

Trabalho de Conclusão de Curso

Desenvolvimento de um Sistema de
Calibração de Cores com
Reconhecimento de Objetos para a
Equipe de Futebol De Robôs Cedro
Categoria IEEE Very Small Size

Jasane Schio

Orientação: Prof. Dr. Gedson Faria

Coorientação: Prof. Me. Angelo Darcy

Área de Concentração: Sistemas de Informação, Visão Computacional

Sistema de Informação
Universidade Federal de Mato Grosso do Sul
30 de Setembro de 2015

Desenvolvimento de um Sistema de
Calibração de Cores com
Reconhecimento de Objetos para a
Equipe de Futebol De Robôs Cedro
Categoria IEEE Very Small Size

Coxim, 01 de Outubro de 2015.

Banca Examinadora:

- Prof. Me. Angelo Darcy (CPCX/UFMS)
- Prof. Dr. Gedson Faria (CPCX/UFMS) - Orientador

Conteúdo

Lista de Figuras	5
1 Introdução	7
1.1 Contexto	7
1.2 Justificativa e Objetivos	7
1.3 Trabalhos Corelatos	8
1.3.1 Calibra	8
1.3.2 VSS-Vision	8
1.3.3 ??	9
1.4 Organização da Trabalho	10
2 Fundamentação Teórica	12
2.1 Processamento de Imagens	12
2.1.1 Detecção de Objetos	12
2.1.2 Detecção de Bordas	12
2.2 Cores	14
2.3 Futebol de Robôs	16
3 Desenvolvimento	17
3.1 Tecnologias Usadas	17
3.2 Projeto	18
3.2.1 Organização do Projeto	18
3.2.2 Classes	18
3.3 O Sistema	20
3.3.1 1 ^a Etapa - Configuração de Camera	20
3.3.2 2 ^a Etapa - Reconhecimento de Fundo	22

3.3.3	3 ^a Etapa - Extração dos Objetos do Fundo	23
3.3.4	4 ^a Etapa - Detecção e validação de Objetos	23
3.3.5	5 ^a Etapa - Classificação do Pixel	24
3.3.6	6 ^a Etapa - Gerar Arquivo de Cores	25
4	Testes e Resultados	26
4.1	Calibração	26
4.2	Testes	28
4.2.1	Cores Comuns	29
4.2.2	Cores Com Problemas Conhecidos	33
Referências Bibliográficas		37
Apêndices		38

Lista de Figuras

1.1	Sistema Calibra desenvolvido pelo Centro Universitário da FEI (1)	8
1.2	Sistema de calibracao desenvolvido peloSIRLab (2)	9
1.3	Sistema de calibracao desenvolvido peloSIRLab (3)	9
1.4	Nova interface do time da SIRLab (3)	10
1.5	Calibração atual do time da SIRLab (3) após calibração	10
2.1	Detecçao de Borda com Algoritmo de Canny (4)	13
2.2	Universo de cores estabelecido pelo CIE.	14
2.3	Exemplo dos espaços de cores RGB E CMY.	15
2.4	Exemplo do Modelo de Cor RGB. Horvath(5)	15
2.5	Exemplo do Modelo de Cor HSV, Horvath(5)	16
3.1	Organizaçao das pastas do projeto	18
3.2	Diagrama de Fluxo	20
3.3	Configuração de Camera	21
3.4	Configuração de Camera	22
3.5	Geraçao de Mascara	23
4.1	Imagen do fundo com a seleção de campo	26
4.2	Objetos dispostos no campo para calibração	27
4.3	Divisao do campo em quinze partes nomeadas alfabeticamente.	28
4.4	Disposiçao de cada parte quanto as cores	28
4.5	Campo apos terem sido dispostas as cores	28
4.6	Imagen somente com os objetos dentro do intervalo do valor da cor amarela	29
4.7	Imagen somente com os objetos dentro do intervalo do valor da cor azul . .	30
4.8	Imagen somente com os objetos dentro do intervalo do valor da cor verde . .	31

Capítulo 1

Introdução

1.1 Contexto

1.2 Justificativa e Objetivos

Este trabalho tem por objetivo principal automatizar o sistema de identificação de objetos coloridos em imagens provenientes de uma câmera em imagens de tempo real, fazendo a calibração dos valores HSV definindo seus limites mínimos e máximos. Se fez necessário um sistema para tal finalidade notando-se:

- A falta, na equipe, de um sistema de fácil manuseio para detecção de valores HSV
- A falta, na equipe, de um sistema automático de registro do valores HSV mínimos e máximos
- A falta, na equipe, de um sistema que defina mínimos e máximos de forma automática, baseando-se nos objetos escolhidos
- A falta, na equipe, de um sistema autônomo de calibração de intervalo de cores
- Aplicação do sistema proposto na identificação de robôs moveis em times de futebol de robôs.

Para alcançar o objetivo principal, foram propostos os seguintes objetivos específicos.

- Implementar uma interface que conte com disposição de informações no estilo gráfico ou histograma de cores para um corte manual de valores visando diminuição da velocidade de detecção;
- Estudo e implementação de um sistema inteligente de calibração de cores e no corte inteligente de valores minimo e máximo das cores
- Testar o sistema proposto para identificação de equipes e participantes do futebol de robô na categoria Very Small Size.

1.3 Trabalhos Corelatos

Para o tema específico deste trabalho, calibração de intervalo de cores para times de futebol de robôs da categoria very small size, não foram encontrados trabalhos relacionados, porém foram encontrados Team Discription Papers e descrições de sistemas usados pelos times, onde consta sobre o processo de calibração e os métodos usados.

1.3.1 Calibra

O Centro Universitário da FEI, como publicado em artigo do I Encontro de Robótica Inteligente(6), utiliza em sua equipe Y04 um sistema denominado CALIBRA(1). Desenvolvido para sistemas Linux e com Graphical User Interface(1), o sistema de calibração possui um módulo chamado de MainWindow, que é responsável pela configuração de brilho, cor e contraste da imagem adquirida pela câmera e gera um arquivo que é analizado na hora da criação das cores padrão(6), onde cores-padrão são definidas como intervalos no espaço de cores HSI(6).



Figura 1.1: Sistema Calibra desenvolvido pelo Centro Universitário da FEI (1)

1.3.2 VSS-Vision

Em 2015 Rosa(2) descreveu em seu Trabalho de Conclusão sobre a equipe de futebol de robôs Very Small Size, do Laboratório de Sistemas Inteligentes e Robótica, SIRLab(Faeterj-Petrópolis), o sistema de visão computacional da equipe, durante a competição do ano de 2014, que abrange inclusive a parte de calibração.

O autor menciona que a calibração de cores é feita calibrando obrigatoriamente laranja, amarelo e azul, e então as outras cores referentes aos jogadores em campo. Como visto na Figura 1.2 a imagem da câmera é dividida em nove cantos, e para calibrar a cor o usuário deve clicar em cima da cor que gostaria de ser calibrada, assim salvando um intervalo de cor tratado como RGB máximo e o mínimo daquela cor , a medida que vão havendo os cliques o sistema verifica para cada atributo se ele é maior que o atributo máximo salvo ou menor que mínimo salvo, caso seja, o mesmo assume o lugar de menor ou maior(2) e esse processo

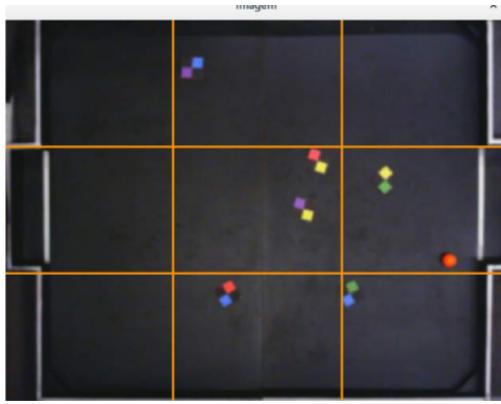


Figura 1.2: Sistema de calibracao desenvolvido peloSIRLab (2)

deve ser feito em cada um dos nove cantos da imagem. Os valores HSV encontrados são ajustados manualmente com a ajuda de sliders, como visto na Figura 1.3. Este processo de calibração pode demorar entre cinco e dez minutos. O desenvolvimento do sistema utiliza para processamento de imagens a biblioteca OpenCV e para telas interativas a biblioteca ImGui.

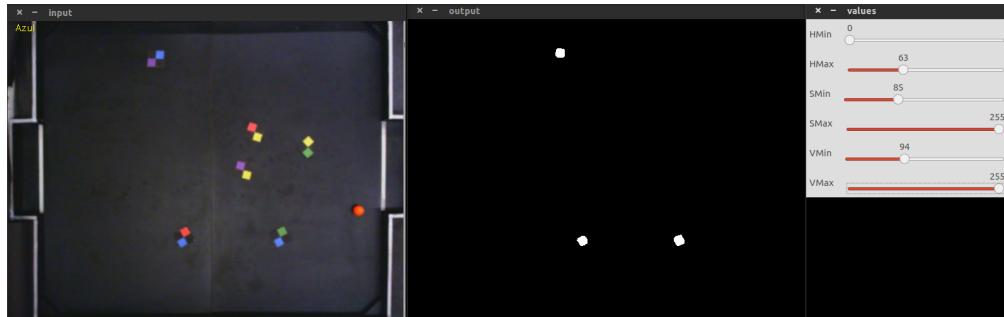


Figura 1.3: Sistema de calibracao desenvolvido peloSIRLab (3)

O atual sistema de visão computacional do SIRLab passou por algumas mudanças desde 2015 e conta com uma interface e método de calibração diferentes(3). Como disponível no repositório online do Laboratório, o atual sistema de calibração de cores utiliza o espaço de cores HSV, no lugar do RGB(2). A antiga interface do sistema, feita inicialmente em ImGui deu lugar a nova, desenvolvida em Qt, como mostra a Figura 1.4.

O método de calibração de cores também foi modificado, segundo a equipe(3) o sistema possibilita a calibragem de 8 cores, Laranja, Amarelo, Azul, Vermelho, Verde, Rosa, Roxo, Marrom. Após o usuário escolher uma cor para calibrar o mesmo deve encontrar um intervalo de cor, no espaço de cores HSV, que represente-a. Ao clicar na tela com o botão direito o sistema faz um zoom na área para ajuste fino. A Figura 1.5 demonstra o novo método de calibração.

1.3.3 ??

(??) em artigo do IX Simposio Brasileiro de Inteligencia Artificial descreve uma abordagem de detecção em tempo real utilizando redes neurais MLP, Perceptron de Multiplas Camadas. Como a utilização de somente redes neurais podem causar um grande custo compu-

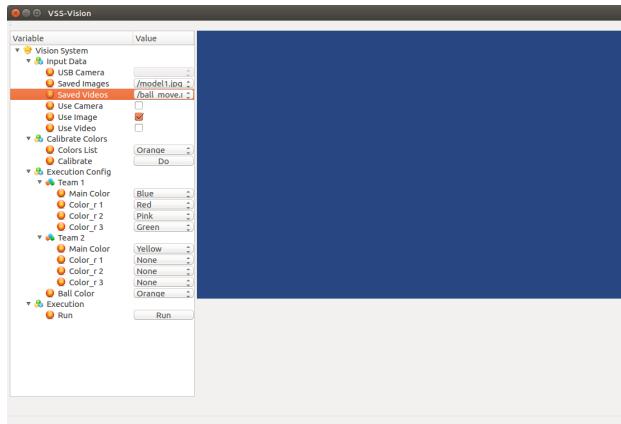


Figura 1.4: Nova interface do time da SIRLab (3)

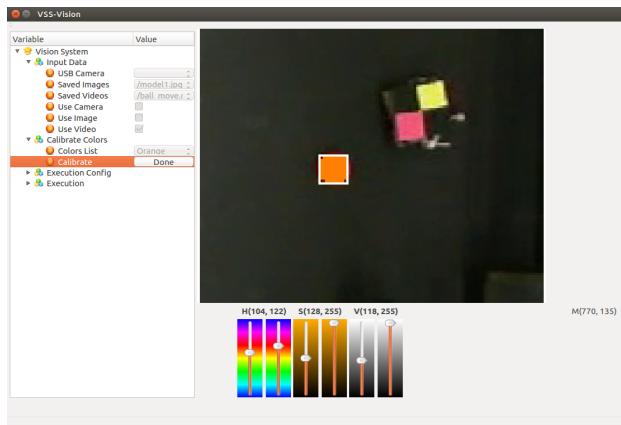


Figura 1.5: Calibração atual do time da SIRLab (3) após calibração

tacional, as mesmas foram implementadas juntamente com um sistema baseado em modelo para classificação cores, assim melhorando a acuracia do sistema. A rede é composta por três camadas, a camada de entrada possui três neuronios, uma vez que os valores de entradas são cada um dos valores HSV de um pixel; a camada de saida possui sete neuronios, onde seis são para a representação de cada uma das cores em que o pixel podera ser classificado e um setimo neuronio para representação do fundo; para a camada intermediaria foram utilizados vinte neurosios, pois apes estudos foi o valor onde indentificaram melhor acuracia. Sendo assim a topologia da rede MLP foi configurada como 3-20-7.

1.4 Organização da Trabalho

Este trabalho está dividido em cinco capítulos, incluindo esta introdução como capítulo inicial.

No segundo capítulo encontra-se a fundamentação teórica do texto, contento informações sobre processamento de imagens referente à detecção de objetos e cores. A descrição da probabilidade usada no trabalho, na descoberta do tamanho desejavel dos objetos, uma breve descrição sobre o futebol de robôs.

No terceiro capítulo encontra-se todo o desenvolvimento do projeto, suas classes, a des-

crição das tecnologias utilizadas no projeto, e o detalhamento de cada um dos tipos de calibração.

No quarto capítulo são apresentados os testes e os resultados obtidos em cada um dos testes do projeto.

No quinto capítulo são feitas as conclusões do trabalho e considerações finais, bem como expostas melhorias que possam vir a ser implementadas em trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 Processamento de Imagens

2.1.1 Detecção de Objetos

A detecção de objetos pode ser considerada uma técnica herdada do reconhecimento de padrões, da área de aprendizado de máquina, esta consiste em separar objetos por categorias de acordo com uma ou mais características específicas. Quando essa técnica se junta ao processamento de imagens, onde são estas características são acentuadas em um determinado objeto dentro da imagem para assim este se destacar, tornou-se possível a detecção de objetos em imagens, que dentro do campo de visão computacional é uma das áreas que mais obtêm a atenção de pesquisadores. O primeiro Framework de métodos que usam base de dados categorizando uma ou mais características de um objetos para fazer o reconhecimento através de aprendizado foi apresentado em 2001 por Viola e Jones(7). Desde o framework de Viola e Jones até os dias atuais muitos métodos e teorias para detecção já foram propostos e implementados como detecção de faces utilizando um classificador de redes neurais na intensidade de padrões de uma imagem, support vector machine para localizar rostos humanos e carros(8), análise de componentes principais, análise independente de componentes, fatoração de matriz não-negativa, análise discriminativa linear, boosting(9), além da classificação binária, onde se considera a detecção do objeto em tamanho fixo apenas variando na posição na imagem(10).

2.1.2 Detecção de Bordas

Para um objeto poder ser detectado por algum método de detecção a imagem passa por um processo de segmentação. A segmentação pode ser dita como o processo de divisão da imagem em objetos(11). De acordo com Wangenheim(12) o processo de segmentação se baseia em dois conceitos: similaridade e descontinuidade. A descontinuidade é o processo onde se separa o fundo das partículas e estas umas das outras, através de linhas, bordas ou pontos. Já a similaridade é o processo onde os pixels provenientes da descontinuidade são agrupados de acordo com a proximidade um dos outros para formar os objetos de interesse. De acordo com Canny(13) o processo de detecção de bordas é um processo simplificado que serve para diminuir drasticamente o total de dados a serem processados e ao mesmo

que o mesmo preserva informações valiosas sobre os objetos, este também é considerado um processamento de imagem de baixo nível, uma vez que age diretamente na imagem original apenas melhorando-a. É muito comum a ocorrência de ruídos quando se trata da detecção de bordas, e por sua vez para evitar esses ruídos é necessário a suavização da imagem antes de fazer a detecção. Vale(14) lembra que a suavização possui pontos negativos como perda de informação e deslocamento de estruturas de feições proeminentes no plano da imagem. Além disso, existem diferenças entre as propriedades dos operadores diferenciais comumente utilizados, o que ocasiona bordas diferentes. Assim, como dito por Ziou e Tabbone citados por Vale(14), se torna difícil encontrar um algoritmo que tenha bom desempenho em diferenciados contextos e capture os requisitos necessários aos estágios subsequentes do processamento. Quando se trata de detecção de bordas existem dois critérios(13) para essa detecção que devem ser levados em consideração, Taxa de Erro e Localização(14).

Taxa de Erro É importante que as bordas contidas na imagem não sejam confundidas ou perdidas e ainda que não sejam detectadas bordas falsas. É necessário que o algoritmo de detecção de borda tenha uma baixa taxa de erro para que seja eficiente.(12, 13, 14)

Localização A distância entre os pixels de borda encontradas pelo algoritmo e a borda atual deveriam ser o menor possível.(12)

Ao tentar aplicar esses dois critérios para desenvolver um modelo matemático para detecção de bordas sem a necessidade de base em regras preestabelecidas em seu artigo *A Computational Approach to Edge Detection* Canny percebeu que somente esses dois critérios não eram o suficiente para obter uma boa precisão da detecção de bordas. E então propôs um terceiro critério: Resposta.

Resposta Para contornar a possibilidade de mais de uma resposta para a mesma borda, ou seja o detector de bordas não deveria identificar múltiplos pixels de borda onde somente existe um único pixel. (12, 13, 14)

Com o acréscimo do terceiro critério então nota-se que o processo de detecção de bordas de Canny mostrou-se bastante flexível, independente da origem da imagem utilizada(14).

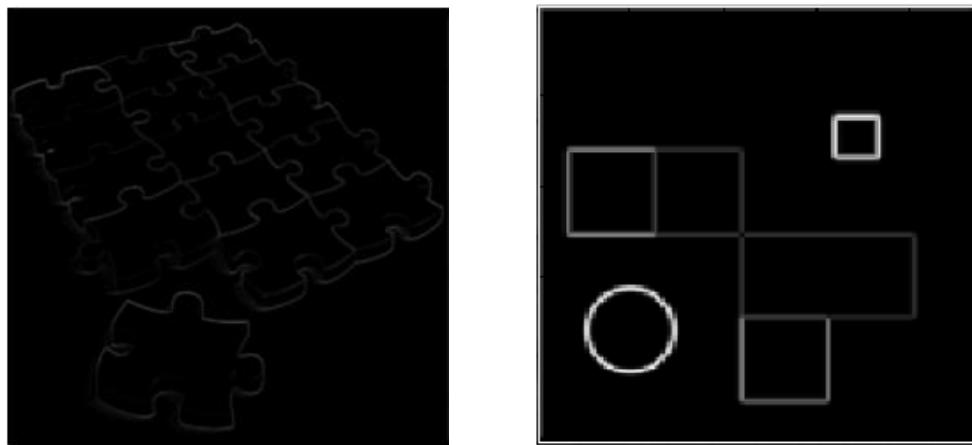


Figura 2.1: Detecção de Borda com Algoritmo de Canny (4)

2.2 Cores

O olho humano é capaz de identificar cores mesmo com as mais diferentes interferências, luminosidade, tonalidade, intensidade, entre outras ações de agentes externos graças aos **cones** e **bastonetes**. Os bastonetes são os responsáveis por distinguirem tons de cinza e pela visão periférica e tem como característica serem sensíveis a baixo nível de luminosidade(15), os cones, por sua vez, são sensíveis ao alto nível de iluminação e responsáveis pela percepção de cores(15). Segundo a teoria tricomática de Thomas Young e mais tarde estudada por Hermann von Helmholtz(15), a retina humana é formada por três tipos de fotopigmentos que seriam os três receptores de cor, ou seja respondiam ao comprimento de onda de apenas três cores: Vermelho, Verde e Azul. A informação obtida através do sistema visual humano é assimilada pelo nosso cérebro e ligando a cor a sua aparência levando em consideração o aprendizado que obtivemos sobre a mesma, já para uma máquina cores são números, códigos, cada cor contém um código específico e cada uma de suas variações também. Para o nosso cérebro é muito fácil entender, exemplo, que o verde, verde lima, verde escuro são todos verde, apenas com tonalidades diferentes, já para o computador estas são: (0,255,0),(50,205,50),(0,128,0), no padrão de cor RGB. Mas se for aplicado luminosidade nessas cores, por exemplo, elas ainda se tornam outras diferentes cores, um código diferente para cada luminosidade possível.

Para poder explicar as propriedades e comportamentos das cores em determinadas circunstâncias, surgiram os **Sistemas de cores**. Devido a complexidade existente em explicar todos os aspectos relacionados às cores são utilizados diversos sistemas para descrever as mais diferentes características das cores e sua percepção pelo ser humano(15). Dentro os sistemas mais conhecidos, estão o RGB, HSV E HSL, sistemas quais falaremos neste trabalho.

Segundo Azevedo(15) o universo de cores que podem ser reproduzidas por um sistema é chamado de **Espaço de Cores**. De acordo com Foley et. al citado por Souto(16) espaço de cores é um sistema tridimensional de coordenadas, onde cada eixo refere-se a uma cor primária. A quantidade de cor primária necessária para reproduzir uma determinada cor, é atribuída a um valor sobre o eixo correspondente. O espaço de cores pode ser entendido como a quantidade de detalhamento, tonalidades de uma cor, dentro do espectro de cores de um determinado modelo de cor.

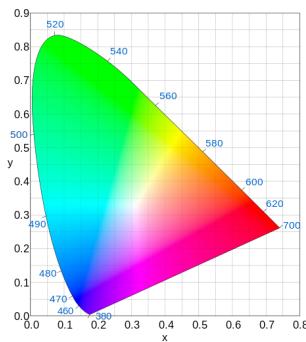


Figura 2.2: Universo de cores estabelecido pelo CIE.

Quando fez sua primeira experiência com a decomposição da luz em um prisma para obter cores Newton percebeu que não havia a cor branca. Ele tentou então misturar as sete cores que obteve para gerar a branca, sem sucesso. Para conseguir cobrir todas as alterações

e características as cores em 1921 a Comissão Internacional de Iluminação (CEI)(16) definiu três primárias (X, Y e Z) que podem ser combinadas para formarem todas as cores e entendeu-se que existe duas formas de se obter cores: através da emissão ou reflexão de luz, espaços RGB e CMY respectivamente, Figura 2.2. Assim entendeu-se então porque em seu experimento Newton não obteve sucesso para gerar a cor branca, pois para gerar a cor branca é necessário a soma das três cores primárias azul, verde e vermelho, uma vez que seu experimento utilizava a reflexão e não emissão de cores.

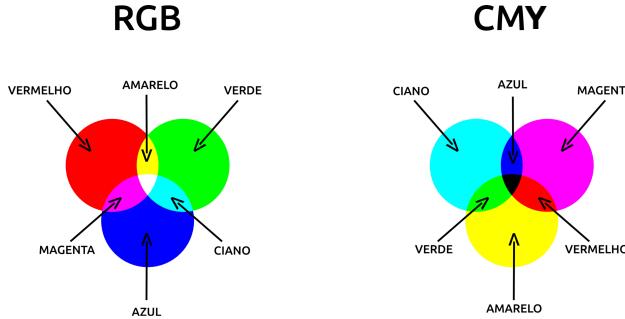


Figura 2.3: Exemplo dos espaços de cores RGB E CMY.

Para utilização prática de sistemas e espaços de cores e descreverem as cores foram criados modelos de cores, neste trabalho falaremos sobre o RGB e HSV, pois são os modelos usados durante o desenvolvimento. Modelos de cores são modelos matemáticos utilizados para classificação das cores de acordo com sua tonalidade, saturação, luminosidade ou crominância na tentativa de conseguir cobrir o maior número de cores possíveis e assim simulando a visão. A representação da cor é definida por um único ponto em um modelo tridimensional. Os modelos de cores tem função definir as cores nos programas gráficos de computadores de forma que combine com a percepção das cores pelo sistema visual humano e utiliza três eixos similares para definirem a cor(17).

O modelo de cores RGB pode ser considerado mais básico dos modelos de cores. Seu nome possui a mesma definição do espaço de cores RGB. Ele não utiliza de nenhum atributo como luminosidade ou tonalidade, por exemplo, para a definição da cor apenas a adição das cores primárias, azul, verde e vermelho. É este também o padrão mais usado e conhecido. Os valores de R, G e B variam de 0 à 255.

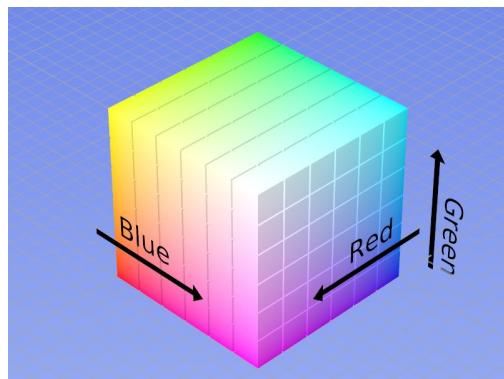


Figura 2.4: Exemplo do Modelo de Cor RGB. Horvath(5)

O modelo HSV define tonalidade (hue) que é a cor em si, variando de 0 a 360°, a saturação (saturation) que define o grau de pureza da cor, variando de 0 a 1, obtido pela

mistura da tonalidade com a cor branca e brilho (value) que tenta fazer referência à percepção humana(17) que é a intensidade da cor, escala de tons de cinza(15), variando também de 0 a 1.

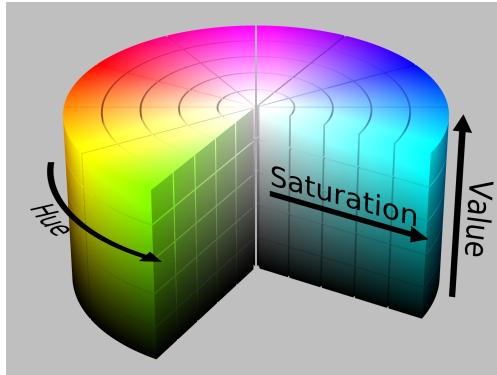


Figura 2.5: Exemplo do Modelo de Cor HSV, Horvath(5)

O sistema HSV utiliza definições de cor mais intuitivas que o conjunto de cores primárias, por isso são mais adequados quando se necessita obter várias tonalidades.

2.3 Futebol de Robôs

Visto como um domínio bastante complexo, dinâmico e imprevisível(18), o futebol de robôs surgiu como uma tentativa de promover pesquisas nos campos de Inteligência Artificial e robótica, pela avaliação teórica, algoritmos e arquiteturas através de problemas padrão(19). A Equipe Cedro se enquadra na categoria IEEE Very Small Size. Esta categoria é regulamentada pelo Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) e possui regras baseadas na MiroSot(2). O futebol de robôs se assemelha ao futebol humano onde o objetivo do jogo é fazer gols para vencer a partida, porém tendo regras adaptadas para o "ambiente" robótico. Rosa(2) em seu trabalho de graduação faz uma boa enumeração das regras básicas:

- A partida dura 10 minutos com dois tempos de 5 minutos;
- Há um intervalo de 10 minutos entre um tempo e outro;
- Cada time tem direito a dois tempos de 2 minutos que podem ser pedidos a qualquer momento;
- Caso a diferença de gols entre os dois times chegue a 10 a partida é encerrada;
- Uma falta ocorre quando há mais de um robô de um mesmo time dentro de sua própria área de gol ou quando um robô empurrar outro robô de outro time;
- Um pênalti ocorre quando a bola fica mais de 10 segundos dentro de alguma das áreas;
- Um chute-livre ocorre quando os robôs ficam travados por mais de 10 segundos, caso ocorra, o juiz posiciona a bola na marca de chute-livre mais próxima de onde ela ficou parada e posiciona os robôs de cada time equidistantes a bola;
- A cada início de partida ou gol feito a bola deve ser posicionada no centro do campo e os robôs devem ser posicionados de acordo com a posse de bola.

Capítulo 3

Desenvolvimento

Para o desenvolvimento foi escolhida a biblioteca OpenCV por ser OpenSource, multiplataforma, conter uma grande quantidade de métodos e algoritmos já implementados e pelo seu rápido desempenho de máquina. A linguagem escolhida para o desenvolvimento foi o C++ pois é uma linguagem de programação compilada, o que torna sua execução mais rápida que as linguagens interpretadas, dando ao sistema uma performance em tempo satisfatório.

3.1 Tecnologias Usadas

Para realização deste trabalho, irei utilizar a biblioteca de processamentos de imagens conhecida como OpenCV: Open Source Computer Vision Library. O trabalho será elaborado na linguagem C++, com uso do framework Qt para sua interface gráfica. Os passos detalhados do projeto e seu desenvolvimento estarão presente no Capítulo de Metodologia.

OpenCV Lançado em 1999 pela Intel(20), com objetivo de ser otimizada, portável e com um grande número de funções, o Open Source Computer Vision Library, OpenCV, se tornou se tornou uma ferramenta que possui mais de 2500 algoritmos e 40 mil pessoas em seu grupo de usuários(20). Já possui interface para as linguagens C++, C, Python e Java além de suporte para as principais plataformas com Windows, Linux, Mac OS, iOS e Android. A biblioteca lida tanto com imagens em tempo real, como vídeos e imagens estáticas.

Qt Qt é um framework de desenvolvimento de aplicações multiplataforma. Entre suas funcionalidades está a possibilidade de criar interfaces gráficas diretamente em C++ usando seu módulo Widgets.

C++ A linguagem de programação C++ foi projetado por Bjarne Stroustrup para fornecer eficiência e flexibilidade da linguagem C para programação de sistemas. A linguagem evoluiu a partir de uma versão anterior chamado C com Classes, o projeto C com Classes durou entre 1979 e 1983 e determinou os moldes para o C++. A linguagem foi oficialmente lançada em 1986(21).

3.2 Projeto

3.2.1 Organização do Projeto

O projeto foi desenvolvido seguindo o paradigma de programação Orientada à Objetos, esse paradigma baseia-se na utilização de objetos individuais para criação de um sistema maior e complexo. A IDE usada para o desenvolvimento foi a QT Creator, esta separada o projeto em três pastas: Headers, Sources e Forms. Na pasta Headers estão os arquivos de cabeçalho(.h), onde estão as declarações dos métodos e variáveis usados nas classes executáveis. Já na pasta Sources estão os arquivos fonte(.cpp), são nesses arquivos que os métodos declarados nos arquivos da pasta Header são implementados. Na pasta Forms está o arquivo de interface gráfica(.ui) que é usado no projeto para ser a ponte entre o usuário e as funções do sistema.

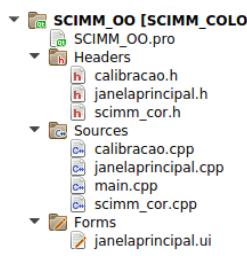


Figura 3.1: Organização das pastas do projeto

Com exceção do arquivo fonte `main`, cada arquivo de cabeçalho possui um arquivo fonte correspondente, formando assim um objeto, todo objeto é uma classe mas nem toda classe fomar um objeto. As classes desenvolvidas no projeto são: `Calibracao`, `scimm_cor` e `janela-principal`.

3.2.2 Classes

`main`

Esta é o que se chama de *ponto de entrada* em programação. *Ponto de entrada* é onde o sistema operacional irá iniciar a execução do sistema desenvolvido. Esta classe possui somente o método `main` e este instancia e inicia o objeto `JanelaPrincipal` que apresenta a interface gráfica para interação com o usuário.

`JanelaPrincipal`

É uma classe-objeto que utiliza da implementação de objetos `QWidget` e suas subclasses disponíveis pelo framework Qt, para a criação de uma interface gráfica que faz a interação com o usuário. Todos os métodos presentes nessa classe são para utilização gráfica, comportamento de botões e menu de seleção, para a comunicação com a classe de calibração.

Calibracao

É classe que contem todos os metodos e variaveis usados no processo de calibração, é tambem dentro dessa classe que são feitas todos os tipos de manipulação em imagem.

Os métodos contidos na classe Calibracao são:

Iniciar: Método onde é instanciado o objeto que faz referencia à **JanelaPrincipal**, para ser usado quando houver necessidade de comunicação com a interface que não seja por meio de retorno, este metodo tambem é onde se inicializa o a camera, a partir do id contido na interface, e se verifica se a mesma esta disponivel.

ConfigurarCamera: Método onde a imagem de exibição é ajustando o tamanho da tela a ser usada durante a detecção de objetos, alem do ajuste de brilho e contraste para tornar mais nitido os contornos durante a detecção de objetos.

ReconhecerFundoExtrairObjetos: Método onde se utiliza o algoritmo de subtração de fundo para identificar, inicialmente, o campo e uma vez identificado se separa os objetos que não fazem parte do fundo inicial, a extração dos objetos. A extração dos objetos gerando uma imagem *mascara* que sera usada na detecção dos objetos para que a mesma ser executada com mais precisão, uma vez que os objetos coloridos eram os unicos que não faziam parte da imagem inicial, o fundo.

Calibrar: Metodo onde se a imagem atual da camera é obtida e ajustada de acordo com os parametros estabelecidos no metodo**ConfigurarCamera** e iniciado o método **DetectarObjetos**. Uma vez que os objetos foram identificados se faz o uso do metodo **Calcular** para serem analisados os valores de cada pixel de cada objetos e estes classificados de acordo com as cores pré definicas pelo sistemas. Uma vez terminado o processo de calibração é exibida na tela para o usuario os objetos da cor que esta selecionada no menu de cores.

DetectarObjetos: Método onde serão aplicados filtros de diminuição de ruido, detectadas as bordas existentes na imagem, detectados os objetos a partir das bordas contidas na imagem, o aumento da precisão do contorno dos objetos e diminuição do tamanho do objeto de acordo com a porcentagem de borda a ser eliminada.

ObterPorcentagem: Método simples para devolver a porcentagem de um valor, ao ser informado o valor e a porcentagem escolhida.

Calcular: Método onde cada pixel pertencente à cada um dos objetos encontrados é analisado, seu valor H é categorizado de acordo com as cores pre-definidas no sistema e adicionado à cor correspondente. O algoritmo de categorizacao:

SCIMM_COR

Classe de cor utilizada do sistema. Onde são salvos os valores máximos e mínimos de HSV. Possui seis valores, um mínimo e um máximo para H, S e V.

3.3 O Sistema

O sistema consiste na apresentação da **interface gráfica** ao usuário. A **interface gráfica** que por sua vez oferece as duas possibilidades ao usuário, de acordo com o tipo de calibração escolhido o sistema inicia a rotina de calibração referente. Após a execução de toda o sistema é finalizada, como mostra o diagrama de fluxo na Figura 3.2.

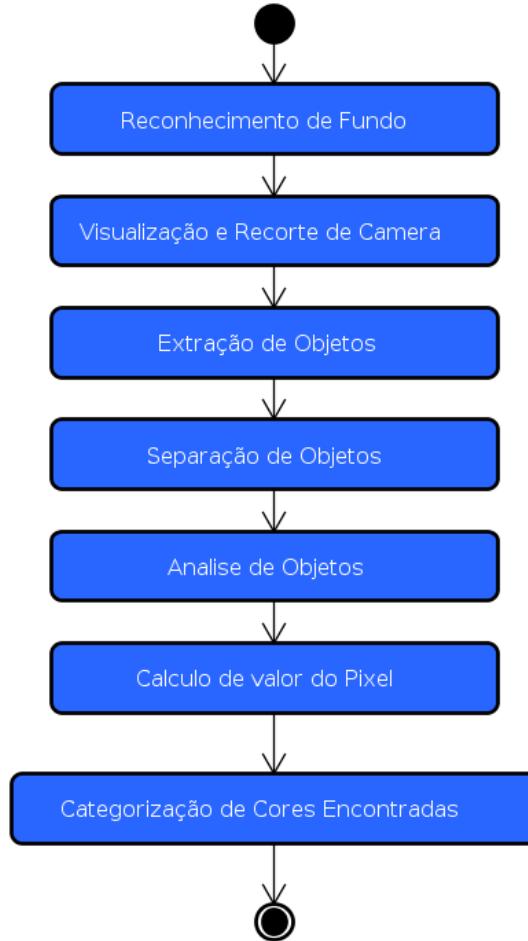


Figura 3.2: Diagrama de Fluxo

A rotina de calibração automática possui cinco etapas e os mesmos possuem o mínimo de interação possível do usuário. Nas sessões à seguir explicarei detalhadamente cada uma das etapas do processo de calibração automática

3.3.1 1^a Etapa - Configuração de Camera

Para melhor desempenho do algoritmo de detecção de objetos, a imagem é recortada somente para o tamanho necessário do campo. O recorte de imagem foi feito utilizando a função `setMouseCallback` para possibilitar a interação do usuário na imagem por meio do mouse, sua utilização é dada da seguinte maneira:

```
cv::setMouseCallback(src_window, mouseHandler, 0);
```

Tem como primeiro parametro **src_windows** que indica a janela na qual a função receberá a interação, o segundo, **mouseHandler**, indica a função na qual esta implementada a interação e o ultimo parametro, **0**, indica parametros opcionais, neste caso não usaremos nenhum então foi usado o numero 0. Dentro da função **mouseHandler** são identificados os pontos inicial e final da seleção na tela e utilizada a função *rectangle* para demarcar a seleção na tela. A utilização da função *rectangle* completa fica da seguinte maneira:

```
cv::rectangle(frameA, point1, point2, CV_RGB(255, 0, 0), 2, 5, 0);
```

A função rebece os parametros **frameA** indicando a imagem na qual será demarcada a area selecionada, depois o parametro **point1** que é o ponto incial de seleção na imagem, **point2** que é o ponto final da seleção. **CV_RGB(255, 0, 0)** que indica a cor da demarcação, **2** indicando a expressura da demarcação, **5** que significa o tipo de linha a ser utilizado na demarcação e **0** que é o numero de bits fracionarios.

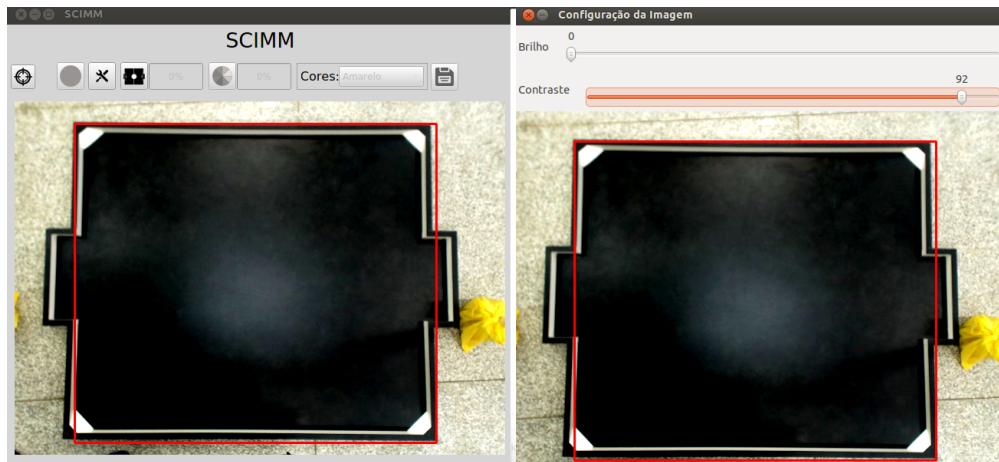


Figura 3.3: Configuração de Camera

Após confirmada a escolha do tamanho da tela este é então salvo na variavel nomeada *tamanho*, está então sera usado durante todo o processo de calibração. Além de recorte da imagem, nessa etapa tambem são configurados Brilho e Constraste de forma que os objetos de cor no campo se tornem mais vividos, se tornando assim tambem mais nitidos, para que os processos de separaçao e detecçao de objetos sejam mais precisos. A configuração de Brilho e Constraste utiliza o metodo *convertTo* da biblioteca *OpenCV* que é utilizada para o melhoramento da imagem antes da detecção dos objetos, a utilização completa fica da seguinte maneira:

```
frameA.convertTo(frameA, -1, contrast_value / 50.0, brightness_value)
```

Esta função recebe quatro parametros. O primeiro **frameA** informa aonde sera salvo o resultado da conversão. O segundo **-1** indica o tipo da matrix, ou numero de canais, da imagem a ser gerada, usa-se **-1** quando se deseja que se use os valores semelhantes aos da imagem da imagem original(22), O terceiro **contrast_value / 50.0** indica o valor de contraste, ou alpha, a ser usado para multiplicar os valores do pixel da imagem(22) e por ultimo **brightness_value** que é o valor do brilho, ou beta, a ser adicionado à imagem. É importante ressaltar que a configuração de Brilho e Constraste é somente usada para a melhor preciso

na detecção dos objetos, no momento da analise do pixel, a imagem esta *limpa* sem alguma alteração.

3.3.2 2^a Etapa - Reconhecimento de Fundo

Um principais problemas que ocorrem na detecção de objetos é a confusão do fundo junto ao proprio objeto, fazendo assim que o mesmo seja detectado porém não seu contorno correto, ou em outras vezes ignorado por ser considerado parte do fundo. Para eliminar este problema foi utilizado a tecnica Subtração do fundo usando Mistura de Gaussianas, por meio do objeto `createBackgroundSubtractorMOG2()` disponivel na biblioteca *OpenCV*. Esta tecnica utiliza um algoritmo de analise pixel a pixel e que classifica o mesmo baseando-se na distribuição da gaussiana que o representa. Para separar o fundo do resto da imagem é levada em consideração que a gaussiana que representa o fundo tenha grande peso e baixa variância, isso significa que a mesma ocorre frequentemente e varie pouco no tempo. O algoritmo atualiza o modelo de fundo a cada quadro da imagem baseando-se na variancia do objetos da mesma e de sua variancia. O objeto criado pela biblioteca, por meio do metodo `apply`, analisa quadro a quadro a imagem, a compara com o fundo obtido e gera uma imagem chamada de **mascara** com os objetos que não fazem parte do fundo, no exato momento da imagem, com o passar dos quadros o objeto se torna parte do fundo. Uso do metodo:

```
pMOG2->apply(frame, mask);
```

Onde **frame** significa a imagem atual capturada pela camera e **mask** a mascara gerada pela diferença da imagem atual com o modelo de fundo.



Figura 3.4: Configuração de Camera

Sabendo inicialmente que o modelo de fundo do objeto **pMOG2** esta vazio, basta que ele seja executado algumas vezes para que o campo se torne o modelo de fundo. Nesse caso a mascara gerada pelo metodo não chega a ser utilizada, porém sera na 3^a etapa.

3.3.3 3^a Etapa - Extração dos Objetos do Fundo

Como visto na 1^a etapa, o objeto **pMOG2** chamando o metodo *apply* o mesmo gera uma mascara de diferença da imagem atual para o modelo de fundo. Sendo assim para obtermos os objetos que virao a ser detectados basta que o metodo *apply* seja chamado um numero de vezes suficiente para criar a mascara e o mesmo nao alterar o modelo de fundo.



Figura 3.5: Geração de Mascara

3.3.4 4^a Etapa - Detecção e validação de Objetos

A detecção dos objetos a serem calibrados é dada pelo algoritmo de detecção de bordas de Canny. Como mais um recurso para eliminação de ruidos e melhoria da imagem antes de ser executado a detecção de objetos atravez da detecção de bordas é utilizado desfoque na imagem. O algoritmo de Canny já está implementado dentro da biblioteca OpenCV e com a seguinte usagem:

```
Canny(src_gray, canny_output, limiar, limiar * 3, 3);
```

O algoritmo de Canny utiliza por padrão imagem em padrões de cinza, sendo assim **src_gray** é a imagem original transformada para escala de cinza, esta é a imagem na qual o algoritmo sera aplicado. **canny_output** será a imagem de saída da função. **limiar** e **limiar*3** são os limites mínimos e máximos para considerar uma borda. **3** é o valor de apertura ou kernel, o valor 3 é utilizado como padro.

Apos o uso do algoritmo de Canny para detecção de bordas é necessario então fazer uso da função *findContours*, nativa no *OpenCV* para detecção de contornos.

```
findContours(canny_output, contours, hierarchy, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0))
```

O primeiro parametro, **canny_output**, é a imagem que o algoritmo de Canny gerou com as bordas encontrada na imagem, e é a imagem que o método *findContours* ira utilizar para detectar os contornos, **contours** é o parametro que indica onde serão salvos os contornos

encontrados, cada contorno é armazenado como sendo um vetor de pontos (22). **hierarchy** é onde será salvo um vetor de informações sobre a topologia da imagem, e terá como total de elementos o mesmo numero que o total de contornos encontrado(22). O quarto parametro, **CV_RETR_EXTERNAL** indica o modo de obtenção de contornos, nesse caso *CV_RETR_EXTERNAL* indica que o metodo só obterá os contornos exteriores(22). **CV_CHAIN_APPROX_SIMPLE** indica o metodo que sera usado para aproximação de contornos, o metodo *CV_CHAIN_APPROX_SIMPLE* comprime segmentos horizontais, verticais, diagonais e deixa apenas os seus pontos finais(22). E o ultimo parametro, **Point(0, 0)**, indica o valor a ser usado para deslocar a imagem ao encontrar os objetos, neste caso esse valor é 0 para Y e 0 para X, pois não sera necessário.

Uma vez obtidos os contornos é necessários que se faça a eliminação de vértices dos polígonos encontrados nos objetos deixando assim o objeto mais preciso. Isso é necessário para deixar a forma encontrada mais precisa dada a forma original. Para este ajuste foi usado o metodo *approxPolyDP*, já implementado dentro da biblioteca OpenCV. Esse método deve ser aplicado em cada um dos contornos encontrados, e foi utilizado da seguinte maneira:

$$\text{approxPolyDP}(\text{Mat}(\text{contours}[i]), \text{contours_poly}[i], 3, \text{true})$$

Onde o método inicia recebendo como parâmetro, **Mat(contours[i])** que é a criação de uma nova imagem, somente com aquele único objeto, que está sendo analisado. A seguir é informado no segundo parâmetro a variável de destino **contours_poly[i]**, onde será salvo o objeto com a eliminação dos vértices. O terceiro parâmetro indica o valor do *epsilon*, usado o valor **3** que especifica a precisão da aproximação, a distância máxima entre a curva original e a sua aproximação(22). O último parâmetro indica se a curva aproximada será fechada ou não, foi usado o valor **true** pois neste caso fechar um uma curva é necessário para que o objeto onde está a cor, seja identificado e analisado na probabilidade. Por último os objetos possuem sua borda ignorada, sendo assim calculado o tamanho interior dele, para que por ventura não hajam pixels de cor preta ou derivadas a serem calculadas.

3.3.5 5ª Etapa - Classificação do Pixel

Para que possam ser feita classificação do pixel é necessário que se faça, primeiramente a conversão da imagem obtida pela câmera, normalmente no espaço de cores RGB, para o espaço de cores HSV, pois a mesma lida melhor com diferenças de luminosidade. A biblioteca **OpenCV** converte o espaço de cor usando a função *cvtColor* que utiliza da imagem original, e de uma imagem vazia com memória alocada para ser salva a imagem após a conversão, além do parâmetro do tipo de conversão, Exemplo do uso do método:

$$\text{cvtColor}(\text{frame}, \text{HSV}, \text{CV_RGB2HSV});$$

Após a conversão é necessário, então, ser feita uma análise dos objetos encontrados. Para cada objeto serão todos os seus pixels, cada pixel separadamente o mesmo categorizado de acordo com o intervalo de valores na tabela a baixo:

Os valores pré-definidos de H foram obtidos

Estes valores foram obtidos através da conversão dos valores de cada cor no universo real para dentro da biblioteca OpenCV. Como visto na sessão 2.2 Cores da fundamentação

Cor	Intervalo de H
Laranja	de 0 à 20
Amarelo	de 21 à 30
Verde	de 61 à 90
Azul	de 91 à 120
Roxo	de 125 à 160
Rosa	de 161 à 168
Vermelha	de 169 à 180

Tabela 3.1: Intervalo de Valores de Cores - HSV

teorica, o intervalo de valores da matiz(h) é de 0 a 360º graus, porém o valor máximo de armazenamento de um bit é 255, foi incorporada na biblioteca o intervalo de valores de 0 à 180 sendo assim metade do valor do intervalo no mundo real. Uma outra ressalva deve ser feita quando à conversão de valores. O modelo padrão de uma imagem é o RGB, porém dentro da biblioteca espaço padrão é o BGR, quando a imagem é obtida a mesma possui alguns valores invertidos, ou seja, os intervalos de valores da tabela acima podem parecer diferentes do que seria considerado real se forem feitos conversao de valores RGB para HSV. Como não há pre-definição destes valores disponivel pela biblioteca, foi feito um estudo dos intervalos onde foram levados em condiseração a conversao padrao do mundo real, dividindo os valores por dois, uma vez o intervalo dentro da biblioteca é, que corresponde a metade do intervalo no mundo real, juntamente com uma serie de testes para encontrar os valores analisando seu resultado ao limitar a exibição da imagem da camera para determinados intervalos de cores.

Durante o desenvolvimento foi observador que, em sua maioria, as cores necessitavam de um valor S e V pré estabelecido. Para as cores Vermelho, Rosa, Azul, Laranja e Verde foi pré estabelecido o valor de 100 para ambos, S e V, já para Amarelo e Roxo ambos os valores assumem 50.

3.3.6 6^a Etapa - Gerar Arquivo de Cores

Assim que todas as cores já estiverem sido assimiladas e a calibração finalizada, gera-se um arquivo chamado **cores.arff** com

Capítulo 4

Testes e Resultados

Este capítulo será dividido em duas seções: **Calibração** e **Testes**. A seção **Calibração** terá a descrição do processo de calibração de cores, a preparação do campo, configuração de imagem, etc. A seção **Testes** é onde descrevo o testes usados para a análise de eficiência do sistema desenvolvido.

4.1 Calibração

A calibração aqui descrita ocorreu dia no 19 de Agosto de 2016, entre 17:36 e 17:39. A rotina de calibração do sistema, já descrita no capítulo 3 deste trabalho, envolve primeiramente uma aquisição da imagem do campo vazio, sem nada no mesmo, como visto na Figura 4.1.

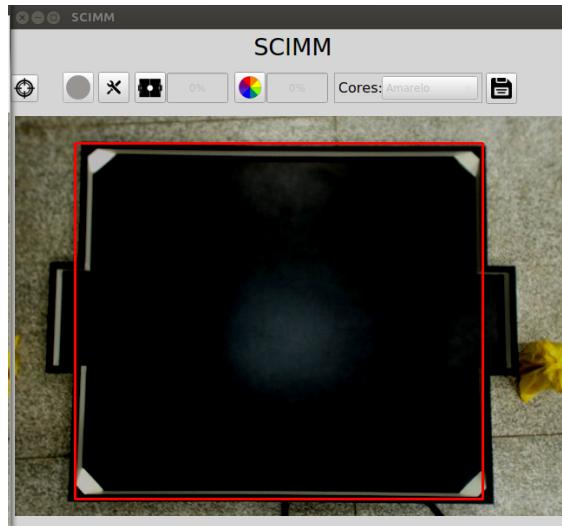


Figura 4.1: Imagem do fundo com a seleção de campo

Após ter o campo identificado pelo sistema, deve-se dispor sobre o campo os objetos coloridos, com as cores cujo se deseja obter o intervalo. É preferido que se usem tiras coloridas ao invés de quadrado, de $4\text{cm} \times 4\text{cm}$ como é comumente feito. Essa preferência se dá pois quanto maior o tamanho da tira de cor, maior será o espectro de cores que seja analisado,

assim sendo possível uma melhor qualidade de calibração. Neste teste foram dispostos no campo tiras coloridas com largura entre *17cm* e *40cm* e altura entre *5,5cm* e *10,5cm*, cada cor com 3 tiras uma vez que o campo foi separado em três partes, significando as partes com diferentes luminosidades, sendo assim cada uma das tiras colocada em uma das partes do campos, como é possível visualizar na Figura 4.2.



Figura 4.2: Objetos dispostos no campo para calibração

Como resultado da rotina de calibração foi gerado um arquivo .arff contendo 14 linhas. O arquivo é descrito na tabela abaixo:

Valor	Descrição
21.50.50	Valor mínimo da cor Amarelo
30.255.255	Valor máximo da cor Amarelo
92.100.100	Valor mínimo da cor Azul
120.255.255	Valor máximo da cor Azul
62.30.100	Valor mínimo da cor Verde
90.255.255	Valor máximo da cor Verde
169.100.100	Valor mínimo da cor Vermelho
179.255.255	Valor máximo da cor Vermelho
0.100.100	Valor mínimo da cor Laranja
20.255.255	Valor máximo da cor Laranja
161.100.100	Valor mínimo da cor Rosa
168.255.255	Valor máximo da cor Rosa
126.30.30	Valor mínimo da cor Roxo
160.255.255	Valor máximo da cor Roxo

Tabela 4.1: Arquivo cores.arff

4.2 Testes

O teste aqui descrito ocorreu dia no 26 de Agosto de 2016, entre à 13:28 e 16:28. Para elaboração do teste optou-se por dividir o campo no maior numero de partes possíveis, e ainda assim que essas partes coubessem todas as 7 cores usadas. Essa divisão foi feita para simular as cores em todas as partes possíveis do campos. O campo foi dividido então em 15 partes de 29cm por 41cm , nomeadas alfabeticamente de A à O, como mostrado na Figura 4.3.

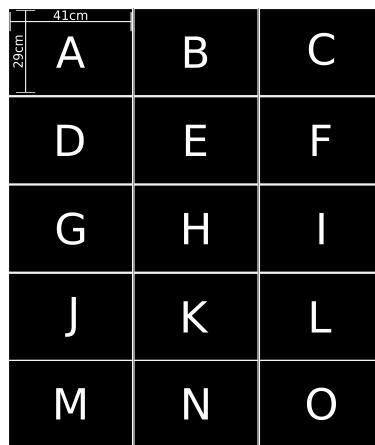


Figura 4.3: Divisão do campo em quinze partes nomeadas alfabeticamente.

Em cada uma das partes do campo estavam dispostas sete cores: Vermelho, Amarelo e Azul na primeira linha, Verde, Roxo, Laranja e Rosa na segunda. As cores estão distantes verticalmente 6cm , na primeira linha a distância entre as cores é de $7,25\text{cm}$ e na segunda linha de 5cm . Um melhor detalhamento da disposição das cores é mostrado na Figura 4.13.

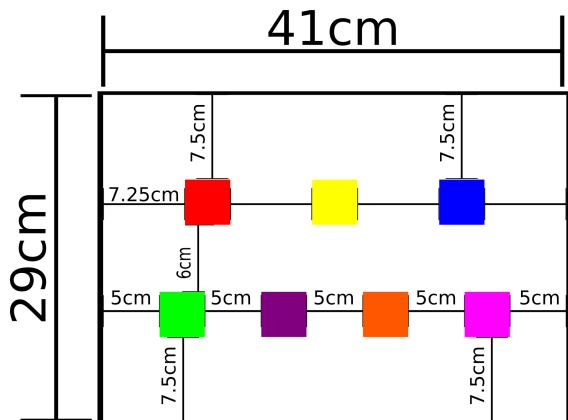


Figura 4.4: Disposição de cada parte quanto as cores



Figura 4.5: Campo apos terem sido dispostas as cores

4.2.1 Cores Comuns

Amarelo



Figura 4.6: Imagem somente com os objetos dentro do intervalo do valor da cor amarela

Dentre os objetos da cor amarela o sistema encontrou quatorze deles completamente e apenas um, que devido a luminosidade implicada em seu centro deixando a tonalizada muito perto do branco, não totalmente preenchido.

Tipo de Objeto	Quantidade	%
Objetos Completos	14	93,33
Objetos Com Falha de Preenchimento	1	6,66
Objetos Com Diminuição de Contorno	0	
Objetos Extrapolados	0	
Objetos Com Diminuição de Área	0	
Objetos Com Falhas Críticas	0	

Tabela 4.2: Categorização Dos Objetos

Dentre as classificações dos objetos as que *Objetos Com Falhas Críticas* e *Objetos Com Falha de Preenchimento* que juntos somam 6,66% dos objetos.

Azul

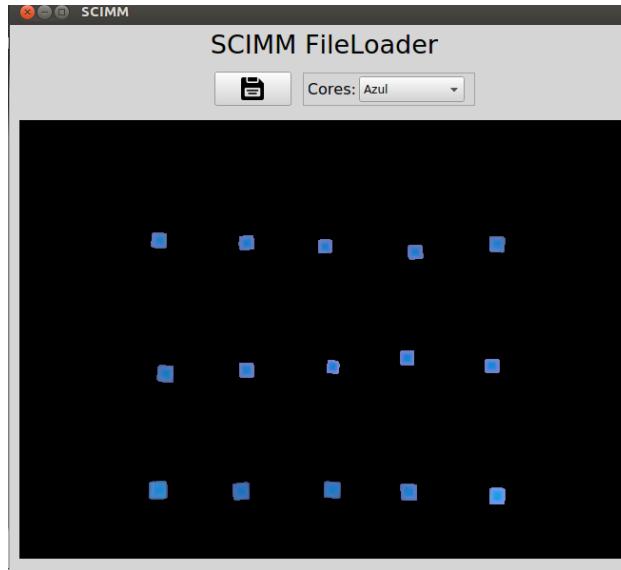


Figura 4.7: Imagem somente com os objetos dentro do intervalo do valor da cor azul

Os objetos da cor azul foram os que obtiveram os melhores resultados, os quinze objetos foram encontrados de forma preenchida. Apesar dos quinze estarem totalmente preenchidos um dos objetos apresentou um tamanho reduzido aos demais devido ao fato de sua borda que não ter sido totalmente detectada.

Tipo de Objeto	Quantidade	%
Objetos Completos	14	93,33
Objetos Com Falha de Preenchimento	0	
Objetos Com Diminuição de Contorno	1	6,66
Objetos Extrapolados	0	
Objetos Com Diminuição de Área	0	
Objetos Com Falhas Críticas	0	

Tabela 4.3: Categorização Dos Objetos

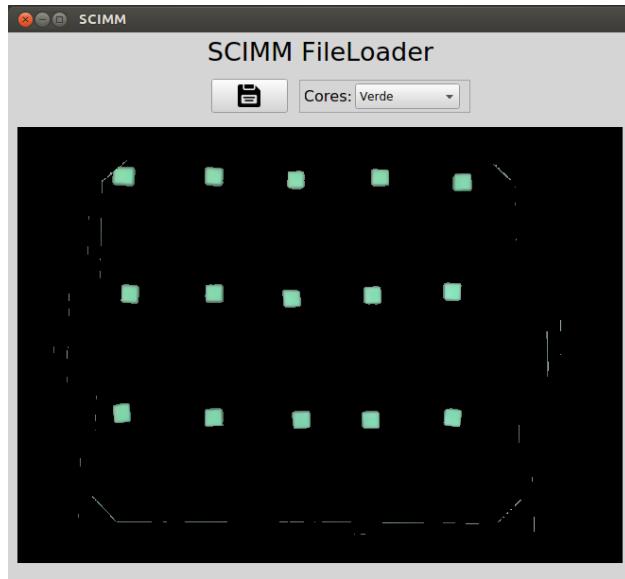
Verde

Figura 4.8: Imagem somente com os objetos dentro do intervalo do valor da cor verde

Os objetos da cor verde foram satisfatoriamente encontrados, com seu preenchimento total e não havendo perda de área devido a qualquer interferência de luz em sua borda.

Tipo de Objeto	Quantidade	%
Objetos Completos	15	100
Objetos Com Falha de Preenchimento	0	
Objetos Com Diminuição de Contorno	0	
Objetos Extrapolados	0	
Objetos Com Diminuição de Área	0	
Objetos Com Falhas Críticas	0	

Tabela 4.4: Categorização Dos Objetos

Rosa

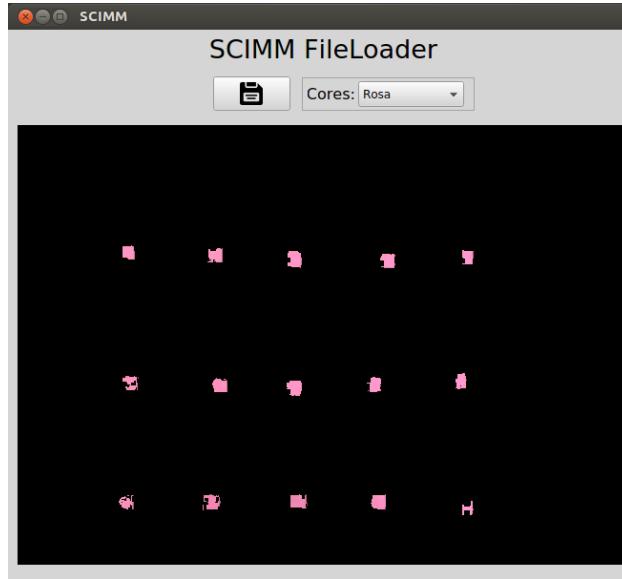


Figura 4.9: Imagem somente com os objetos dentro do intervalo do valor da cor rosa

Dentre os objetos rosa detectados, quatro obtiveram falhas em sua detecção, falhas de preenchimento e diminuição de borda, estes podem ser desconsiderados dos objetos. Dentro os outros onze: três foram detectados sem falhas de preenchimentos apenas com diminuição de sua área para aproximadamente metade da área real do objeto; os outros oito foram encontrados apesar com diminuição de área devido a diminuição de borda.

Tipo de Objeto	Quantidade	%
Objetos Completos	0	
Objetos Com Falha de Preenchimento	0	
Objetos Com Diminuição de Contorno	8	53,33
Objetos Extrapolados	0	
Objetos Com Diminuição de Área	3	20
Objetos Com Falhas Criticas	4	26,66

Tabela 4.5: Categorização Dos Objetos

Dentre as classificações dos objetos as que *Objetos Com Falhas Criticas* e *Objetos Com Falha de Preenchimento* que juntos somam 26,66% dos objetos. Apesar de ser um numero não tão baixo, a cor rosa não é uma cor que a equipe Cedro costuma usar em seus jogos, sendo assim, essa taxa de erro não influencia na eficiencia do sistema.

Roxo



Figura 4.10: Imagem somente com os objetos dentro do intervalo do valor da cor roxo

Todos os objetos roxos dispostos no campo foram encontrados pelo intervalo da cor. Dentro os quinze, quatro não apresentaram problema algum e foram completamente detectados; oito possuiram diminuição em seu contorno e três diminuição em sua área relativa.

Tipo de Objeto	Quantidade	%
Objetos Completos	4	26,66
Objetos Com Falha de Preenchimento	0	
Objetos Com Diminuição de Contorno	8	53,33
Objetos Extrapolados	0	
Objetos Com Diminuição de Área	3	20
Objetos Com Falhas Críticas	0	

Tabela 4.6: Categorização Dos Objetos

4.2.2 Cores Com Problemas Conhecidos

Cores como Vermelho e Laranja possuem o problema de por vezes, devido a interferencia externas, se assemelharem a outras. Este problemas ja sao de conhecimento da area.

Vermelho

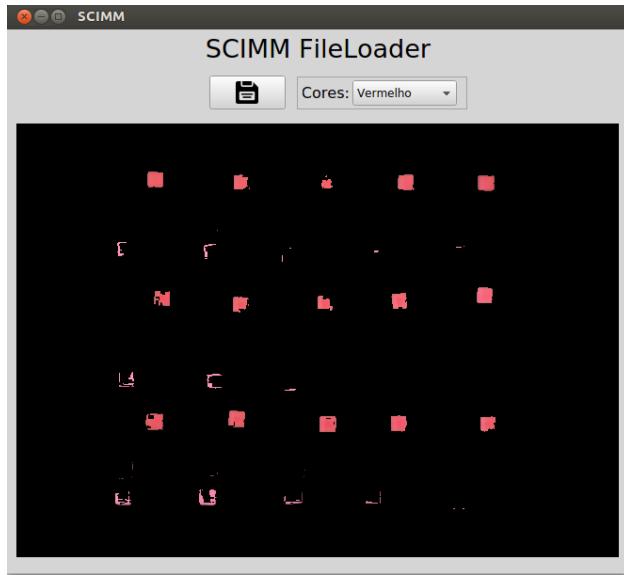


Figura 4.11: Imagem somente com os objetos dentro do intervalo do valor da cor vermelho

A cor rosa, em determinadas luminosidades pode acabar sendo semelhante a cor vermelha, por este motivo que alguns traços dos objetos rosas estão aparecendo dentro do intervalo de calibração. Porem serão ignorados, uma vez que é possível de ser feito a separação entre as cores via software.

Tipo de Objeto	Quantidade	%
Objetos Completos	8	53,33
Objetos Com Falha de Preenchimento	1	6,66
Objetos Com Diminuição de Contorno	4	26,66
Objetos Extrapolados	13	
Objetos Com Diminuição de Área	1	6,66
Objetos Com Falhas Críticas	1	6,66

Tabela 4.7: Categorização Dos Objetos

Dentre as classificações dos objetos as que *Objetos Com Falhas Críticas* e *Objetos Com Falha de Preenchimento* que juntos somam somente 13,32% dos objetos.

Laranja

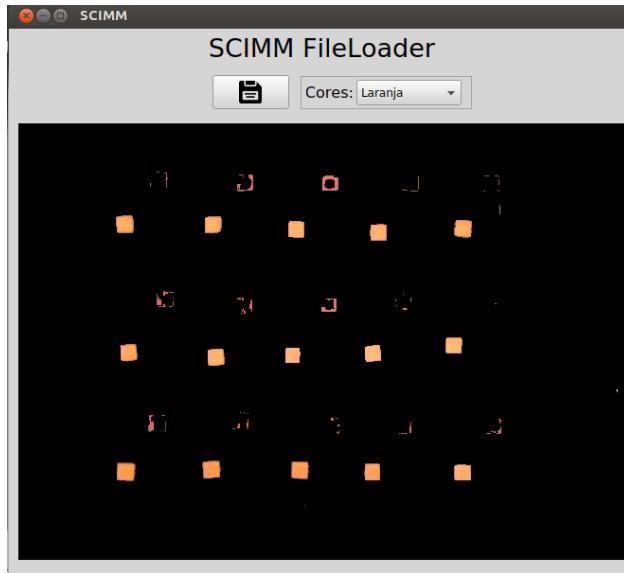


Figura 4.12: Imagem somente com os objetos dentro do intervalo do valor da cor laranja

Devido à um problema muito comum na área de calibração de cores, a cor laranja possui o problema de ser, por muitas vezes, semelhante a vermelha, e devido a luminosidade implicada tanto em uma quanto na outra, ambas as cores tendem a se tornarem próximas. Sabendo deste problema, o fato de terem sido encontrados objetos da cor vermelha dentro do intervalo de valores da cor laranja é ignorado e somente serão levados em consideração os objetos visualmente laranjas. Os quinze objetos da cor laranja foram encontrados com precisão. Todos possuindo seu completo preenchimento e borda.

Tipo de Objeto	Quantidade	%
Objetos Completos	15	100
Objetos Com Falha de Preenchimento	0	
Objetos Com Diminuição de Contorno	0	
Objetos Extrapolados	8	
Objetos Com Diminuição de Área	0	
Objetos Com Falhas Críticas	0	

Tabela 4.8: Categorização Dos Objetos

Totais

De modo total estavam disposto pelo campo 105 objetos coloridos. Destes, 66,7% dos objetos foram encontrados corretamente, 1,9% foram encontrados com falhas de preenchimento, 20% apresentaram perda de contorno, 6,67% apresentaram diminuição da área e 4,76% apresentaram falhas e faltas que prejudicaram totalmente o objeto.

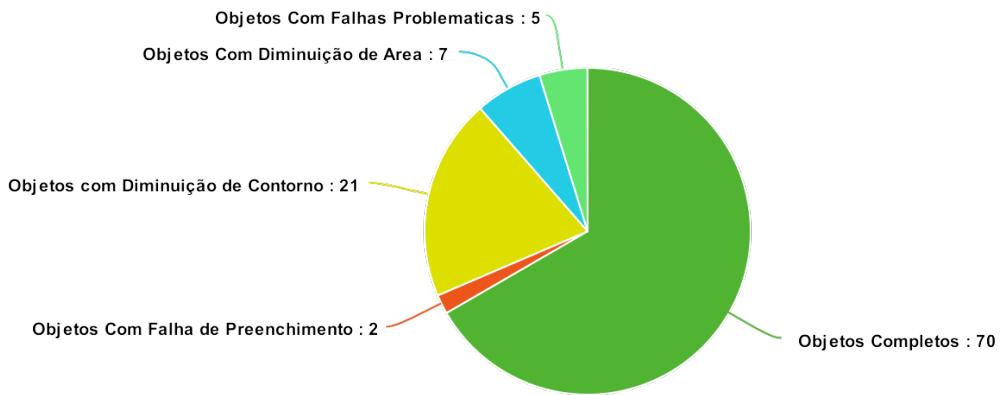


Figura 4.13: Gráfico de análise do resultado dos testes

Sabendo destes valores, os que podem ser considerados criticos na detecção de objetos através da sua cor são *Objetos Com Falhas Criticas* e *Objetos Com Falha de Preenchimento* que juntos somam 6,67% dos objetos.

Bibliografia

- 1 PENHARBEL, E. A. et al. Filtro de imagem baseado em matriz rgb de cores-padrão para futebol de robôs. *Submetido ao I Encontro de Robótica Inteligente*, 2004.
- 2 ROSA, J. F. da. *Construção de um time de futebol de robôs para a categoria IEEE Very Small Size Soccer*. Dissertação (Trabalho de Conclusão de Curso) — FACULDADE DEE DUCAÇÃO TECNOLÓGICA DO ESTADO DO RIO DE JANEIRO, 2015. Disponível em: <<http://docplayer.com.br/11866286-Construcao-de-um-time-de-futebol-de-robos-para-a-categoria-ieee-very-small-size-soccer.html>>.
- 3 SIRLAB. Imagens hsv, hsl e rgb. In: . [s.n.]. Acessado 06/06/2016 14:50. Disponível em: <<https://github.com/SIRLab/VSS-Vision/wiki>>.
- 4 SAINI, S.; KASLIWAL, B.; BHATIA, S. Comparative study of image edge detection algorithms. *arXiv e-print service in the fields of physics, mathematics, computer science, quantitative biology, quantitative finance and statistics, Cornell University Library*, 2013. Disponível em: <<https://arxiv.org/pdf/1311.4963>>.
- 5 HORVATH, M. "Imagens HSV, HSL e RGB". Disponível em: <<http://isometricland.net/artwork/artwork.php>>.
- 6 PENHARBEL, E. A. et al. Time de futebol de robôs y04 do centro universitário da fei.
- 7 VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: IEEE. *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. [S.l.], 2001. v. 1, p. I–511.
- 8 NASCIMENTO, M. C. *Detecção de Objetos em Imagens*. Dissertação (Trabalho de Graduação) — Centro de Informática, Universidade Federal de Pernambuco, 2007. Disponível em: <www.cin.ufpe.br/~tg/2007-2/mcn2.doc>.
- 9 ROTH, P. M.; WINTER, M. Survey of appearance-based methods for object recognition. *Inst. for Computer Graphics and Vision, Graz University of Technology, Austria, Technical Report ICGTR0108 (ICG-TR-01/08)*, 2008.
- 10 AMIT, Y.; FELZENZWALB, P. Object detection. In: IKEUCHI, K. (Ed.). *Computer Vision, A Reference Guide*. New York, NY, USA: Springer, 2014. v. 2, p. 537–542.
- 11 GONZALEZ, R.; WOODS, R. *Digital Image Processing*. Pearson/Prentice Hall, 2008. ISBN 9780131687288. Disponível em: <<https://books.google.com.br/books?id=8uGOnjRGEzoC>>.

- 12 WANGENHEIM, A. Von. encontrando a linha divisória: Detecção de borda. *Departamento de Informática e Estatística-Universidade Federal de Santa Catarina, 2013a*, v. 16, 2014. Disponível em: <www.inf.ufsc.br/~visao/bordas.pdf>.
- 13 CANNY, J. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8, n. 6, p. 679–698, Nov 1986. ISSN 0162-8828.
- 14 VALE, G. M. do; POZ, A. P. D. O processo de detecção de bordas de canny: Fundamentos, algoritmos e avaliação experimental. In: BERGIN, T. J.; GIBSON, R. G. (Ed.). *Simpósio Brasileiro de Geomática*. [S.l.: s.n.], 2002. p. 292–303.
- 15 AZEVEDO, E.; A., C. *Computação Gráfica: Geração de Imagens*. [S.l.]: Elsevier, 2003. ISBN 9878535212525.
- 16 SOUTO, R. P. *Segmentação de imagem multiespectral utilizando-se o atributo matiz*. Dissertação (Dissertação de Mestrado) — INPE, São José dos Campos, 2003.
- 17 LEÃO, A. C.; ARAÚJO, A. de A.; SOUZA, L. A. C. Implementação de sistema de gerenciamento de cores para imagens digitais. In: TEIXEIRA, A. C.; BARRÉRE, E.; ABRÃO, I. C. (Ed.). *Web e multimídia: desafios e soluções*. [S.l.]: PUC Minas, 2005.
- 18 COSTA, A. H. R.; PEGORARO, R. Construindo robôs autônomos para partidas de futebol: o time guaraná. *SBA Controle & Automação*, v. 11, n. 03, p. 141–149, 2000.
- 19 KITANO, H. et al. Robocup: A challenge problem for ai. *AI magazine*, v. 18, n. 1, p. 73, 1997.
- 20 CULJAK, I. et al. A brief introduction to opencv. In: *MIPRO, 2012 Proceedings of the 35th International Convention*. [S.l.: s.n.], 2012. p. 1725–1730.
- 21 STROUSTRUP, B. A history of c++: 1979–1991. In: BERGIN, T. J.; GIBSON, R. G. (Ed.). *History of programming languages — II*. [S.l.]: Addison-Wesley, 1996.
- 22 ITSEEZ. *OpenCV*. Acessado 29/06/2016 16:53. Disponível em: <<http://docs.opencv.org/3.1.0/>>.