dependência de um *hardware* específico. O resultado deste processamento é uma outra imagem onde as informações nela contidas podem ser interpretadas de forma mais acessível, seja por humanos ou sistemas de visão computacional.

3.1.1 Representação digital de imagens

No processamento digital de imagens geralmente utiliza-se o modelo matricial para representação de imagens, enquanto que na área da computação gráfica geralmente se baseia no

modelo de objetos vetoriais. Neste modelo os objetos são armazenados apenas a partir da descrição das coordenadas de seus vértices, sejam elas espaciais ou planares (três ou duas dimensões, respectivamente). Dessa maneira utiliza-se um sistema de coordenadas Cartesiano, onde os objetos podem ser escalados, rotacionados e transladados com maior liberdade para cada objeto. O modelo matricial utiliza uma matriz de dados para armazenar a informação de cor em cada ponto da imagem, onde o sistema de coordenadas é obviamente uma grade de números inteiros que descrevem a posição na matriz. Portanto, no modelo matricial não há distinção dos objetos contidos na imagem. Além disso, armazenar a matriz que contém a imagem geralmente exige muito mais memória que armazenar a descrição vetorial (SCURI, 1999).

Uma imagem monocromática é uma função bidimensional contínua *f(x,y)*, na qual *x* e *y* são coordenadas espaciais e o valor de *f* em qualquer ponto *(x,y)* é proporcional à intensidade luminosa (brilho ou nível de cinza) no ponto considerado. Como os computadores não são capazes de processar imagens contínuas, mas apenas *arrays* de números digitais, é necessário representar imagens como arranjos bidimensionais de pontos.

Cada ponto na grade bidimensional que representa a imagem digital é denominado elemento de imagem ou pixel. Na Figura 3.3, apresenta-se a notação matricial usual para a localização de um pixel no arranjo de pixels de uma imagem bidimensional. O primeiro índice denota a posição da linha, *m*, na qual o pixel se encontra, enquanto o segundo, n, denota a posição da coluna. Se a imagem digital contiver *M* linhas e *N* colunas, o índice *m* variará de 0 a M-1, enquanto *n* irá variar de 0 a *N-1* (QUEIROZ; GOMES, 2006).

Em uma imagem digital colorida no sistema RGB, um pixel pode ser visto como um vetor cujas componentes representam as intensidades de vermelho, verde e azul de sua cor. A imagem colorida pode ser vista como a composição de três imagens monocromáticas, i.e.:

$$f(x, y) = f\mathrm{R}(x,y) + f\mathrm{G}(x,y) + f\mathrm{B}(x,y),$$

na qual $f\mathrm{R}(x,y)$, $f\mathrm{G}(x,y)$ e $f\mathrm{B}(x,y)$ representam, respectivamente, as intensidades luminosas dos componentes vermelho, verde e azul da imagem, no ponto *(x,y)* (QUEIROZ; GOMES, 2006).

Na Figura 3.4 é apresentada uma imagem no espaço de cor RGB (à direita) e seus canais monocromáticos (à esquerda). A imagem colorida é a composição dos três planos.

3.1.2 Operações morfológicas

A morfologia matemática consiste em extrair as informações relativas à geometria e à topologia de um conjunto desconhecido (uma imagem), pela transformação através de outro

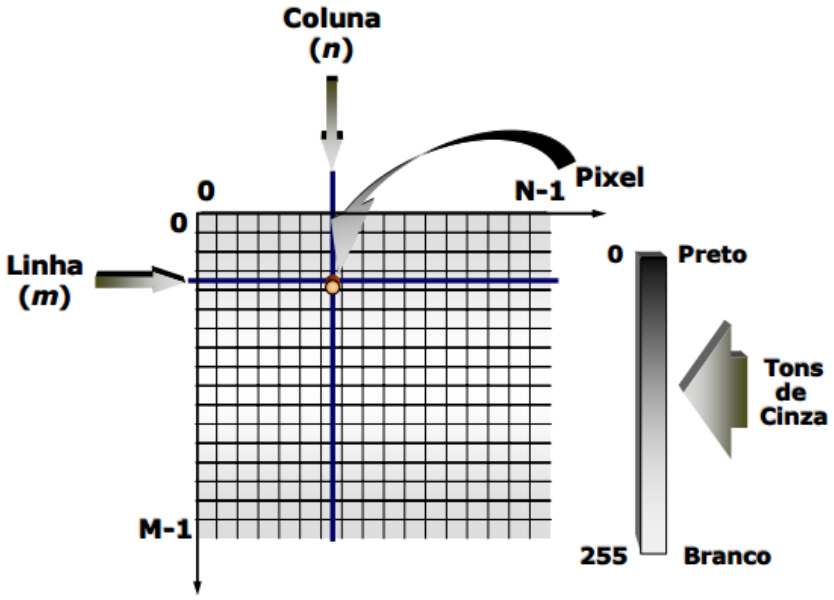


Figura 3.3 – Representação de uma imagem digital bidimensional (QUEIROZ; GOMES, 2006)

conjunto completamente definido, chamado elemento estruturante. Portanto, a base da morfologia matemática é a teoria de conjuntos. Por exemplo, o conjunto de todos os *pixels* pretos em uma imagem binária descreve completamente a imagem (uma vez que os demais pontos só podem ser brancos). Em imagens binárias, os conjuntos em questão são membros do espaço inteiro bidimensional Z^2, onde cada elemento do conjunto é um vetor 2-D cujas coordenadas são as coordenadas (x,y) do pixel preto (por convenção) na imagem. Imagens com mais níveis de cinza podem ser representadas por conjuntos cujos elementos estão no espaço Z^3. Neste caso, os vetores têm três elementos, sendo os dois primeiros as coordenadas do pixel e o terceiro seu nível de cinza (MARQUES FILHO; NETO, 1999).

3.1.2.1 Dilatação e Erosão

As operações de dilatação e erosão são consideradas operações morfológicas básicas. Para melhor compreendê-las serão apresentadas algumas definições da teoria de conjuntos, adaptadas de (MARQUES FILHO; NETO, 1999).

Sejam *A* e *B* conjuntos em Z^2, cujos componentes são $a = (a_1, a_2)$ e $b = (b_1, b_2)$, respectivamente. A translação de *A* por $x = (x_1, x_2)$, denotada $(A)_x$, é definida como:

$$(A)_x = \{c | c = a + x, para\ a \in A\}$$

A reflexão de B, denotada $\hat{B}$, é definida como:

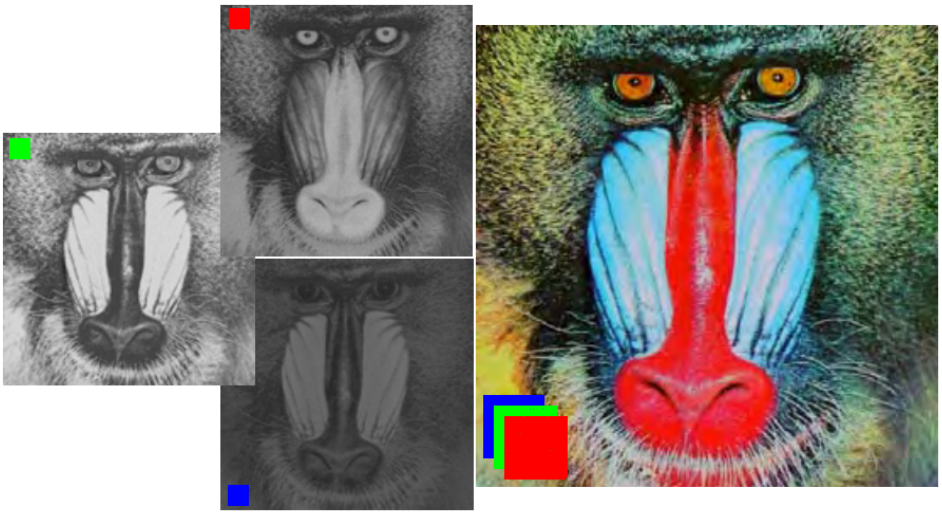


Figura 3.4 – Representação de uma imagem digital bidimensional de 3 canais

$$\hat{B} = \{x | x = -b, para\ b \in B\}$$

O complemento do conjunto A é:

$$A^c = \{x | x \notin A\}$$

Finalmente, a diferença entre dois conjuntos A e B, denotada $A - B$, é definida como:

$$A - B = \{x | x \in A, x \notin B\} = A \cap B^c$$

A Figura 3.5 ilustra geometricamente as definições apresentadas, onde pontos pretos identificam a origem do par de coordenadas. Em (a) é mostrado o conjunto A. A parte (b) mostra a translação de A por $x = x(x_1, x_2)$. O conjunto B é exibido na parte (c), enquanto a (d) mostra sua reflexão em relação à origem. Finalmente, a parte (e) apresenta o conjunto A e seu complemento, enquanto a parte (f) mostra a diferença entre este conjunto A e o conjunto B.

Sejam A e B conjuntos no espaço Z^2 e seja $\varnothing$ conjunto vazio. A dilatação de A por B, denotada $A \oplus B$, é definida como:

$$A \oplus B = \left\{ x | (\hat{B})_x \cap A \neq \varnothing \right\}$$

Portanto, o processo de dilatação consiste em obter a reflexão de B sobre sua origem e depois deslocar esta reflexão de x. A dilatação de A por B é, então, o conjunto de todos os x deslocamentos para os quais a interseção de $(\hat{B})_x$ e A inclui pelo menos um elemento diferente de zero. Com base nesta interpretação, a equação anterior pode ser escrita como:

$$A \oplus B = \left\{ x | [(\hat{B})_x \cap A] \subseteq A \right\}$$

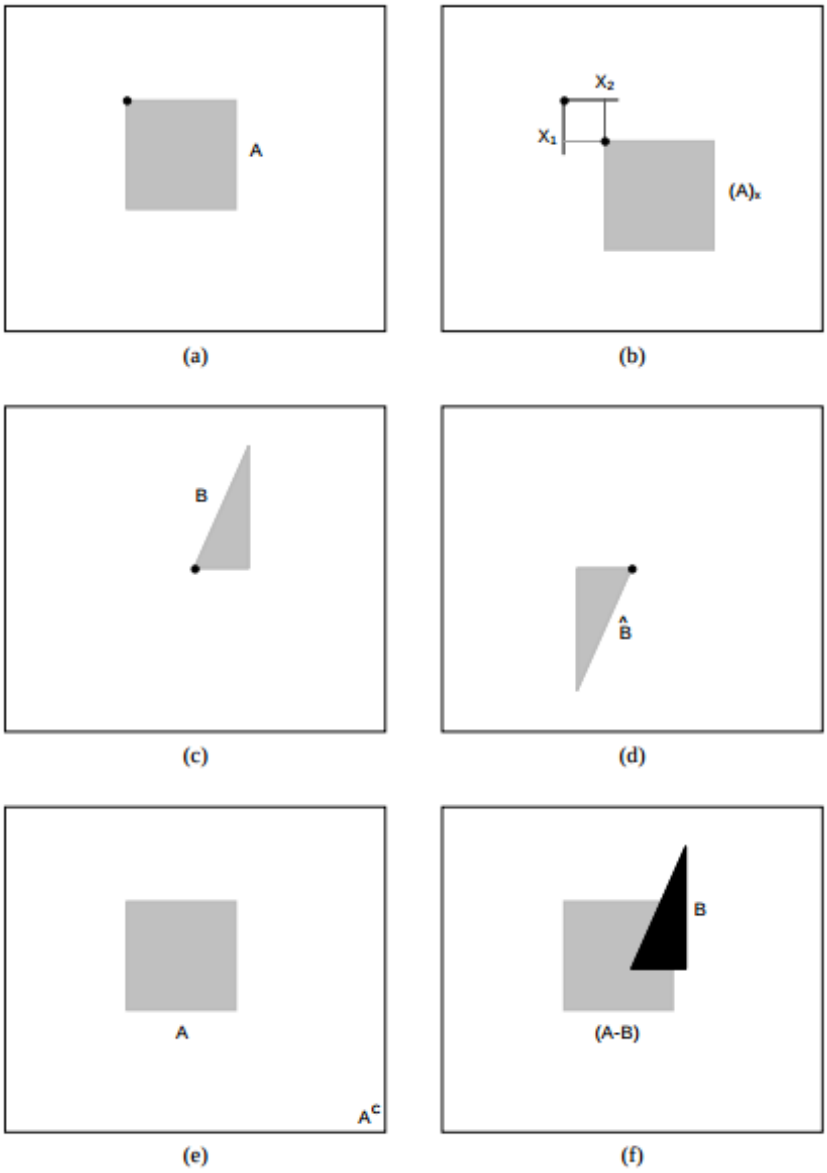


Figura 3.5 – Exemplos de operações básicas sobre conjuntos

O conjunto B é normalmente denominado elemento estruturante.

A Figura 3.6 mostra os efeitos da dilatação de um conjunto A usando três elementos estruturantes (B) distintos. Observar que as operações morfológicas são sempre referenciadas a um elemento do conjunto estruturante (neste caso, o elemento central).

Sejam A e B conjuntos no espaço Z^2. A erosão de A por B, denotada $A \ominus B$, é definida como:

$$A \ominus B = \{x|(B)_x \subseteq A\}$$

o que, em outras palavras significa dizer que a erosão de A por B resulta no conjunto de pontos x tais que B, transladado de x, está contido em A.

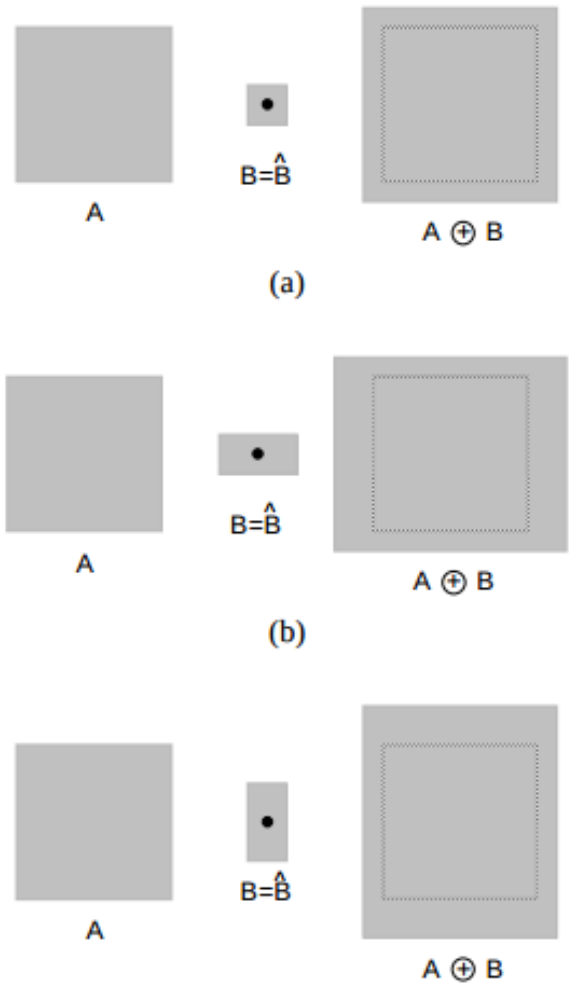


Figura 3.6 – Dilatação

A Figura 3.7 mostra os efeitos da erosão de um conjunto A usando três elementos estruturantes (B) distintos. A dilatação e a erosão são operações duais entre si com respeito a complementação e reflexão. Ou seja,

$$(A \ominus B)^c = A^c \oplus \hat{B}$$

3.1.2.2 Abertura e Fechamento

Como foi mostrado na subseção 3.1.2.1, a dilatação expande uma imagem enquanto a erosão a encolhe. E a partir da combinação destas duas técnicas temos as operações de abertura e fechamento.

A abertura consiste em aplicar uma operação de erosão seguida de uma dilatação e tem por característica suavizar o contorno de uma imagem, quebrar istmos estreitos e eliminar proeminências delgadas. O fechamento, por sua vez, é o processo inverso, ou seja, consiste em

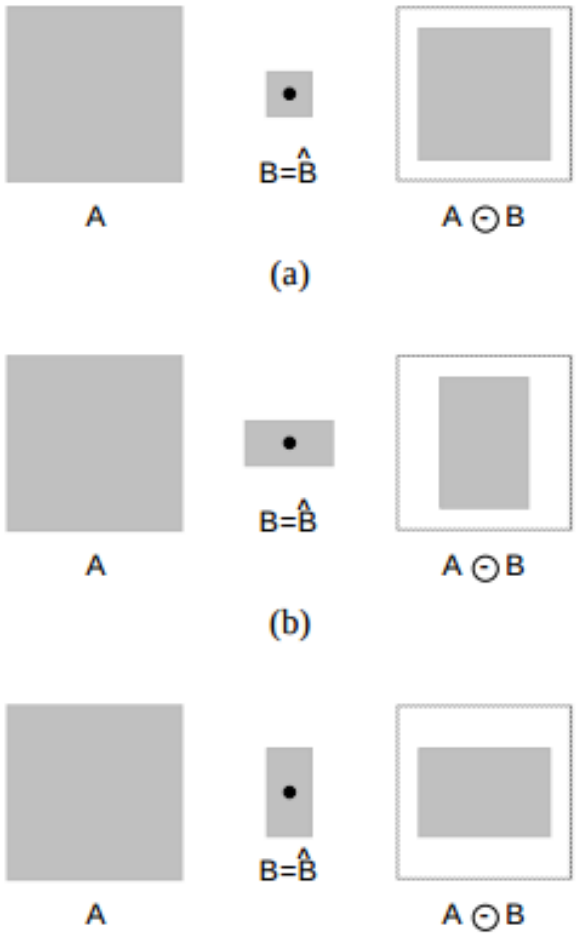


Figura 3.7 – Erosão

aplicar uma operação de dilatação seguida de uma erosão. O fechamento no geral funde pequenas quebras e alarga golfos estreitos, elimina pequenos orifícios e preenche *gaps* no contorno.

A abertura de um conjunto A por um elemento estruturante B, denotada $A \circ B$, é definida como:

$$A \circ B = (A \ominus B) \oplus B$$

o que equivale a dizer que a abertura de A por B é simplesmente a erosão de A por B seguida de uma dilatação do resultado obtido por B.

O fechamento do conjunto A pelo elemento estruturante B, denotado $A \bullet B$, é definido como:

$$A \bullet B = (A \oplus B) \ominus B$$

o que nada mais é que a dilatação de A por B seguida da erosão do resultado pelo mesmo elemento estruturante B.

Na Figura 3.8a, pode ser visualizado um exemplo da operação morfológica de abertura, que atua removendo de ruídos tais como pequenos componentes presentes na imagem. O elemento à esquerda na figura é a imagem original e o elemento à direita é o resultado da transformação.

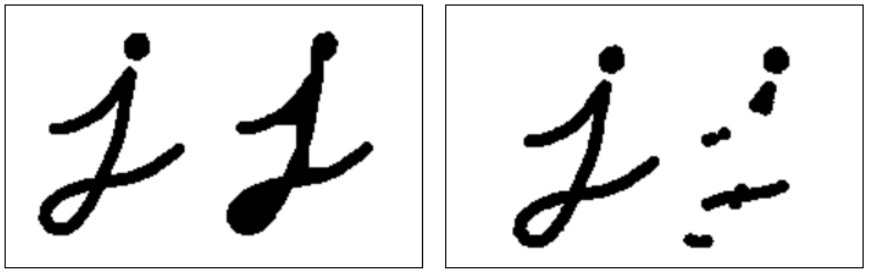


Figura 3.8 – Operações morfológicas de (a) abertura e (b) fechamento (OPENCV, 2016)

Já a operação morfológica de fechamento consiste no processo inverso ao de abertura. É realizada uma operação de dilatação seguida por uma erosão, eliminando pequenos ”buracos” da imagem original, também com o intuito de reduzir ruídos. A operação de fechamento pode ser visualizada na Figura 3.8b.

Ao realizar as operações morfológicas como as citadas acima, deve-se informar as propriedades do elemento estruturante (chamado de *kernel*). É a partir dele que o filtro define quais os pixels vizinhos ao que está sendo processado que serão afetados.

Na Figura 3.9, podem ser visualizados outros dois exemplos de transformações de abertura (a) e fechamento (b) morfológico, porém, com um valor de *kernel* reduzido. É possível identificar que ruídos menores foram identificados e eliminados.

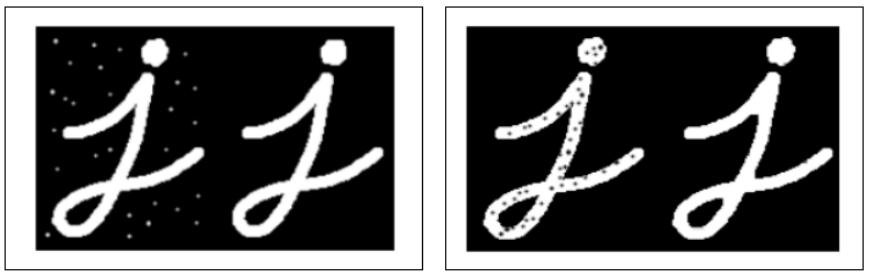


Figura 3.9 – Operações morfológicas de (a) abertura e (b) fechamento com *kernel* reduzido

3.1.3 Redução de cores

A redução de cores de uma imagem consiste em transformar uma imagem que utiliza M *bits* de cor para N *bits*, sendo que $M > N$. Ou seja, se temos uma imagem que possui 256 tons de cinza podemos transformar o espaço de cor desta imagem para 64 tons de cinza. Formalmente a redução de cores é um processo de "discretização" de cor que denominamos *quantização*. A quantização é a transformação sobrejetiva $q : C \rightarrow C'$, de um sólido de cor C utilizando M *bits* de cor para um sólido de cor C' com N *bits* de cor, sendo que $M > N$. Nota-se que o processo de quantização altera a resolução de cor de uma imagem. Utiliza-se a quantização de cores principalmente para reduzir a quantidade necessária de memória para armazenar/representar uma imagem e para transferência entre dispositivos nos quais serão exibidas estas imagens (BENDER, 2003).

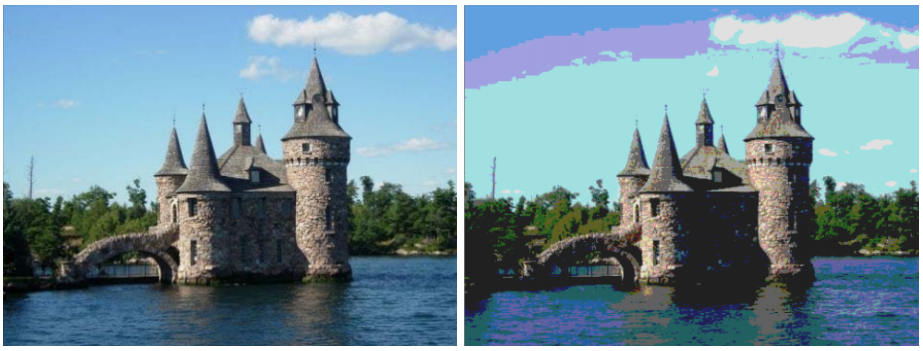
Quando uma quantização de cores é aplicada, o espaço de cores da imagem é dividido em uma série de conjuntos de cores. Cada conjunto é denominado de célula de quantização e a cada célula de quantização está associado um valor constante denominado de nível de quantização. Por exemplo, ao quantizar uma imagem de 256 tons de cinza (8 bits) para 16 tons de cinza (4 bits) serão utilizadas quatro células de quantização. O nível de cada célula poderia ser 32, 96, 128 e 192. Todos os pixels com valor entre 0 e 64 da imagem original receberiam o valor 32 na imagem quantizada, todos os pixels com valor entre 65 e 128 da imagem original receberiam o valor 96 na imagem quantizada e assim por diante. A este tipo de quantização denomina-se *quantização uniforme* e o nível de quantização é dado pela equação:

$$q_i = \frac{c_i + c_{i-1}}{2}, 1 \leq i \leq L$$

Onde q_i é o nível de quantização da célua de quantização *i*, c_i é o valor da célula de quantização e L é a quantidade de cores que se deseja reduzir (VELHO; GOMES, 1994).

Na Figura 3.10 pode-se visualizar resultado da aplicação do processo de redução de cores em uma imagem de 8 bits no espaço de cor RGB. A imagem original (a) de 8 bits, possui aproximadamente 16 milhões de cores, 256 valores possíveis em cada canal (256x256x256), é reduzida por 16, para uma imagem com 4 bits (b), ou seja, com 16 valores possíveis em cada canal (16x16x16) e com um total de 4096 cores distintas .

É possível ainda utilizar métodos mais elaborados ao realizar o processo de discretização de cores de uma imagem para obter resultados visuais mais adequados. Entretanto, dependendo da técnica aplicada, é preciso estar atento ao tempo de processamento da redução de cor, que



(a) Imagem original (b) Imagem discretizada

Figura 3.10 – Aplicação do algoritmo de redução de cores em uma imagem RGB (LAGANIÈRE, 2014)

pode ser bastante elevado de acordo com a complexidade do algoritmo utilizado e não apresentar resultados muito satisfatórios. Em (MIKOLOV, 2008) é apresentada uma abordagem com o uso do algorítimo de *clustering* K-Means.

3.1.4 Histogramas

O histograma de uma imagem registra a distribuição de frequência, ou a distribuição estatística, dos níveis de cinza de uma imagem. Imaginando uma imagem de 8 bits, pode-se pensar em uma tabela com 256 entradas, indexadas de 0 a 255. O nível de cinza mais escuro é representado pelo 0, e o nível de cinza mais claro é representado pelo 255 (MIRANDA, 2006).

O histograma de uma imagem I pode ser definido pela equação:

$$h(I_k) = n_k$$

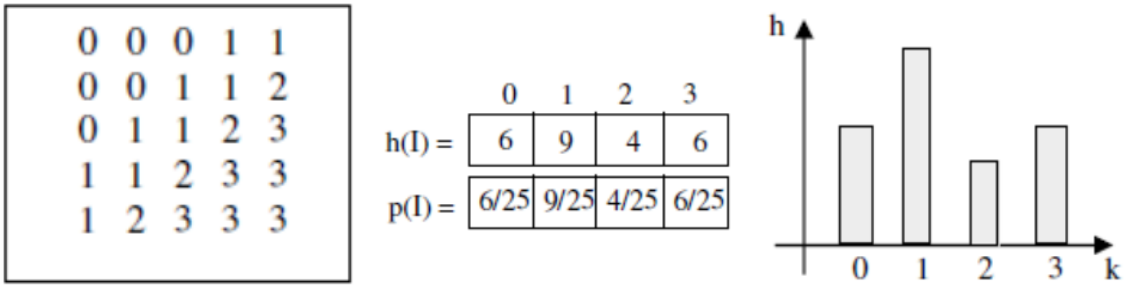
onde I_k é um valor de intensidade k, $(0 \leq k \leq G)$ da imagem I e n_k é o número de pixels na imagem I que possuem a intensidade k. É possível normalizar um histograma, representando os valores em termos de porcentagem, conforme mostrado na equação

$$p(I_k) = \frac{h(I_k)}{n} = \frac{n_k}{n}$$

onde n é o número de pixels da imagem.

A Figura 3.11(a) mostra uma matriz de dados referente a uma imagem com 25 pixels. Neste caso, uma figura com tons de cinza de nível 0 a 3. Na Figura 3.11(b) tem-se a contagem e classificação destes pixels. Como se observa, em h(I), tem-se 6 pixels de nível de cinza 0, 9 pixels com nível de cinza 1, 4 pixels com nível de cinza 2 e 6 pixels com nível de cinza 3.

Na Figura 3.11(c), um histograma gráfico foi gerado com base na contagem e classificação dos pixels. Neste histograma, tem-se 4 bins, que são referentes a cada nível de cinza encontrado na imagem. Observa-se que o nível de cinza mais incidente na imagem foi o nível de cinza 1, logo, o bin 1 aparece com comprimento maior no histograma (SOUZA; CAPOVILLA; ELEOTERIO, 2010).



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

Figura 3.11 – Representação do histograma de uma imagem I (MARENGONI; STRINGHINI, 2009)

Uma operação bastante comum utilizando histogramas é o ajuste dos valores de intensidade de forma a melhorar o contraste em uma imagem. Esta operação é chamada de equalização de histogramas. A ideia desta operação é mapear os valores de intensidade de uma imagem de um intervalo pequeno (pouco contraste) para um intervalo maior (muito contraste) e ainda distribuir os pixels ao longo da imagem de forma a obter uma distribuição uniforme de intensidades (embora na prática isso quase sempre não ocorra). A expressão que fornece um histograma equalizado é apresentada na equação abaixo:

$$h\_eq(k) = \frac{L-1}{MN} \sum_{j=0}^{k} n_j$$

onde k é a intensidade no histograma equalizado, L é o valor máximo de intensidade na imagem, M e N são as dimensões da imagem e n_j é o número de pixel na imagem com valor de intensidade igual a j (MARENGONI; STRINGHINI, 2009).

Na Figura 3.12 é possível visualizar um exemplo de imagem ajustada com a utilização de equalização de histograma. À esquerda tem-se uma imagem, em tons de cinza, e abaixo dela seu histograma. A mesma imagem, após sua equalização, pode ser visualizada no topo à direita e, abaixo, seu histograma equalizado.

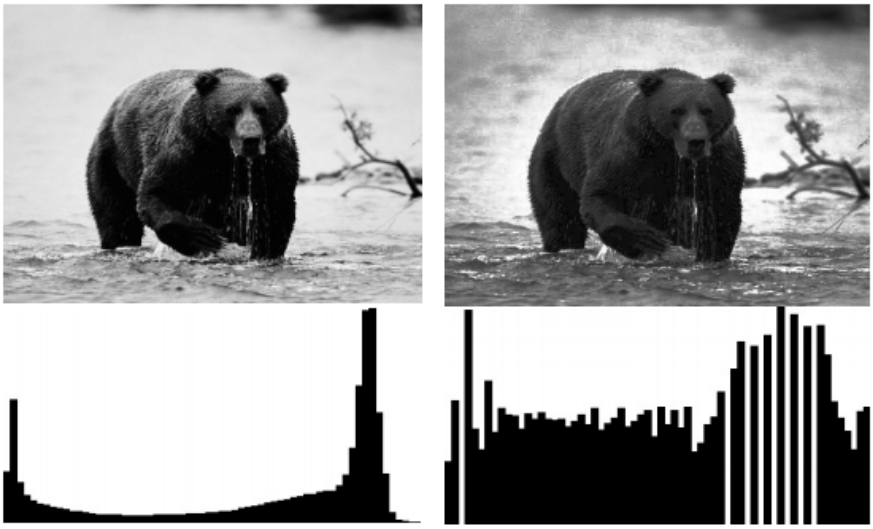


Figura 3.12 – Imagem ajustada com equalização de histograma (MARENGONI; STRINGHINI, 2009)

3.2 Espaços de cores

Um espaço de cor pode ser definido como um método para expressar a cor de um objeto a partir de uma notação específica. A cor é o resultado da percepção da luz (comprimento de onda de 0.4 - 0.7μm) que incide na retina em células foto-receptoras, denominadas cones (GONZALEZ; WOODS, 2008). No olho humano há dois tipos de sensores, cones e bastonetes. Bastonetes são sensíveis a todos os comprimentos de onda e por isso não são capazes de distinguir cor, mas possuem melhor resposta à luz de baixa intensidade (visão noturna). Já os cones são menos sensíveis à luz (visão direta), mas permitem realizar a distinção de cores, uma vez que existem três tipos de cones, sensíveis a comprimentos de onda próximos a 450nm, 550nm e 600nm (nm=10^{-9}m). Isso caracteriza o processo de discriminação de cor do olho, chamado de tricromacidade.

A informação de cor é enviada ao cérebro humano por dois canais, um que percebe a intensidade da luz e outra que percebe as diferenças de cor. Essa decomposição tem o nome de luminância-crominância e é usada em muitas representações de cor (SCURI, 1999). A informação obtida dos bastonetes é chamada de luminância e a informação dos cones é combinada em um único canal denominado crominância. Isto mostra que o sinal de luminância é muito importante para distinguir objetos na imagem, através da acuidade espacial.

Uma cor pode ser decomposta em três componentes independentes: intensidade, matiz e saturação. A intensidade é a responsável pela sensação de brilho. O matiz é quem define

a cor por meio do comprimento de onda. E a saturação, por sua vez, determina o grau de pureza da cor (intensidade do branco). Dessa forma, imagens coloridas são armazenadas em três componentes primários formando um espaço de cor (GONZALEZ; WOODS, 2008). É possível classificar os processos de formação de cores em dois tipos: aditivos, onde vários raios de luz são combinados para formar um novo, e subtrativos, onde um raio de luz passa por um filtro que reduz o comprimento de onda formando uma nova cor. O processo aditivo é encontrado nos monitores de computadores e televisões, enquanto que o subtrativo aparece quando imagens são geradas por projetores, por exemplo. Estes sistemas são conhecidos como RGB (*Red, Green, Blue* - síntese aditiva) e CMY (*Cyan, Magenta, Yellow* - síntese subtrativa).

O espaço RGB é composto pela sensação de soma ponderada do vermelho (*Red*), verde (*Green*) e azul (*Blue*), os quais geram a maioria das cores visíveis. Seu espaço complementar CMY é formado pelo Ciano (C=255-*Red*), Magenta (M=255-*Green*) e Amarelo (Y=255-Blue). Na Figura 3.13 pode-se visualizar o processo de síntese aditiva, que apresenta os seguintes parâmetros: luz igual a zero, ausência de cores primárias resulta na cor preto e a cor primária máxima é o branco; e na síntese subtrativa há: incidência de luz máxima, ausência de cores primárias resulta no branco e a cor primária básica é o preto.

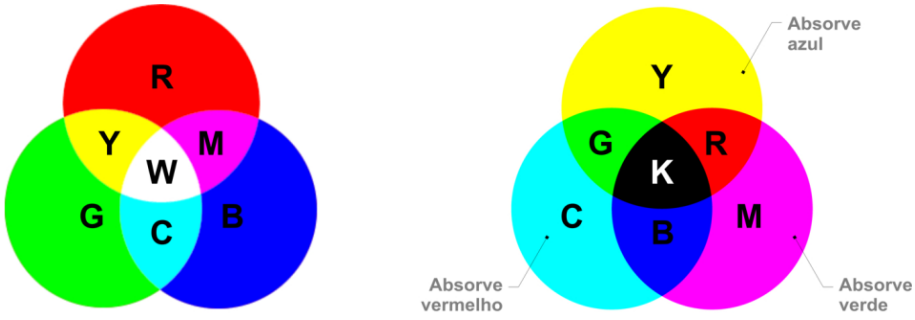


Figura 3.13 – Síntese aditiva e subtrativa para o espaço de cores

Percebe-se que as cores primárias permitem a formação outras novas cores, denominadas cores secundárias. As cores secundárias são a combinação de cores primárias em proporções iguais.

3.2.1 Similaridade de cores

No espaço de cores RGB, quando se calcula a distância no espaço euclidiano, não se obtém uma percepção de similaridade semelhante à percepção humana. Uma cor A que nesta representação é similar a uma cor B com a uma distância N de uma para a outra, ao ser com-

parada com uma cor C que também se encontra a uma distância N em relação à A pode ser totalmente diferente.

Este problema pode ser resolvido com a utilização de espaços de cores que representam todo espectro de cores em um mesmo canal de forma homogênea, como é o caso do espaço de cores HSV. HSV é a abreviatura para o sistema de cores formado pelos componentes (H)ue, (S)aturation e (V)alue.

O canal Hue (ou matiz) assume a representação de todo o espectro de cores, desde o vermelho até o violeta. Seu valor varia de 0 a 360, entretanto em algumas implementações seu valor pode ser adaptado e representado com valores entre 0 e 100% ou 0 e 180, como é o caso da biblioteca OpenCV. A Saturação define a pureza da imagem, ou seja, a quantidade de cor branco presente na mesma e seu valor varia entre 0 e 100%. Por fim, o canal Value representa o brilho presente na imagem e também varia entre 0 e 100%. Na Figura 3.14 pode ser visualizada a representação dos espaço de cores HSV em uma representação circular.

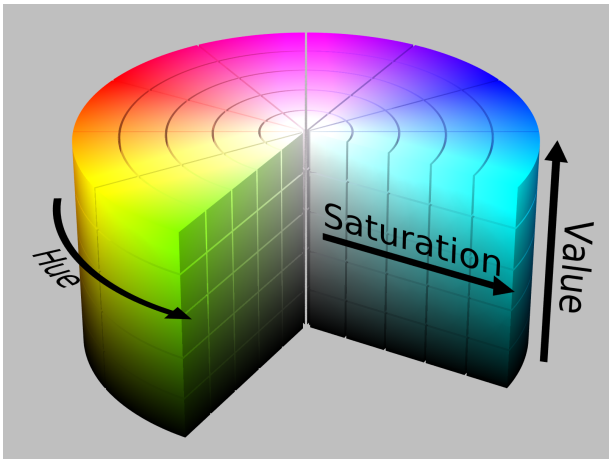


Figura 3.14 – Representação circular do espaço de cores HSV

A informação do espectro de cores presente no canal Hue apresenta uma percepção de similaridade semelhante à do olho humano quando comparadas cores de valores próximos. A utilização deste canal quando se há a necessidade de comparar cores pode ser feita de forma isolada ou em combinação com o canal de saturação, dependendo dos resultados que se espera alcançar.

3.3 Template Matching

É chamada de Template Matching a técnica para buscar a localização de uma imagem dentro de uma imagem maior, baseada em similaridade. O algorítimo percorre a imagem maior

à procura da imagem menor, em busca de similaridade. A imagem menor é movida um pixel por vez, da esquerda para a direita e de cima par baixo e, em cada localização, é aplicado um cálculo com uma métrica que representa o quão similar uma imagem é da outra naquele ponto.

Diferentes métodos para análise de similaridade podem ser aplicados nesta técnica, na Figura 3.15 estão listados os métodos disponíveis na biblioteca do OpenCV. Têm-se R nas fórmulas da figura como a matriz resultante da comparação, a qual será carregada com o nível de similaridade para cada ponto comparado. T e I são as imagens envolvidas no processamento, a imagem que está sendo buscada e a imagem fonte, respectivamente.

$$R(x,y) = \sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2$$

(a) CV_TM_SQDIFF

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

(b) CV_TM_SQDIFF_NORMED

$$R(x,y) = \sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))$$

(c) CV_TM_CCORR

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

(d) CV_TM_CCORR_NORMED

$$R(x,y) = \sum_{x',y'} (T'(x',y') \cdot I(x+x',y+y'))$$

(e) CV_TM_CCOEFF

$$R(x,y) = \frac{\sum_{x',y'} (T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}}$$

(f) CV_TM_CCOEFF_NORMED

Figura 3.15 – Métodos para análise de similaridade do Template Matching

Nas imagens da Figura 3.16 têm-se um exemplo da aplicação da técnica de Template Matching e suas matrizes resultantes para cada um dos métodos citados acima. Nas Figuras 3.16a e 3.16b estão a imagem que será buscada e a imagem onde ocorrerá a busca. Na Figura 3.16c está o resultado das comparações com os diferentes tipos de análise de similaridade. Na primeira linha estão os resultados dos métodos SQDIFF, CCORR e CCOEFF os resultados dos mesmos métodos em suas versões normalizadas. Nota-se que na primeira coluna quanto mais

similar as imagens comparadas, mais escuro o pixel na matriz resultante. Enquanto que nas duas colunas à direita, o pontos com maior similaridade aparecem mais claros.

(a) Imagem a ser buscada

(b) Imagem onde ocorrerá a busca

(c) Matrizes resultantes da análise

Figura 3.16 – Métodos para análise de similaridade do Template Matching

3.4 A biblioteca OpenCV

A biblioteca OpenCV foi desenvolvida pela Intel e possui mais de 500 funções [4]. Foi idealizada com o objetivo de tornar a visão computacional acessível a usuários e programadores em áreas tais como a interação humano-computador em tempo real e a robótica. A biblioteca

está disponível com o código fonte e os executáveis (binários) otimizados para os processadores Intel. Um programa OpenCV, ao ser executado, invoca automaticamente uma DLL (Dynamic Linked Library) que detecta o tipo de processador e carrega, por sua vez, a DLL otimizada para este. Juntamente com o pacote OpenCV é oferecida a biblioteca IPL (Image Processing Library), da qual a OpenCV depende parcialmente, além de documentação e um conjunto de códigos exemplos (MARENGONI; STRINGHINI, 2009).

As funções presentes na biblioteca OpenCV estão relacionadas às várias áreas da Visão Computacional, como: processamento de imagens, detecção de movimento e rastreamento, reconhecimento de padrões em imagens e calibração de câmera. São funções de alto nível, que tornam mais fácil a resolução de problemas complexos em Visão Computacional.

3.5 Considerações do capítulo

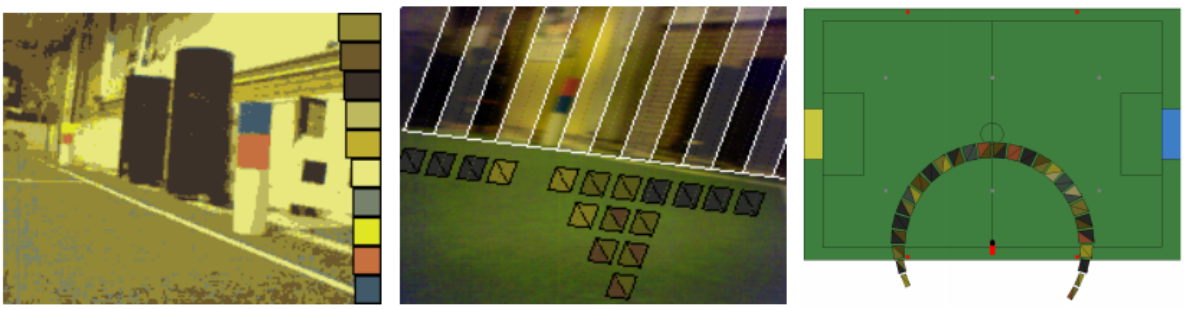
Neste capítulo foi apresentada uma revisão bibliográfica das técnicas e conceitos de visão computacional utilizadas no desenvolvimento do sistema proposto. No capítulo seguinte há uma discussão sobre os trabalhos publicados que apresentam propostas semelhantes à apresentada nesta dissertação.

4 TRABALHOS RELACIONADOS

Este capítulo aborda trabalhos relacionados com a proposta desta dissertação. Ou seja, trabalhos que apresentam abordagens computacionais para obter a orientação de um robô com base em informações oriundas da câmera. Na proposta apresentada por Sturm (STURM; VISSER, 2009), as características extraídas das imagens capturadas são armazenadas em um mapa cilíndrico, que simula uma representação de 360º do ambiente ao redor do robô. As imagens capturadas posteriormente são processadas e têm suas características confrontadas com as presentes no mapa cilíndrico.

As informações extraídas da imagem e utilizadas para inferir a orientação do robô são as quantidades de trocas de cores presentes em segmentos verticais das imagens no espaço de cor RGB. É realizado um processo de discretização na cena para que a quantidade de cores diferentes presentes na imagem seja reduzida. As transições de cores contidas em cada coluna da imagem e suas frequências são armazenadas em um vetor, no qual são aplicados algorítimos de filtro de dados para estimar a orientação correta. Para este fim foram aplicadas técnicas de Filtro de Kalman (WELCH; BISHOP, 2001), Filtro de Bayes (FOX et al., 2003) e Monte Carlo Localization (FOX et al., 1999).

Na Figura 4.1 podem ser vistas as etapas de inicialização da bússola visual de Sturm. Na figura à esquerda (a) é exibida uma imagem capturada do campo de futebol do laboratório com suas cores reduzidas para um total de 10. Em (b) é mostrada a imagem segmentada em colunas de 4,5º (de uma câmera com abertura horizontal padrão de 45º) e suas transições de cores. E à direita é exibida uma representação da bússola já calibrada em relação ao campo de jogo.



(a) (b) (c)

Figura 4.1 – Modelo de dados circular com as transições de cores definidas por Sturm e Visser (STURM; VISSER, 2009)

Esta abordagem apresentou resultados satisfatórios para o que se propôs, entretanto há uma dependência em *landmarks* pré-definidas, os identificadores azul e vermelho na Figura 4.1 (a), fator este que era utilizado em edições passadas de torneios da RoboCup para auxiliar na orientação. O espaço de cores utilizado para manipular as imagens também não era o ideal para se identificar similaridade, pois não houve esta preocupação devido à utilização das *landmarks* do campo.

Uma outra abordagem, apresentada por Kok et. al (GUDI A., 2013), estende a proposta de Sturm e Visser de modo que há um conjunto de bússolas que são inicializadas em regiões específicas do campo, visando obter resultados mais apurados ao estimar a orientação do robô. Esta técnica é chamada pelos autores de ViCTOriA (Visual Compass To Orientate Accurately). Na Figura 4.2 é mostrado o campo de jogo dividido em uma matriz com 15 células, onde em cada uma um modelo de dados semelhante ao definido por Sturm e Visser é carregado. Dessa maneira os autores conseguem obter a orientação de forma mais apurada, pois a comparação entre as informações extraídas das imagens sofrem menos impacto de distorções ou mudanças de luminosidade de pontos distintos do campo.

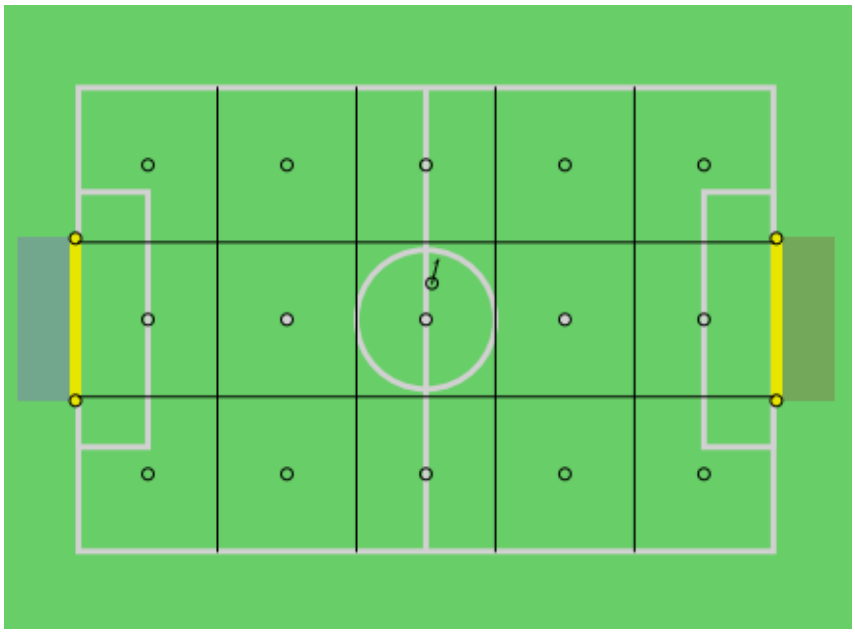
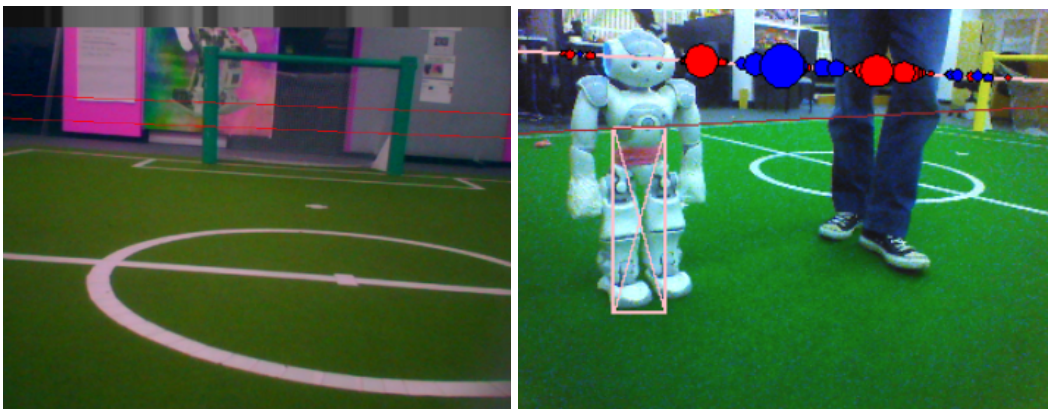


Figura 4.2 – Modelo utilizado em ViCTOriA

Os robôs do mesmo time trocam informações entre si para inicializar as diferentes bússolas dispostas pelo campo. Dessa maneira é possível carregar e atualizar os dados das bússolas em todos os robôs de forma simultânea. A discretização de cores das imagens é realizada utilizando a técnica de *clustering k-means*, obtendo-se um total de 8 cores distintas, no espaço de

cores YUV422, que possui uma percepção mais próxima à humana.

Investigou-se também a abordagem proposta por Anderson e Hengst (ANDERSON; HENGST, 2013), onde a bússola consiste em extrair características de uma linha do horizonte da visão do robô e utilizá-las para estimar a orientação ao confrontá-las com características extraídas de outras imagens capturadas. Uma linha de 30 pixels da imagem em preto e branco é analisada e as informações nela contidas são coletadas e armazenadas em seu modelo de dados. Os autores adaptaram o algorítimo de Speeded-Up Robust Features (SURF) (BAYA et al., 2008) para trabalhar com uma dimensão (1D SURF), de modo a otimizar seu desempenho. Esta abordagem apresenta uma boa margem de acertos ao inferir a orientação em relação às outras analisadas quando há obstáculos ao redor do robô. Na Figura 4.3 (a) é mostrada a linha de pixels de onde as características são extraídas e em (b) o resultado da comparação com outra imagem. Nota-se em (b) que os objetos estáticos do horizonte apresentam uma margem de acerto elevada (círculos azuis).



(a) (b)

Figura 4.3 – Detecção de características com 1D SURF

Por fim, no estudo realizado por Burke e Vardy (BURKE; VARDY, 2006), são investigadas diversas técnicas de correlação 1D e 2D para análise de características armazenadas em bússolas visuais. Nas comparações realizadas levou-se em consideração a efetividade e eficiência de cada algorítimo. Os métodos mais sofisticados mostraram-se mais eficientes mas com pouco variação na precisão. Outra conclusão à que os autores chegaram foi de que os métodos de correlação de fase 2D apresentam melhores resultados quando a bússola é utilizada para obter a orientação em distâncias curtas, enquanto que para distâncias mais longas os métodos de correlação de fase 1D se mostraram melhores tanto em precisão quanto em eficiência.

A partir dos trabalhos encontrados na literatura percebe-se a ausência de uma aborda-

gem de bússola visual que possa ser aplicada de forma abrangente em outros contextos. Na maioria dos casos analisados, as propostas atendem apenas cenários específicos, para os quais foram projetadas. Na proposta desenvolvida neste trabalho, descrita no capítulo 5, levou-se em consideração os pontos positivos e negativos encontrados nos resultados dos estudos analisados, de modo que possibilitasse desenvolver uma bússola visual que se mostrasse ágil e precisa não apenas no cenário onde está inserida.

4.1 Considerações do capítulo

Neste capítulo foi realizada uma discussão acerca dos trabalhos presentes na literatura que possuem propostas semelhantes ao desenvolvido. No capítulo seguinte é detalhada a metodologia e o desenvolvimento da abordagem proposta.

5 BÚSSOLA VISUAL

Neste capítulo é apresentado o projeto e a descrição de um sistema para obtenção da orientação de um robô utilizando uma abordagem chamada de bússola visual. As principais características e funcionalidades deste sistema bem como as etapas de processamento, métodos e algoritmos utilizados em sua construção são descritos a seguir.

5.1 Contexto

Estimar a posição de um robô em relação a um mapa do ambiente é uma tarefa complexa. Uma maneira de obtê-la é através da análise de informações oriundas do processamento dos dados que os sensores disponíveis podem perceber. A precisão da localização estimada depende da qualidade das informações que se têm do ambiente e das técnicas utilizadas para processá-las. O sistema desenvolvido se insere neste contexto como uma ferramenta para auxiliar na precisão do processo de localização do robô. A orientação em relação ao ambiente é inferida pela bússola e, com base nestas informações, o sistema autônomo melhora sua capacidade de obter a localização.

A bússola visual permite que obtenha-se rapidamente a orientação do robô em relação ao ambiente comparando as informações visuais contidas em um *frame* capturado com os dados armazenados previamente. Estes dados são características extraídas de imagens dos arredores do robô e ficam reunidos em um modelo de dados circular que mapeia uma área de 360 graus, simulando uma visão panorâmica do ambiente.

Com a abordagem apresentada não é necessário que haja um conhecimento prévio de características do meio, podendo dessa forma ser aplicado em diversos cenários além do campo de jogo da RoboCup, como projetos para ambientes *indoor*, por exemplo. Para este fim, basta alterar os parâmetros de processamento de imagem do sistema para adaptá-lo à outros contextos.

5.2 Características e requerimentos do sistema

Como é habitual em projetos que envolvem técnicas de visão computacional, a concepção deste trabalho inicia-se com a definição das técnicas que são necessárias quanto ao processamento de imagens. As imagens obtidas pela câmera presente no robô precisam receber o tratamento adequado de forma que possam ser extraídos dados relevantes à inferência de sua

orientação. Após ajustar-se à resolução das imagens provenientes da câmera, o sistema as converte para o espaço de cores mais apropriado para o tipo de análise que será realizada e executa operações morfológicas para facilitar seu processamento. A imagem também passa por um algoritmo de redução de cores, com o intuito de reduzir o custo computacional a ser aplicado nas etapas posteriores.

As etapas definidas para a fase de processamento de imagem dos *frames* capturados podem ser visualizadas na imagem 5.1.

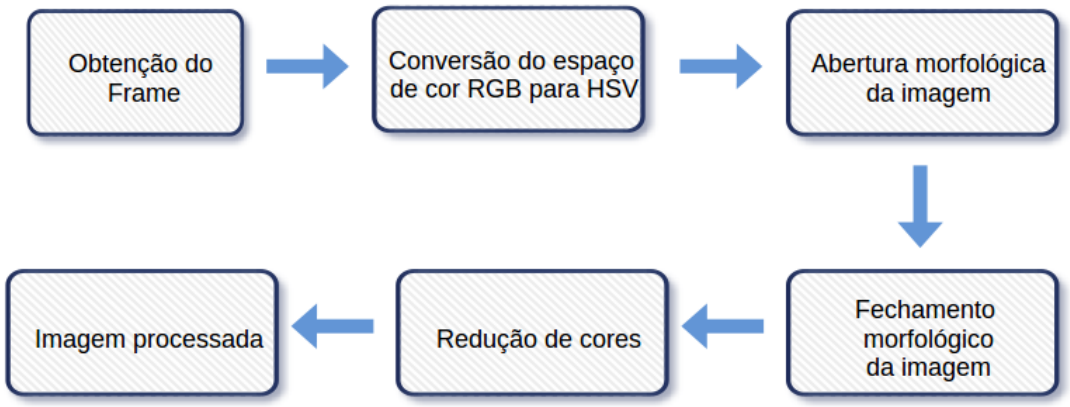


Figura 5.1 – Etapas da fase de processamento de imagens

A partir da imagem resultante do processamento realizado é segmentada verticalmente em colunas e, de cada coluna, extrai-se as características chave que serão armazenadas e comparadas com o modelo de dados da bússola visual. As características que são buscadas durante esta análise são as classes de cores, suas transições e histogramas.

O modelo de dados definido para manipulação das informações da bússola é uma estrutura de dados circular, inspirada nas propostas de (STURM; VISSER, 2009) e (KOK; METHENITIS; NUGTEREN, 2013), onde as informações de cada coluna de uma imagem panorâmica de 360º são armazenadas. Em cada coluna, chamada de *Feature*, armazena-se as informações de um segmento vertical da imagem de uma câmera CMOS com abertura horizontal de 45º. As dimensões da imagem, altura e largura, são definidas nos parâmetros do sistema, bem como a resolução dos *frames* capturados e seus ângulos de abertura. Dessa forma o sistema pode adaptar-se à equipamentos de captura óptica com propriedades distintas. Na etapa de calibração o robô também deve passar à bússola os ângulos verticais e horizontais da cabeça do robô no momento da captura para que o modelo de dados seja inicializado. Na Figura 5.2 é possível visualizar a forma como as Features e as informações extraídas da imagem são armazenadas.

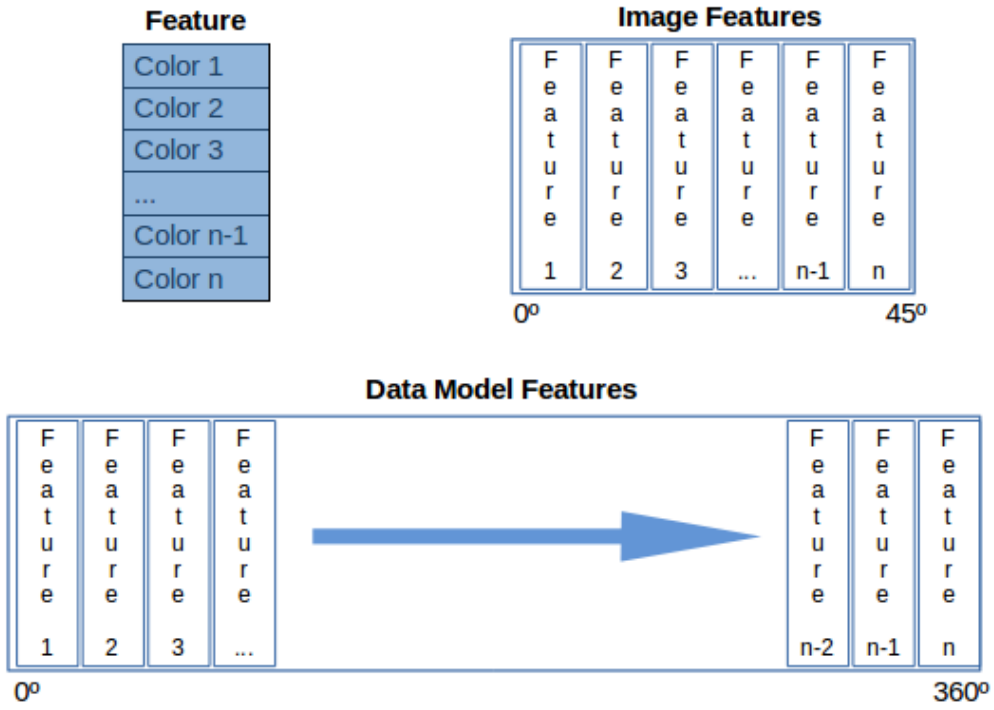


Figura 5.2 – Representação visual do modelo de dados proposto

O sistema possui uma rotina de inicialização, chamada de **Calibração** da bússola, na qual são passadas as imagens iniciais do ambiente ao redor do robô, juntamente com os dados da posição da câmera no momento da captura. Assim, é realizado o carregamento inicial da bússola e, a partir daqui, é possível inferir a orientação ao obter os novos *frames* e comparar com as informações armazenadas.

O processo de inferência da orientação do robô consiste na comparação entre as características armazenadas no modelo de dados e os *frames* capturados. Com base nas transições entre as classes de cores definidas e nos histogramas do ambiente, o sistema retorna a provável orientação com um grau de certeza que varia entre 0 e 1.0, onde 1.0 significa 100% de compatibilidade entre as informações comparadas.

Com a resposta fornecida pela bússola, fica a cargo do sistema de comportamento do robô decidir qual decisão será tomada a seguir com relação às próximas ações do robô. E tal decisão pode ser tomada levando em conta o grau de certeza da bússola e as últimas consultas aferidas.

5.3 Funcionalidades do Sistema

Para que o sistema atenda o objetivo a que se propõe, as seguintes funcionalidades são requeridas e elencadas a seguir:

- Extrair informações relevantes através de técnicas de processamento de imagem;
- Armazenar estas informações de modo que possam ser prontamente consultadas e analisadas;
- Possuir uma rotina para inicialização da bússola;
- Comparar os dados armazenados na bússola com os extraídos das novas imagens;
- Inferir a orientação do robô junto com um grau de certeza.

Na Figura 5.3, o processo como um todo é ilustrado utilizando a notação BPMN (*Business Process Model and Notation*) em forma de um diagrama independente de metodologia.

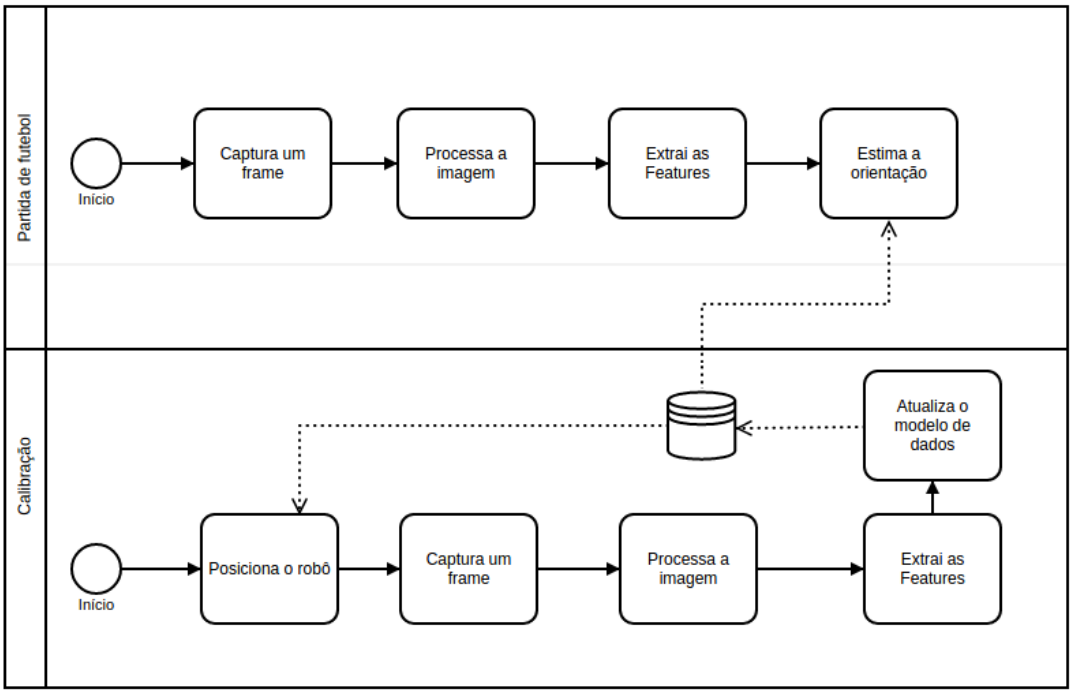


Figura 5.3 – Etapas do sistema para obtenção da orientação

5.4 Etapas de processamento de imagem

Um dos principais requisitos do sistema desenvolvido é a capacidade de extrair informações relevantes das imagens oriundas da câmera do robô para uma posterior análise. Este processo é tido como parte fundamental do projeto de visão computacional e consiste em definir quais filtros de processamento de imagem devem ser utilizados e aplicá-los.

A sequência de técnicas de processamento de imagem desenvolvida neste projeto foi definida com base nos resultados apresentados em testes realizados com diversas combinações de filtros, sempre com o intuito de tornar as imagens mais homogêneas e facilitar o processo de extração de informações. Dessa forma, foi possível reduzir os ruídos presentes nos *frames* capturados e demandar um menor poder de processamento na execução das etapas posteriores.

Nas subseções a seguir, as técnicas desenvolvidas e aplicadas no presente trabalho, são apresentadas e detalhadas.

5.4.1 Operações morfológicas

Foram aplicadas no sistema desenvolvido as operações morfológicas de **Abertura** (*image opening*) e **Fechamento** (*image closing*). Ambas baseiam-se em dois operadores mórficos fundamentais, detalhados no Capítulo 3, que são as operações de Dilatação e Erosão.

Para definir a forma e tamanho ideal do elemento estruturante utilizado nos filtros de transformação morfológica, deve-se levar em conta as características da imagem original e o resultado que se espera. Normalmente este valor é definido com base na aplicação de testes com diferentes valores e análise dos resultados.

No sistema desenvolvido, optou-se por realizar uma operação de abertura seguida de uma de operação de fechamento de modo a remover pequenos elementos (ruídos) presentes no *frame* e, então, conectar os elementos próximos, com o objetivo de obter um conjunto mais regular e menos rico em detalhes que o original.

Na Figura 5.4 é apresentada uma imagem do cenário de uma partida de futebol de robôs. Após aplicar o filtro de morfológico de abertura, a imagem resultante é mostrada na Figura 5.5.

Nota-se que pequenos elementos que destoam do restante (*pixels* ao redor) da imagem são removidos. Mas, apesar de apresentar uma redução no nível de detalhes da imagem, esta redução ainda não se mostra suficiente para realizar a análise esperada.

Na Figura 5.6, o filtro de fechamento é aplicado na imagem original. É possível notar

Figura 5.4 – Imagem do campo de futebol de robôs

que pequenos buracos presentes na imagem original são fechados. Entretanto, também não se obtém um resultado satisfatório, pois os elementos do cenário continuam com um grau elevado de detalhes e suscetíveis à alterações.

Com a aplicação dos filtros de abertura e de fechamento na mesma imagem, obtém-se um resultado mais próximo do esperado, como pode ser visto na Figura 5.7.

Os algoritmos de transformação morfológica foram implementados utilizando a biblioteca OpenCV. A seguir estão apresentados os códigos fonte desenvolvidos para aplicação destes filtros.

O método abaixo realiza o carregamento de uma imagem e aplica a transformação de abertura morfológica seguida pela transformação de fechamento.

```
1
2  void ColorProcessing::doOpeningClosing() {
3
4          Mat image;
5          image = imread("/location/to/image.jpg");
6
7          imshow("BGR Original Image", image);
8
9          this->imageOpening(image, 5);
10         this->imageClosing(image, 5);
11
12         imshow("HSV to BGR Image", image_back_to_bgr);
13
14 }
```

Nos trechos de código fonte abaixo, estão as rotinas chamadas no método principal.

Figura 5.5 – Operação morfológica de abertura em uma imagem do campo de futebol de robôs com valor do kernel igual a 5

```
1
2 void ColorProcessing::imageOpening(Mat& image, int size) {
3
4         //morphological opening
5         erode(image, image, getStructuringElement(MORPH_ELLIPSE, Size
              (size, size)));
6         dilate(image, image, getStructuringElement(MORPH_ELLIPSE,
              Size(size, size)));
7
8 }
```

```
1
2 void ColorProcessing::imageClosing(Mat& image, int size) {
3
4         //morphological closing
5         dilate(image, image, getStructuringElement(MORPH_ELLIPSE,
              Size(size, size)));
6         erode(image, image, getStructuringElement(MORPH_ELLIPSE, Size
              (size, size)));
7
8 }
```

O resultado obtido com a combinação dos filtros morfológicos aplicados apresentou-se satisfatório, porque mantiveram-se as características principais dos elementos presentes na imagem e foi alcançado um nível aceitável de redução de ruídos.

Na seção seguinte está descrita a próxima etapa da estratégia de processamento de imagem aplicada, chamada de redução de cores, que é executada com o intuito de obter um nível maior de homogeneidade na imagem.

Figura 5.6 – Operação morfológica de fechamento em uma imagem do campo de futebol de robôs com valor do *kernel* igual a 5

5.4.2 Redução de cores

Mantendo o propósito de reduzir a complexidade dos elementos presentes no *frame* e possibilitar a análise posterior, nesta etapa é aplicado um algoritmo para redução da quantidade de cores da imagem. Nesta abordagem as cores resultantes são chamadas de classes de cores.

O algoritmo de redução de cores recebe um parâmetro N que é o fator de redução da imagem. E com base neste fator é possível determinar a quantidade de classes de cores que a imagem resultante poderá possuir. Cada pixel da imagem têm sua cor alterada para a classe de cor mais próxima da cor original.

O processo básico de redução de cores funciona da seguinte forma: cada um dos 3 canais, de cada pixel da imagem, tem seu valor divido por N (fator de redução) e, então, a parte inteira do resultado da divisão é multiplicado por N, obtendo-se o maior múltiplo de N menor que o valor inicial; à este múltiplo soma-se N/2 e obtém-se então a posição central do intervalo entre dois múltiplos adjacentes de N. Se este processo for aplicado em uma imagem de 8-bits, por exemplo, irá resultar em uma imagem com um total de 256/N x 256/N x 256/N classes de cores (LAGANIÈRE, 2014).

O algoritmo implementado para redução de cores também faz uso da biblioteca OpenCV e é apresentado no trecho de código-fonte abaixo. No laço de repetição da linha 9, para cada valor presente nos canais de cada pixel é aplicada a fórmula de redução. Caso a imagem seja contínua o laço é executado apenas uma vez.

Figura 5.7 – Operações morfológicas de abertura e fechamento em uma imagem do campo de futebol de robôs com valor de *kernel* igual a 5

```
void ColorProcessing::colorReduction(cv::Mat& image, int div) {
        int nl = image.rows; // number of lines
        int nc = image.cols * image.channels();
        if (image.isContinuous()) {
                nc = nc * nl;
                nl = 1; // it is now a 1D array
        }
        for (int j = 0; j < nl; j++) {
                uchar* data = image.ptr<uchar>(j);
                for (int i = 0; i < nc; i++) {
                        data[i] = data[i] / div * div + div / 2;
                }
        }
}
```

No sistema desenvolvido, decidiu-se utilizar um fator de redução de 70, o que resultou em um total de 64 classes de cores possíveis após a aplicação do filtro de redução.

Na Figura 5.9, é apresentada a imagem do robô no campo de futebol com suas cores reduzidas após a aplicação dos filtros morfológicos.

Com a aplicação dos filtros descritos até o momento, o nível de detalhamento retratado nas imagens resultantes mostrou-se satisfatório para que se desse continuidade nas etapas de processamento de imagem.

A seguir, as transições entre as classes de cores do *frame* analisado e as presentes na bússola são uma das características da imagem que serão analisadas para que se possa inferir a orientação do robô.

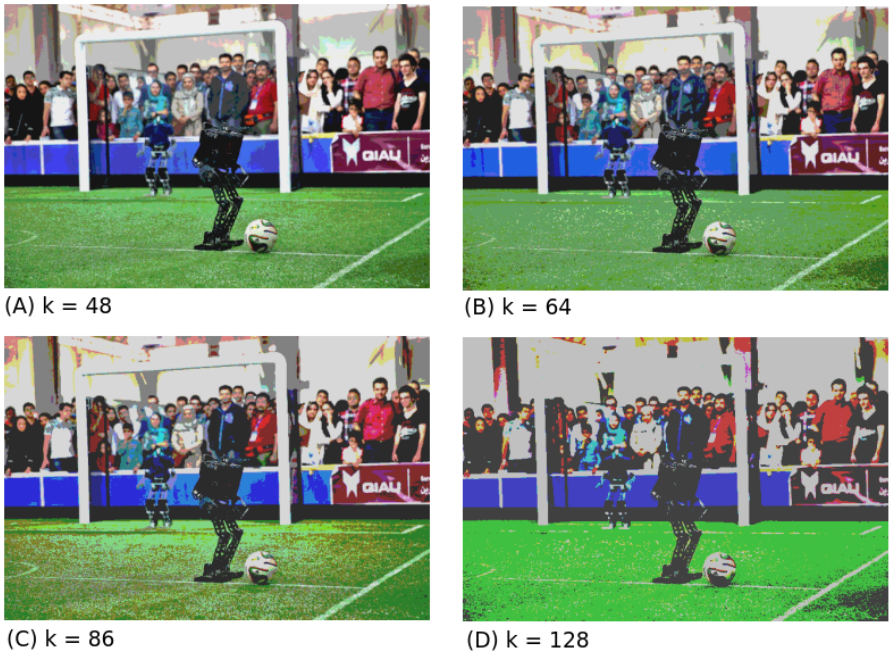


Figura 5.8 – Redução de cores na imagem original com diferentes fatores de redução

5.4.3 Espaços de cores

O espaço de cores comumente utilizado em sistemas computacionais e dispositivos de captura óptica é o RGB. Ele é baseado nas cores primárias vermelho (*Red*), verde (*Green*) e azul (*Blue*), das quais derivam uma ampla gama de cores a partir da combinação das três primárias.

Apesar de a maneira como ocorre a percepção de cores pela visão humana possuir semelhanças com a forma como as cores são representadas no modelo RGB, este não é o mais indicado para medir similaridade entre duas cores distintas, pois o RGB não é um espaço de cores com percepção uniforme. Ou seja, dada uma distância entre duas cores que parecem similares, outras duas cores separadas por esta mesma distância podem ser totalmente diferentes.

Portanto, para que o sistema desenvolvido pudesse realizar as comparações entre as classes de cores, foi necessário que se convertesse as imagens processadas para um espaço de cores mais adequado, o HSV.

Este padrão permite uma melhor distinção entre cores puras por trabalhar com tonalidade ou matiz (*Hue*), saturação (*Saturation*) e valor (*Value*, que equivale à luminosidade ou brilho da imagem). No espaço de cores HSV, os clusteres de cores da imagem são melhores separados, em uma representação mais próxima da percepção do olho humano. Dessa forma, ao realizar uma pequena alteração no valor do canal de cores, se obtém uma cor similar à original e não totalmente diferente, como pode ocorrer no espaço de cores RGB.

Figura 5.9 – Filtro de redução de cores aplicado sobre a imagem juntamente com as transformações morfológicas de abertura e fechamento

Na Figura 5.10, podem ser visualizados os canais de cores no espaço HSV, após a conversão da imagem original em RGB.

A rotina implementada para conversão entre os espaços de cores RGB e HSV é mostrada no trecho de código fonte abaixo. Utilizou-se métodos da biblioteca OpenCV para realizar as conversões.

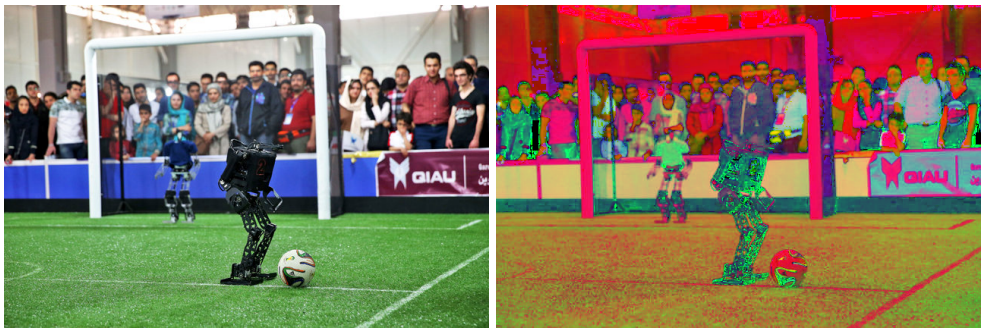
```
1 void ColorProcessing::bgrToHsv(Mat& image, Mat& output) {
2         cvtColor(image, output, COLOR_BGR2HSV);
3 }
```

```
1 void ColorProcessing::hsvToBgr(Mat& image, Mat& output) {
2         cvtColor(image, output, COLOR_HSV2BGR);
3 }
```

5.4.4 Histogramas

Mesmo com a aplicação do algoritmo de redução de cores e das técnicas de suavização, a imagem processada continua possuindo cores distintas nos pixels das áreas de interesse definidas. Torna-se necessário obter a classe de cor predominante na região analisada para armazená-la no modelo de dados.

A classe de cor predominante é obtida a partir da geração e análise de um histograma do fragmento da imagem analisado. Abaixo é apresentada a rotina para obtenção da cor predominante no fragmento de imagem já no espaço de cor HSV, através da análise dos canais Hue e



(a) Imagem no espaço de cor RGB (b) Imagem no espaço de cor HSV

(c) Canal H (Hue) (d) Canal S (Saturation) (e) Canal V (Value)

Figura 5.10 – Conversão do espaço de RGB para HSV

Saturation.

```
1
2  std::pair<Color, Color> Histogram::getMaxHueSat(Mat& hsv) {
3
4          int hbins = 256, sbins = 256;
5          int histSize[] = {hbins, sbins};
6          float hranges[] = { 0, 180 };
7          float sranges[] = { 0, 256 };
8          const float* ranges[] = { hranges, sranges };
9          MatND hist;
10         int channels[] = {0, 1};
11
12         calcHist( &hsv, 1, channels, Mat(),
13         hist, 2, histSize, ranges,      true, false );
14
15         double maxVal=0;
16         minMaxLoc(hist, 0, &maxVal, 0, 0);
17
18         Mat histImg = Mat::zeros(sbins*scale, hbins*scale, CV_8UC3);
19
20         std::pair<Color, Color> max_colors;
21         Color maxColor, secMaxColor;
22         int sec_maxVal = 0;
23         int count = 0;
24
25         for( int h = 0; h < hbins; h++ )
26                 for( int s = 0; s < sbins; s++ )
27                 {
```

```
28                          float binVal = hist.at<float>(h, s);
29                          count += binVal;
30                          int intensity = cvRound(binVal*100/maxVal);
31
32                          if(binVal == maxVal){
33                                  maxColor.channel1 = h;
34                                  maxColor.channel2 = s;
35                                  maxColor.freq = binVal;
36                                  maxColor.intensity = intensity;
37                                  max_colors.first = maxColor;
38                                  continue;
39                          }
40                          if(sec_maxVal < binVal && binVal != maxVal){
41                                  sec_maxVal = binVal;
42                                  secMaxColor.channel1 = h;
43                                  secMaxColor.channel2 = s;
44                                  secMaxColor.intensity = intensity;
45                                  secMaxColor.freq = binVal;
46                                  max_colors.second = secMaxColor;
47                          }
48
49                  }
50          max_colors.first.freq = max_colors.first.freq * 100/count;
51          max_colors.second.freq = max_colors.second.freq * 100/count;
52          return max_colors;
53  }
```

O método acima retorna um par de valores contendo as informações das duas cores mais frequentes na região de interesse da imagem analisada. Nas linhas 4 à 10 são definidos os parâmetros para geração do histograma. Como cada canal da imagem possui 8 bits e espera-se obter o valor real representado em cada canal, os valores do *bin* são definidos em 256 (de zero à 255) para ambos. Os intervalos de valor para o canal Hue varia de 0 à 180, pois é a notação adotada pela biblioteca OpenCV para representação destes valores, que representam a variação de cor da imagem, onde a cor presente no valor 0 é a mesma do valor 180. O valor contido no segundo canal varia entre 0 e 256 e representa o nível de branco da imagem, onde zero é totalmente branco, valores intermediários são tons de cinza e 255 é preto.

Nas linhas seguintes o histograma é gerado, com a função *calcHist*, e seu valor máximo é obtido através do método *minMaxLoc*, ambos da biblioteca OpenCV. E o resultado é analisado para que extraia-se os valores dos canais *Hue* e *Saturation* dos pixels mais recorrentes na imagem, bem como sua intensidade e frequência. Por fim, o método retorna este par de valores.

Na seção seguinte é apresentado o modelo de dados definido para o desenvolvimento da bússola visual.

5.5 Modelo de dados

O modelo de dados definido consiste em uma representação circular de informações extraídas de um conjunto de imagens capturadas pelo robô. Cada imagem, após ser processada, é segmentada em N colunas, chamadas de **Features**, de modo a seccionar verticalmente as informações. Cada Feature é ainda segmentada em um número pré-definido de **Classes de Cores**. Estes dados são carregados na fase de Calibração da bússola.

A quantidade de Features e Classes de Cores que o sistema gera é definida pelo utilizador nos parâmetros do sistema. É importante atentar-se à estes valores pois há uma implicação direta no resultado das comparações realizadas posteriormente para obter a orientação, uma vez que as classes de cores e suas transições são utilizadas para inferir a similaridade entre os dados da bússola e os *frames* capturados. Uma quantidade muito grande de Features pode gerar um ruído desnecessário ao calcular a orientação e uma quantidade pequena pode gerar uma análise imprecisa.

O sistema foi implementado seguindo uma abordagem orientada a objetos, na qual suas classes e relações foram definidas e podem ser visualizadas no Apêndice A.

Com base em testes realizados chegou-se a uma quantidade ideal de 128 Features para mapear o ambiente ao redor do robô. Utilizando uma câmera com campo de visão de 45 graus são necessárias 8 imagens de pontos distintos do ambiente para que se possa calibrar completamente a bússola. Um *frame* capturado possui 16 Features com 8 Classes de Cores cada, formando um total de 128 regiões de interesse por imagem.

Na Figura 5.11 é mostrada a combinação de dois *frames* do campo de treino para futebol de robôs da equipe TauraBots. Na Figura 5.12 a mesma imagem é mostrada segmentada, já processada e no espaço de cor HSV, com os pontos de interesse em destaque.

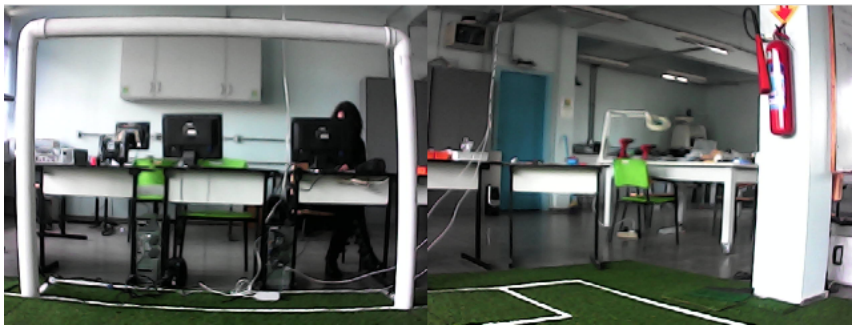


Figura 5.11 – Dois frames de imagem capturados do campo de jogo sem processamento

Os *frames* enviados para a bússola pelo robô devem conter as informações referentes

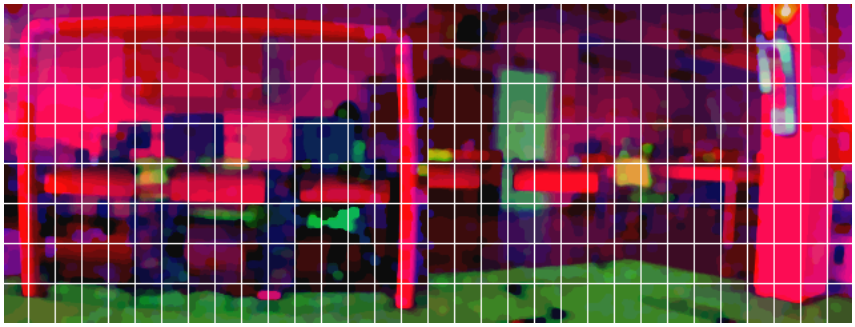


Figura 5.12 – Imagem no espaço de cor HSV segmentada

aos ângulos verticais e horizontais da câmera no momento da captura. Nos testes realizados, os conjuntos de imagens foram passados para a bússola visual com estas informações, para que na fase de calibração o sistema carregue o modelo de dados da bússola de acordo com a posição estimada da câmera.

5.6 Calibração

É chamada de calibração a fase de inicialização da bússola visual e a mesma deve ser realizada antes de a partida de futebol começar. Neste momento, o robô deve capturar *frames* de posições distintas do ambiente e passá-las à bússola junto com seu ângulo de rotação. A bússola por sua vez, processa cada uma destas imagens, identifica e extrai as Features nela contidas e atualiza o modelo de dados. O processamento da imagem consiste em aplicar as etapas descritas na seção 5.4 para que se obtenha as características de cores de cada imagem.

O código fonte da etapa de calibração da bússola pode ser visualizado no Apêndice B desta dissertação. Na seção seguinte é descrito o processo de detecção de Features da bússola visual projetada.

5.7 Detecção de Features

No modelo de dados definido no projeto da bússola visual desenvolvida neste trabalho, cada Feature é um conjunto de informações sobre as cores de um segmento vertical da imagem do ambiente. O conjunto de todas as Features do sistema, após sua calibração, constitui uma representação abstrata do panorama de 360º das cores ao redor do robô.

As informações sobre a classe de cor de cada região de interesse do frame são obtidas através da análise dos canais *Hue* e *Saturation*. Levando assim em consideração os aspectos

de nível de cor e quantidade de branco da imagem, respectivamente. Esta informação é obtida através da obtenção do histograma destes dois canais da imagem e análise da frequência com que estes valores aparecem. A classe de cor é então carregada com o valor mais frequente levando em conta a combinação destes dois canais analisados.

O sistema foi projetado para que também trabalhe com a análise de apenas um canal de cor (Hue). No capítulo 6 são discutidos os resultados da realização de experimentos com esta análise.

Com as classes de cores da imagem devidamente extraídas, temos para a bússola uma matriz de 8 linhas por 128 colunas, onde as linhas representam as Features e as colunas as classes de cor. E para cada imagem que vai ser comparada temos uma matriz de 8 classes de cor por 16 Features. Com estes dados em mãos é aplicada uma técnica de *Template Matching*, de modo a obter a comparação entre as matrizes de imagem em busca de áreas similares.

Para realizar a comparação é passado ao algorítimo de Template Matching (Apêndice C) o vetor de Features da bússola, que é a imagem onde as características serão buscadas, e o vetor de Features do frame capturado. Os dados do frame capturado são buscados dentro dos dados da bússola e é retornado um vetor contendo a informação de quão similar a imagem que está sendo comparada é com cada segmento da bússola em cada ponto da mesma. Ao realizar uma análise neste vetor resultante, é possível obter a posição da bússola onde o frame que está sendo compara melhor se encaixa e seu grau de certeza.

Dessa forma, ao realizar a conversão da matriz de Features x Cores para as dimensões originais, obtém-se a posição da bússola onde o frame melhor se encaixa e calcula-se então quantos graus à direita da posição inicial do robô a imagem está.

5.8 Considerações do capítulo

Neste capítulo foi apresentada e detalhada a metodologia proposta bem como os passos realizados em seu desenvolvimento. No capítulo seguinte são descritos os testes realizados com a calibração da bússola e detecção de Features.

6 RESULTADOS

Neste capítulo são apresentados os testes e experimentos realizados com a bússola visual desenvolvida ao longo deste trabalho.

6.1 Calibrando a bússola

Para validar o funcionamento da abordagem proposta, foram realizados testes, no campo de treino de futebol de robôs da equipe TauraBots, com imagens simulando o funcionamento do robô em uma partida. Inicialmente, *frames* são capturados juntamente com seu ângulo de rotação e são então utilizados na fase de calibração. Uma vez calibrada, a bússola está apta a estimar a orientação do robô com base em novas imagens capturadas.

A fase de calibração foi executada com um conjunto de 8 imagens, capturadas com uma câmera com 45º de abertura horizontal. Na Figura 6.1, estas imagens podem ser visualizadas juntamente com seus ângulos de rotação, também informados à bússola. Nota-se que há uma grande variação de luminosidade no ambiente, fato que contribui para a validação do método em cenários reais.

(a) 0º (b) 45º (c) 90º (d) 135º

(e) 180º (f) 225º (g) 270º (h) 315º

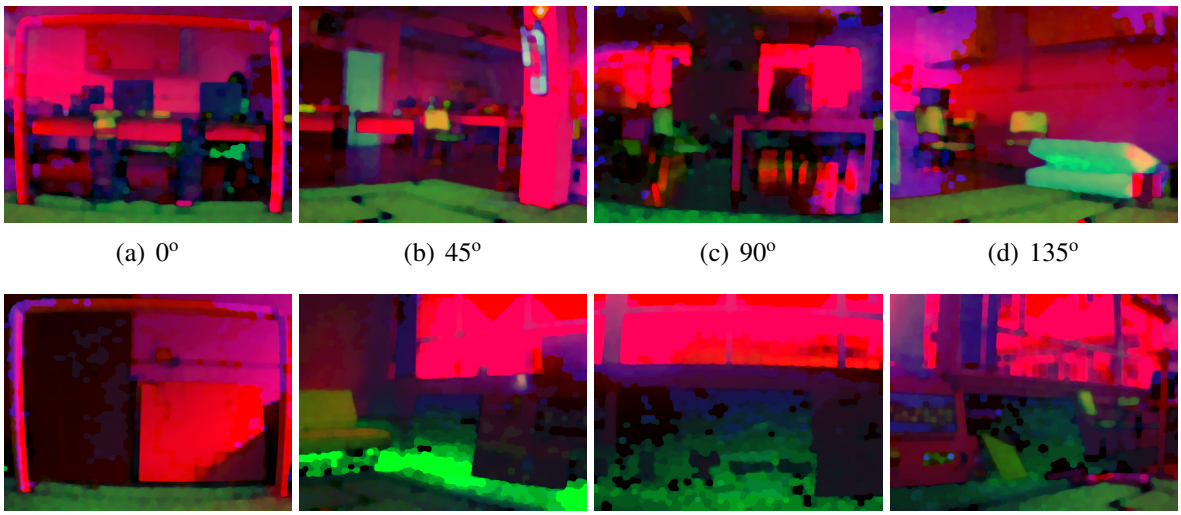
Figura 6.1 – Conjunto de imagens capturadas pelo robô na fase de calibração

Após realizar o carregamento das imagens, a bússola realiza o processamento das mesmas de modo a possibilitar a extração das Features que serão utilizadas para inferir a orientação. Este processo está descrito a seguir. Na Figura 6.2 está apresentada a visão panorâmica dos *fra-*

mes capturados pela câmera.

Figura 6.2 – Imagem RGB panorâmica sem processamento

Após receber cada frame, a bússola aplica as etapas de processamento de imagem definidas. Inicialmente é realizada a conversão do espaço de cor padrão (RGB) para o espaço de cor HSV, onde há uma percepção das cores da imagem mais próxima à percepção da visão humana. A seguir, são realizadas as etapas de alteração morfológica e redução de cores, de modo a reduzir a quantidade de ruídos presentes na leitura do ambiente realizado pela câmera e aprimorar a precisão na inferência da bússola. Na Figura 6.3 são mostradas as imagens originais após a conversão do espaço de cores e a aplicação dos filtros de abertura e de fechamento morfológico.



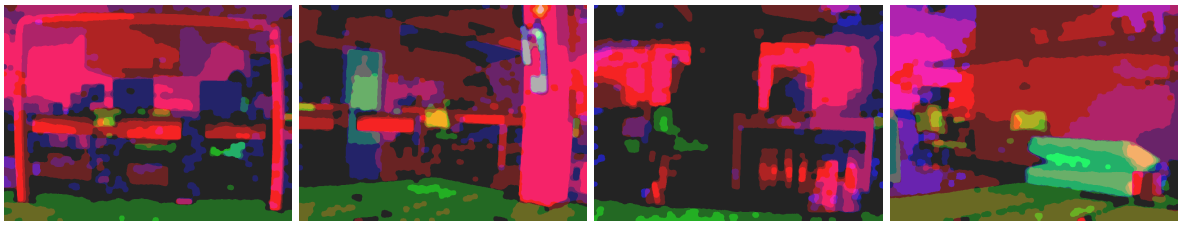
(a) 0º (b) 45º (c) 90º (d) 135º

(e) 180º (f) 225º (g) 270º (h) 315º

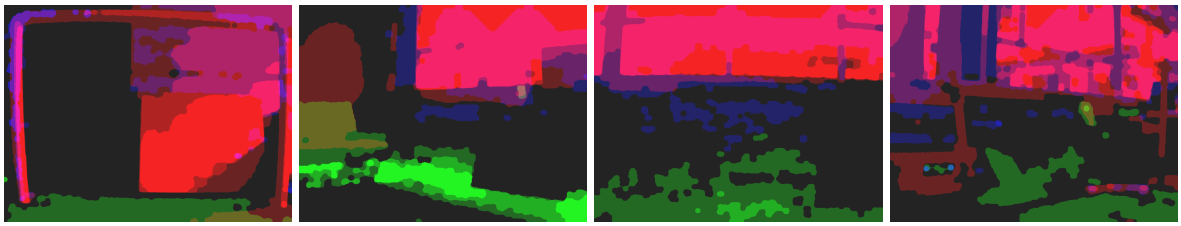
Figura 6.3 – Imagens em HSV após o processamento morfológico

Na etapa seguinte, onde é realizada a discretização das cores da imagem, é aplicada a técnica de redução de cores. Como descrito na Seção 3, o processo de discretização de cores reduz o valor de cada canal de cor da imagem para um grupo de clusteres reduzido, com base em um fator de redução. Neste teste foi utilizado um fator de redução de 48, de modo que cada canal de cor da imagem que possuía 256 valores possíveis (0-255), passa a possuir apenas 6 valores distintos. E a combinação dos três canais resulta em um total de 216 cores distintas, ao invés das 16 milhões iniciais. Mas como nesta abordagem serão analisados apenas os canais

Hue e *Saturation*, que representam o nível de cor e intensidade do branco, respectivamente, o total de clusteres de cor analisados pela bússola será de 36. As imagens resultantes deste processo podem ser visualizadas na Figura 6.4.



(a) 0º (b) 45º (c) 90º (d) 135º



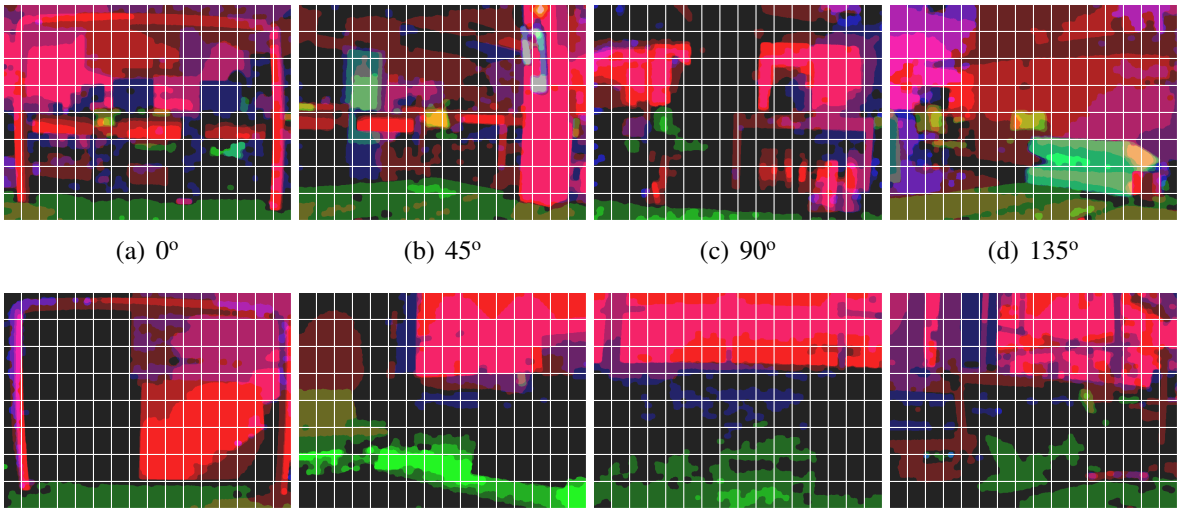
(e) 180º (f) 225º (g) 270º (h) 315º

Figura 6.4 – Imagens em HSV após o processamento morfológico e redução de cores

Vale ressaltar que o canal *Hue*, em sua definição teórica, possui valores entre 0 e 360. Entretanto, na representação de 8 bits da biblioteca OpenCV, valores entre 0 e 180 são atribuídos ao mesmo, de modo a ser possível armazená-lo nesta notação.

Após a conclusão destas etapas de processamento de imagem, os *frames* capturados são então tratados pela bússola de modo a identificar as regiões de interesse de cada cena. As informações extraídas de cada região de interesse são armazenadas em Classes de Cor e Features (colunas verticais). Cada Feature é abastecida com um total de oito Classes de Cor e de cada frame são extraídos um total de 16 Features, que representam os 45º de abertura da câmera presente no robô. Quando calibrada totalmente a bússola possui um total de 128 Features, representando a visão panorâmica de 360º do robô. Na Figura 6.5 é possível visualizar os *frames* processados pela bússola com suas regiões de interesse identificadas.

Em cada Classe de Cor (segmentos horizontais dentro da Feature), são armazenadas as informações de cor extraídas da região de interesse correspondente para, posteriormente, realizar a comparação com as novas imagens adquiridas. A principal informação guardada na Classe de Cor é a cor predominante daquela região de interesse, com base nos canais *Hue* e *Saturation*. A cor predominante é obtida a partir da análise dos histogramas destes dois canais da imagem. Os histogramas são equalizados e combinados pelo sistema para que então se



(a) 0º (b) 45º (c) 90º (d) 135º

(e) 180º (f) 225º (g) 270º (h) 315º

Figura 6.5 – Regiões de interesse realçadas nas imagens HSV após o processamento

obtenha a cor predominante da região, bem como sua frequência e intensidade.

Na Figura 6.6 são mostrados os canais de cor da imagem panorâmica de forma independente. Em (a) e (b) estão os canais *Hue* e *Saturation*, utilizados na obtenção da cores predominantes da Feature.

(a) Canal Hue

(b) Canal Saturation

(c) Canal Value

Figura 6.6 – Canais da imagem panorâmica no espaço de cores HSV

Ao final destas etapas descritas acima, o processo de calibração está concluído e a bússola está totalmente inicializada. O próximo passo agora é passar à ela novas imagens do ambiente e obter sua posição em relação à posição inicial do robô. Ou seja, quantos graus à direita o robô está posicionado em relação à sua posição de origem.

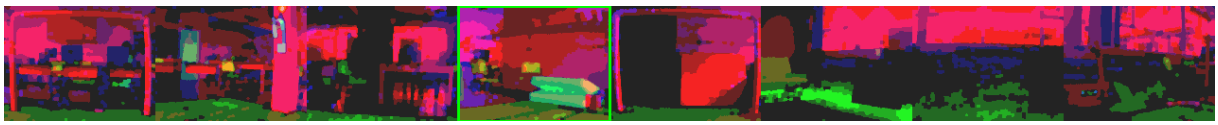
Para cada novo frame passado à bússola, as mesmas etapas de processamento de imagem

e extração de Features realizadas na fase de calibração, são novamente aplicados à esta nova imagem. Obtém-se assim as informações das regiões de interesse da nova imagem para que se realize a comparação e obtenha a orientação estimada.

As classes de cores extraídas da imagem a ser comparada são modeladas em uma matriz de FxC (Features x Classe de Cor) de dimensões 16x8 e são então confrontadas junto às informações presentes na matriz FxC da bússola (128x8) resultando na posição onde o frame analisado melhor se "encaixa"nos dados da bússola. Esta comparação é realizada com a utilização do algorítimo de Template Matching, o qual analisa similaridades de uma imagem contida em outra.

Para confirmar o funcionamento desta etapa (calibração), utilizou-se uma imagem capturada da posição inicial do robô para confrontá-la com a bússola calibrada. Nesta situação, esperava-se um resultado com precisão próxima de 100%, devido às características bastante semelhantes das imagens utilizadas para calibração e da imagem utilizada na comparação. Na Figura 6.7(a) têm-se o frame capturado e passado à bússola para obter a orientação do robô com base na mesma.

(a) Imagem passada à bússola para obter a orientação



(b) Desfecho do Feature Matching com realce na cor verde no resultado

Figura 6.7 – Resultado da obtenção da orientação do robô

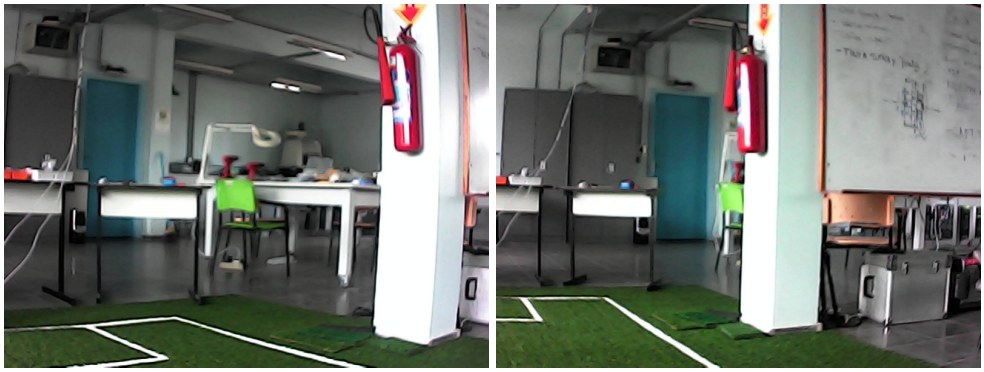
O algoritmo de Matching utilizado retorna uma matriz com a provável localização da imagem dentro da bússola. Esta matriz contém em cada célula o nível de similaridade da imagem que está sendo buscada dentro da bússola. Agora, varrendo essa matriz em busca da posição com o maior valor obtém-se a posição estimada do robô. Na Figura 6.7(b) é exibido o panorama da bússola com a realce em verde na posição encontrada após a aplicação do algo-

ritmo de Matching. A bússola retornou a posição 135º, referente aos graus à direita que o robô estaria posicionado quando capturou esta nova imagem em relação à sua posição inicial, como era esperado.

6.2 Obtendo a orientação dentro do campo

De modo a validar a metologia proposta, simulou-se também uma situação real de jogo onde o robô tem sua calibração realizada no centro do campo e posteriormente estima sua orientação de posições diferentes. Para os testes descritos nesta seção, assume-se que o robô está com sua bússola calibrada com o conjunto de *frames* descritos na seção anterior.

Duas imagens foram utilizadas neste teste, uma capturada com o robô no centro do campo (Figura 6.8a), posição inicial onde foi realizada a calibração, e outra com o robô movido para 1.3 metros à direita em direção à linha lateral (Figura 6.8b).



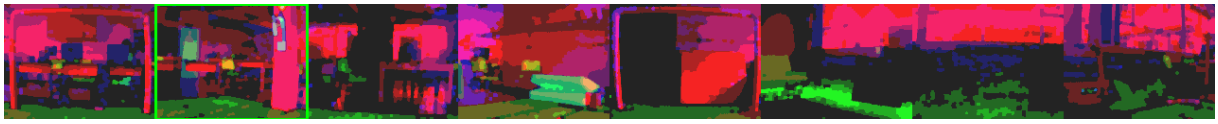
(a) Frame capturado com o robô no centro do campo

(b) Frame capturado com o robô posicionado a 1.3 metros à direita

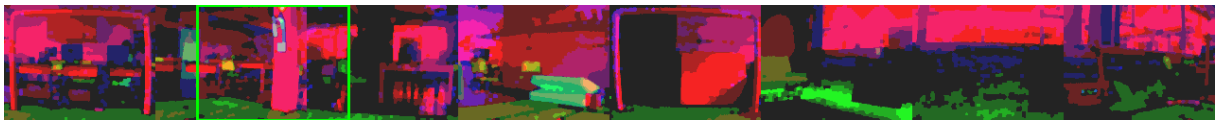
Figura 6.8 – Imagens utilizadas no teste de orientação do robô

A bússola, ao receber cada uma destas duas imagens, realizou as etapas de processamento descritas anteriormente para extração das Features. Nota-se que há uma diferença significativa de profundidade e luminosidade entre as cenas. Entretanto, como a caraterística desta abordagem é analisar as diferenças de cores dos elementos da imagem estes aspectos são minimizados ao realizar a análise de similaridade entre os dados dos *frames* e os da bússola.

Na Figura 6.9 temos o resultado do Matching destas destes dois *frames* com os dados da bússola, realçado com as linhas verdes. Em (a) a imagem utilizada para obter a orientação aufere uma orientação estimada de 45º (à direita em relação a posição zero do robô). E em (b), onde a imagem é capturada pelo robô em uma posição distante da sua posição de origem, obtém-se uma posição estimada de 57º.



(a) Resultado do Matching com a imagem capturada com o robô posicionado ao centro do campo



(b) Resultado do Matching com a imagem capturada com o robô posicionado à direita da posição inicial

Figura 6.9 – Resultado do segundo teste de obtenção da orientação do robô

A diferença de 12º apresentada nos resultados deste experimento deve-se à variação de perspectiva do robô em relação ao modelo de dados do ambiente presente na bússola nas imagens analisadas. Entretanto, esta diferença não mostrou-se significativa e foi possível inferir a orientação do robô de forma aceitável, levando-se em consideração as mudanças presentes no contexto entre um *frame* e outro.

6.3 Considerações do capítulo

Neste capítulo foram descritos alguns dos experimentos realizados com a bússola visual desenvolvida, no intuito de validar a abordagem apresentada neste trabalho. Além dos testes apresentados neste capítulo, realizou-se uma série de outros experimentos com valores e parâmetros diferentes nas etapas de processamento de imagem e análise de similaridade até se chegar na abordagem proposta.

Com base nos resultados dos experimentos realizados e em comparação com os resultados apresentados por outras abordagens presentes na literatura, considerou-se boa a eficácia e a precisão apresentada pela metodologia proposta. Pois, apesar das mudanças e variações tanto do robô quanto do ambiente, obteve-se uma orientação inferida próxima da esperada.

A proposta possui algumas limitações como a forte dependência da etapa de calibração onde variações bruscas nos elementos do cenário nesta fase podem acarretar em um funcionamento inapropriado da bússola. Exemplos de variações seriam mudanças ou transições em placas de anúncios ou patrocinadores e a movimentação de pessoas ao fundo.

Outra limitação conhecida é a dependência dos valores de ângulos de rotação e posição da cabeça do robô no momento da captura dos *frames*, pois em um cenário real estes valores podem possuir variações consideráveis devido às imperfeições do solo e movimentação do robô.

Entretanto, entende-se a abordagem apresentada e sua combinação de técnicas de visão

computacional como a principal colaboração deste trabalho. Pois, reduziu-se a complexidade das características extraídas das imagens a ponto de possibilitar uma comparação com resultados satisfatórios.

No capítulo seguinte são apresentadas as considerações finais do trabalho.

7 CONCLUSÃO

Neste trabalho foi apresentada uma abordagem de bússola visual, puramente baseada em informações visuais e cores, para estimar a orientação de robôs autônomos. O contexto abordado ao longo do projeto foi o de futebol de robôs humanóides, os quais possuem grande dificuldade na tarefa de obter sua localização de forma precisa e eficiente. Esta dificuldade em se orientar e/ou localizar em um ambiente desconhecido não é exclusividade destes tipos de robôs, é um problema recorrente em outros segmentos da robótica. Dessa forma, a proposta de bússola visual apresentada para estimar a orientação de robôs humanóides autônomos jogadores de futebol pode ser facilmente adaptada para outros segmentos da robótica.

Os experimentos realizados demonstraram que a utilização dos canais de nível de cor e intensidade de branco para se identificar as cores predominantes e utilizá-las para obter a orientação resultaram em um grau aceitável de erros e acertos ao realizar o *Matching* das Features. Variações de luminosidade também não se mostraram um problema para a abordagem proposta.

Entretanto, exitem pontos a serem melhorados no sistema desenvolvido, como melhorar o desempenho das rotinas aplicadas, de modo a tornar viável sua utilização em *hardwares* com pouco poder processamento, o que é comum em muitos projetos de robôs de baixo a médio custo atualmente.

Devido à enormidade de combinações de técnicas de processamento de imagens que podem ser aplicadas em sistemas de visão computacional deste tipo, o projeto do sistema desenvolvido preocupou-se em encapsular as rotinas de processamento de modo a torná-lo flexível e facilitar a integração de novo algorítimos, estimulando assim a continuidade da pesquisa, seja no âmbito do futebol de robôs ou em outras aplicações.

Identifica-se como trabalho futuro a extensão desta proposta para uma abordagem onde seja utilizado um conjunto de bússolas visuais para estimar a orientação. Dessa forma acredita-se que a precisão ao inferir a orientação seria aumentada de forma significativa.

Outra possível continuidade que pode ser dada ao trabalho de pesquisa é a calibração constante dos dados da bússola visual. Ao passo que o robô se movimenta e recebe novas informações visuais, o modelo de dados vai sendo atualizado e melhorando a qualidade de suas informações. Esta funcionalidade pode ser aplicada tanto à bússola na forma como foi apresentada neste trabalho quanto no conjunto de bússolas sugerido no parágrafo anterior.

Por fim, como os experimentos realizados com as imagens do robô foram feitos de forma

simulada, a integração da bússola desenvolvida ao sistema que opera no robô também seria de grande valia para a proposta. Dessa forma, poderia-se analisar o funcionamento do sistema de forma mais precisa e, então, obter resultados reais.

REFERÊNCIAS

ALBUQUERQUE, M. P. de; ALBUQUERQUE, M. P. de. Processamento de imagens: métodos e análises. **Centro Brasileiro de Pesquisas Fısicas MCT**, [S.l.], 2000.

ALTEROVITZ, R.; KOENIG, S.; LIKHACHEV, M. Robot planning in the real world: research challenges and opportunities. **National Science Foundation**, [S.l.], 2014.

ANDERS, B. et al. WF Wolves & Taura Bots–Humanoid Teen Size Team Description for RoboCup. , [S.l.], 2016.

ANDERSON, P.; HENGST, B. Fast monocular visual compass for a computationally limited robot. In: ROBOT SOCCER WORLD CUP. **Anais...** [S.l.: s.n.], 2013. p.244–255.

BALLARD, D. H.; BROWN, C. M. **Computer Vision**. 1st.ed. [S.l.]: Prentice Hall Professional Technical Reference, 1982.

BAYA, H. et al. Speeded-up robust features (SURF). **Computer Vision and Image Understanding**, [S.l.], v.110, n.3, p.346–359, 2008.

BEKEY, G. A. **Autonomous robots**: from biological inspiration to implementation and control. [S.l.]: MIT press, 2005.

BENDER, T. C. Classificaçao e recuperaçao de imagens por cor utilizando técnicas de inteligência artificial. , [S.l.], 2003.

BURKE, A.; VARDY, A. Visual compass methods for robot navigation. In: NEWFOUNDLAND CONFERENCE ON ELECTRICAL AND COMPUTER ENGINEERING. **Proceedings...** [S.l.: s.n.], 2006.

CONCI, A.; AZEVEDO, E.; LETA, F. R. **Computação Gráfica-Vol. 2**. [S.l.]: Rio de Janeiro: Elsevier, 2007.

FOX, D. et al. Monte carlo localization: efficient position estimation for mobile robots. **AAAI/IAAI**, [S.l.], v.1999, p.343–349, 1999.

FOX, D. et al. Bayesian filtering for location estimation. **IEEE pervasive computing**, [S.l.], v.2, n.3, p.24–33, 2003.

GIACHETTI, A.; CAMPANI, M.; TORRE, V. The use of optical flow for road navigation. **Robotics and Automation, IEEE Transactions on**, [S.l.], v.14, n.1, p.34–48, Feb 1998.

GONZALEZ, R. C.; WOODS, R. E. Digital image processing. **Nueva Jersey**, [S.l.], 2008.

GUDI A., d. K. P. M. G. S. N. **Feature Detection and Localization for the RoboCup Soccer SPL. Project report, Universiteit van Amsterdam**. 2013.

HA, I. et al. Development of open humanoid platform DARwIn-OP. In: SICE ANNUAL CONFERENCE (SICE), 2011 PROCEEDINGS OF. **Anais...** [S.l.: s.n.], 2011. p.2178–2181.

KARLSSON, N. et al. The vSLAM algorithm for robust localization and mapping. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 2005. **Proceedings...** [S.l.: s.n.], 2005. p.24–29.

KOK; GEORGIOS K METHENITIS, A. G. P. de; STEENBERGEN, N. **Feature detection and localization for the RoboCup Soccer SPL**. 2013.

KOK, P. M. de; METHENITIS, G.; NUGTEREN, S. ViCTOriA: visual compass to orientate accurately. , [S.l.], 2013.

LABROSSE, F. Visual Compass. In: UNIVERSITY OF ESSEX, COLCHESTER, UNITED KINGDOM. **Anais...** [S.l.: s.n.], 2006. p.85–92.

LAGANIÈRE, R. **OpenCV Computer Vision Application Programming Cookbook Second Edition**. [S.l.]: Packt Publishing Ltd, 2014.

LULIO, L. C. **Técnicas de visão computacional aplicadas ao reconhecimento de cenas naturais e locomoção autônoma em robôs agrícolas móveis**. 2011.

MARENGONI, M.; STRINGHINI, S. Introdução à visão computacional usando opencv. **Revista de Informática Teórica e Aplicada**, [S.l.], v.16, n.1, p.125–160, 2009.

MARQUES FILHO, O.; NETO, H. V. **Processamento digital de imagens**. [S.l.]: Brasport, 1999.

MARTINS-FILHO, L. S. et al. Kinematic control of mobile robots to produce chaotic trajectories. In: INT. CONGRESS OF MECHANICAL ENGINEERING, OURO PRETO, 18. **Proceedings...** [S.l.: s.n.], 2005.

METHENITIS, G. et al. Orientation finding using a grid based visual compass. In: BNAIC 2013: PROCEEDINGS OF THE 25TH BENELUX CONFERENCE ON ARTIFICIAL INTELLIGENCE, DELFT, THE NETHERLANDS, NOVEMBER 7-8, 2013. **Anais...** [S.l.: s.n.], 2013.

MIKOLOV, T. š. Color Reduction Using K-Means Clustering. , [S.l.], 2008.

MIRANDA, J. I. Processamento de Imagens Digitais: pratica usando javatm. , [S.l.], 2006.

OPENCV. **Morphology Transformation**. [Online; accessado em 12-03-2016], http://docs.opencv.org/2.4/doc/tutorials/imgproc/opening_closing_hats/opening_closing_hats.html.

QUEIROZ, J. E. R. de; GOMES, H. M. Introdução ao Processamento Digital de Imagens. **RITA**, [S.l.], v.13, n.2, p.11–42, 2006.

ROBOCUP. **Objetivo da RoboCup Federation**. [Online; accessado em 19-09-2015], http://www.robocup.org.br/objetivo.php.

ROBOCUP. **Areas de pesquisa da RoboCup**. [Online; accessado em 19-09-2015], http://www.robocup.org.br/pesquisa.php.

SCURI, A. E. Fundamentos da imagem digital. **Pontifícia Universidade Católica do Rio de Janeiro**, [S.l.], 1999.

SOUZA, C. F.; CAPOVILLA, F.; ELEOTERIO, T. C. Visão Computacional. **Faculdade de Tecnologia de Taquaritinga, Centro Estadual De Educação Tecnológica**, [S.l.], 2010.

STURM, J.; VISSER, A. An appearance-based visual compass for mobile robots. **Robotics and Autonomous Systems**, [S.l.], v.57, n.5, p.536–545, May 2009.

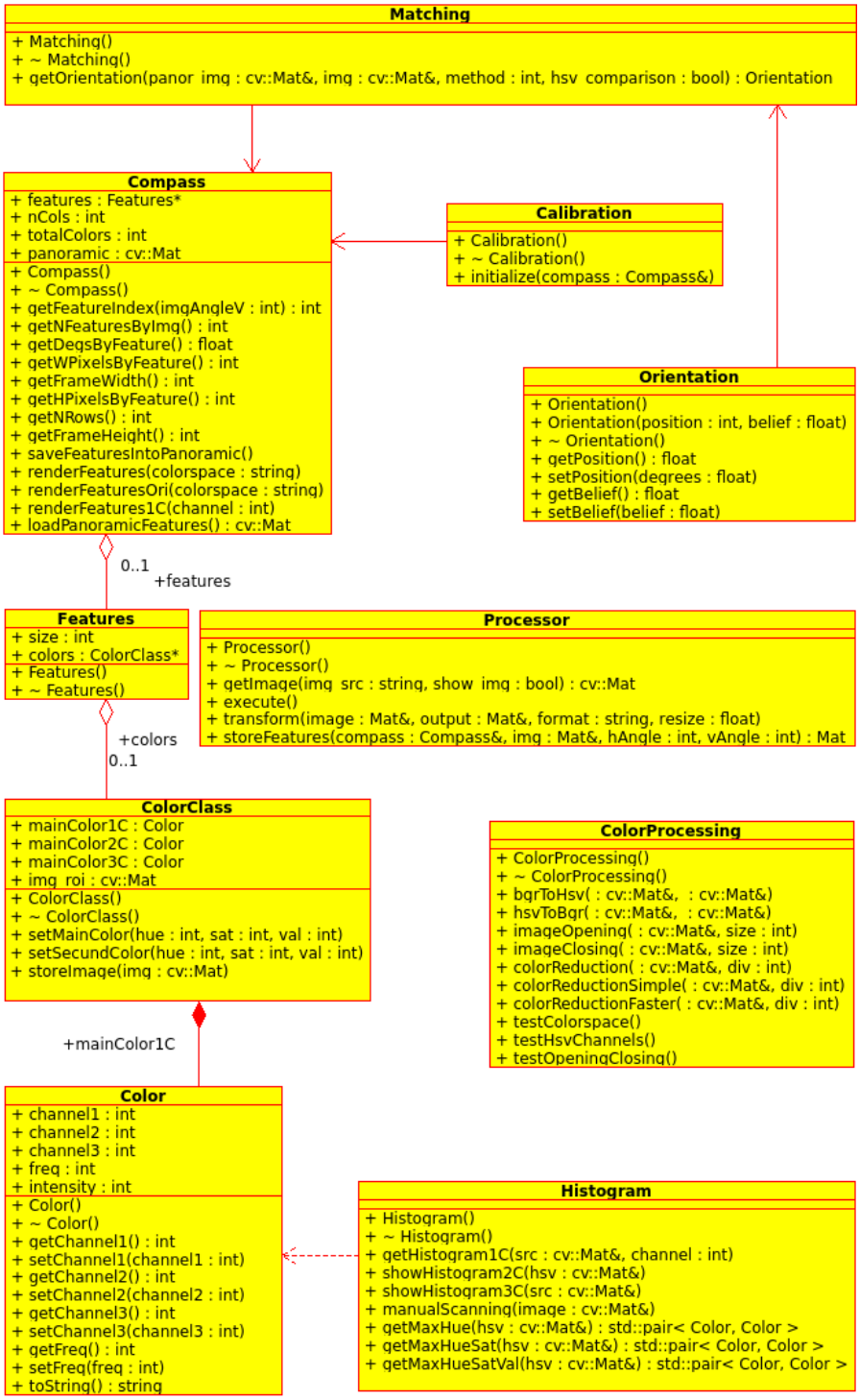
VELHO, L.; GOMES, J. Computação gráfica: imagem. **Sociedade Brasileira de Matemática. Rio de Janeiro**, [S.l.], 1994.

WELCH, G.; BISHOP, G. An introduction to the kalman filter. **Proceedings of the Siggraph Course, Los Angeles**, [S.l.], 2001.

WOLF, D. F. et al. Robótica móvel inteligente: da simulação às aplicações no mundo real. In: MINI-CURSO: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA (JAI), CONGRESSO DA SBC. **Anais...** [S.l.: s.n.], 2009. p.13.

APÊNDICES

APÊNDICE A – Diagrama de classes da Bússola Visual



APÊNDICE B – Métodos do para calibração da bússola

Calibration.cpp

```
1
2  void Calibration::initialize(Compass& compass){
3
4          string loc = "/location/to/images/",
5                  ext = ".png";
6
7          string images[8][3] = {
8          //name,hAngle,vAngle
9          {"01", "0"  , "0"},
10         {"02", "45" , "0"},
11         {"03", "90" , "0"},
12         {"04", "135", "0"},
13         {"05", "180", "0"},
14         {"06", "225", "0"},
15         {"07", "270", "0"},
16         {"08", "315", "0"}
17         };
18
19
20         for (int i=0; i < 8; i++){
21
22                 string img_loc = loc + images[i][0] + ext;
23
24                 //load img
25                 cv::Mat original_img = cv::imread(img_loc),
26                 image;
27
28                 int hAngle = atoi(images[i][1].c_str()),
29                 vAngle = atoi(images[i][2].c_str());
30
31
32                 Processor * improc;
33                 image = improc->storeImageFeatures(compass,
                      original_img, hAngle, vAngle, true);
34
35         }
36 }
```

Processor.cpp

```
1  Mat Processor::storeImageFeatures(Compass& compass, Mat& src_image,
       int hAngle, int vAngle, bool update_compass){
2
3          Mat image;
4          this->transform(src_image, image, "hsv");
5
6          Histogram hist;
```

```
7
8           /* Load the feature index to be inserted
9           * */
10          int fStart = compass.getFeatureIndex(hAngle),
11          fTotal= compass.getNFeaturesByImg(),
12          fEnd = fTotal + fStart,
13          // Load the pixels count of each feature
14          fWPx = compass.getWPixelsByFeature(),
15          fHPx = compass.getHPixelsByFeature(),
16          wPx = compass.getFrameWidth(),
17          hPx = compass.getFrameHeight(),
18          nRows = compass.getNRows();
19
20          int vscale = 1, hscale=1;
21          Mat return_image = Mat(Params::nRows * hscale, (Params::nCols
               /(360/Params::camHOpening)) * vscale, CV_8UC1);
22          return_image.setTo(-1);
23          int ret_col = 0, ret_row = 0;
24
25          int col = 0;
26          // For each feature (to be inserted on the compass)
27          for (int f = fStart; f < fEnd; f++){
28
29                  // goes through the pixels of the feature
30                  for (int k = col * fWPx; k < col*fWPx+fWPx; k+=fWPx){
31
32                  if(k+fWPx <= image.cols){
33                  for (int row = 0; row < nRows; row++){
34
35                          int x1 = row*fHPx,
36                          y1 = row*fHPx+fHPx-1,
37                          x2 = k,
38                          y2 = k+fWPx-1;
39
40
41                          // Extract de Region of Interest to be
                               analyzed
42                          cv::Mat imageROI= image(cv::Range(x1,y1), cv
                               ::Range(x2,y2));
43                          cv::Mat imageROI_ori= src_image(cv::Range(x1,
                               y1), cv::Range(x2,y2));
44
45                          std::pair<Color, Color> max_colors1C = hist.
                               getMaxHue(imageROI);
46                          std::pair<Color, Color> max_colors2C = hist.
                               getMaxHueSat(imageROI);
47
48
49                          if(update_compass){
50                                  compass.features[f].colors[row].
                                       storeImage(imageROI,imageROI_ori,
```

```
                                    x1, y1, x2, y2);
51
52                                  compass.features[f].colors[row].
                                       mainColor1C = max_colors1C.first;
53                                  compass.features[f].colors[row].
                                       secColor1C = max_colors1C.second;
54
55                                  compass.features[f].colors[row].
                                       mainColor2C = max_colors2C.first;
56                                  compass.features[f].colors[row].
                                       secColor2C = max_colors2C.second;
57                          }
58
59
60
61                          for (int j=(f - fStart)* hscale; j <= (f-
                               fStart) * hscale + hscale - 1; j++ )
62                                  for (int i=row * vscale; i <= row *
                                       vscale +vscale - 1; i++ ){
63                                          return_image.at<uchar>(i,j) =
                                               max_colors1C.first.
                                              channel1;
64                                          }
65
66                                          ret_row++;
67                                  }
68                          }
69                  }
70                  ret_col++;
71                  col++;
72          }
73          return return_image;
74  }
```

APÊNDICE C – Feature matching

```
1
2  Orientation Matching::getOrientation(Mat& panor_img, cv::Mat& img,
       int method, bool hsv_comparison){
3
4          this->hsv_comparison = hsv_comparison;
5
6          /*
7          * Template MAtching
8          *                0: SQDIFF
9          *                1: SQDIFF NORMED
10         *                2: TM CCORR
11         *                3: TM CCORR NORMED
12         *                4: TM COEFF
13         *                5: TM COEFF NORMED
14         *
15         * */
16
17         switch (method) {
18         case 0:
19         case 1:
20         case 2:
21         case 3:
22         case 4:
23         case 5:
24                 return this->templateMatching(panor_img, img, method)
                       ;
25                 break;
26         default:
27
28                 return this->templateMatching(panor_img, img,
                       CV_TM_CCOEFF);
29                 break;
30         }
31
32 }
33
34
35 Orientation Matching::templateMatching(Mat& panor, cv::Mat img, int
       match_method){
36
37         /// Create the result matrix
38         int result_cols =  panor.cols - img.cols + 1;
39         int result_rows = panor.rows - img.rows + 1;
40
41         Mat result( result_cols, result_rows, CV_8UC1 );
42
43         /// Do the Matching and Normalize
44         matchTemplate(panor, img, result, match_method);
```

```
45          normalize(result, result, 0, 1, NORM_MINMAX, -1, Mat());
46
47          double minVal; double maxVal; Point minLoc; Point maxLoc;
48          Point matchLoc;
49          minMaxLoc( result, &minVal, &maxVal, &minLoc, &maxLoc, Mat()
               );
50
51          if( match_method  == TM_SQDIFF || match_method ==
               TM_SQDIFF_NORMED ){
52          matchLoc = minLoc;
53          }
54          else{
55          matchLoc = maxLoc;
56          }
57
58          rectangle(panor, matchLoc, Point( matchLoc.x + img.cols ,
               matchLoc.y + img.rows ), Scalar(0, 255, 0), 2);
59          rectangle(result, matchLoc, Point( matchLoc.x + img.cols ,
               matchLoc.y + img.rows ), Scalar(0, 255, 0), 2);
60
61          //  ImageView viewer;
62          //  imshow("panor",panor);
63          //  imshow("result",result);
64          //  viewer.renderScaledImage(panor, "panor", 10, 10);
65          //  viewer.renderScaledImage(result, "result", 10, 10);
66
67          float degree;
68          if(this->hsv_comparison)
69          degree =floor((matchLoc.x * 360 )/(Params::pxWidth*360/Params
               ::camVOpening * Params::panorStoreScale));
70          else
71          degree = floor(matchLoc.x *(360.0 / (float)Params::nCols));
72
73          Orientation o;
74          o.setPosition(degree);
75
76          return o;
77  }
```