TRABALHO DE GRADUAÇÃO

Identificação de features de campo com OpenCV para uso em robô NAO

Samuel Sousa Almeida

Brasília, Dezembro de 2020

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

Identificação de features de campo com OpenCV para uso em robô NAO

Samuel Sousa Almeida

Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação

Banca Examinadora

Profª. Mariana Bernardes, FGA/UnB _____
Orientadora

Profª. Cláudia Patrícia Ochoa Diaz, FGA/UnB _____
Examinadora interna

Prof. Geovany Araujo Borges, ENE/UnB _____
Examinador interno

Brasília, Dezembro de 2020

FICHA CATALOGRÁFICA

ALMEIDA, SAMUEL SOUSA
Identificação de features de campo com OpenCV para uso em robô NAO.

[Distrito Federal] 2020.

vii, 46p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2015). Trabalho de Graduação – Universidade de Brasília.Faculdade de Tecnologia.

1. Visão Computacional 2.OpenCV
3. Features 4. NAO

I. Mecatrônica/FT/UnB II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

S. ALMEIDA, SAMUEL (2020). Identificação de features de campo com OpenCV para uso em robô NAO. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$022, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 101p.

CESSÃO DE DIREITOS

Agradecimentos

Agradeço aos meus pais, Nilson e Maria de Jesus, pelo apoio desde o princípio, pela educação que me deram, pelo o incentivo de me fazer buscar sempre pelas melhores oportunidades. Agradeço à minha irmã Nathália por ser minha grande amiga nas melhores e piores horas em todos os momentos da minha vida. Gostaria de agradecer à toda a minha imensa família, aos meus tios, tias, primos e primas que tanto amo e que se eu fosse citar todos, levaria uma seção inteira. Agradeço aos meus amigos, Daniele, Filipe, Gabriel, Lorena, Lucas Ito, Mykael, Nonato e Scarllet pelo apoio moral, pelo interesse sempre demonstrado e por momentos de descontração proporcionados.
Equipe acadêmica que me deu suporte e incentivo em outras áreas de pesquisa acadêmica merecem uma menção nos meus agradecimentos, Valerio Martins, Rafael Timóteo de Sousa Júnior e Daniel Alves da Silva. Agradeço imensamente a minha orientadora Mariana Bernardes pela confiança e boa vontade em me aceitar como seu orientando. Agradeço também à todos os membros da equipe UnBeatables que sempre foram muito prestativos e atenciosos comigo durante a execução do trabalho.
Um agradecimento especial para Débora Ferreira dos Santos, que me acompanhou durante todo o trabalho, rodou testes comigo, avaliou os resultados, deu sugestões e me deu força e incentivo para continuar progredindo na elaboração desse estudo.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Samuel Sousa Almeida

RESUMO

As competições de futebol de robôs trazem diversos desafios, dentre eles o reconhecimento de características do campo. Utilizando imagens a partir do ponto de vista do robô NAO, amplamente utilizado nessas competições, a intenção desse trabalho é desenvolver um algoritmo que permita a identificação de *features* de campo para que possam ser utilizadas na melhoria da estratégia de jogo e refinamento da localização do robô em campo. Com uma metodologia definida e um processo de validação dessas *features*, obteve-se resultados promissores para as imagens simuladas. Para as imagens reais, no entanto, são necessários métodos mais robustos e avançados para a melhoria da taxa de acerto.

Palavras Chave: visão computacional, OpenCV, features, NAO.

ABSTRACT

Robot soccer competitions present several challenges, including the recognition of specific characteristics of the field. Using images from NAO robot's point of view, widely used in these competitions, the goal of this work is developing an algorithm that allows field features identification so that they can be used to improve game strategy and refine methods of robot's self localization in the field. With a defined methodology and a feature validation process, the results were promising to simulation images. To real images, more robust methods are required to increase success rate.

Keywords: computer vision, OpenCV, features, NAO.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Siglas

FIFA	Federação Internacional de Futebol
SPL	*Standard Platform League*
ISO	*International Organization for Standardization*
BRA	*British Robot Association*
JIS	*Japanese Industrial Standards*
IA	Inteligência Artificial
RGB	*Red, Green and Blue*
HSV	*Hue, Saturation and Value*
V-REP	*Virtual Robot Experimentation Platform*

Capítulo 1

Introdução

Este capítulo traz uma contextualização dos temas a que se refere o trabalho, a definição do problema que será estudado, o objetivo desta pesquisa, um resumo dos resultados que foram obtidos e a estrutura do trabalho.

1.1 Contextualização

Um robô é um sistema capaz de interagir com o ambiente por meio de seus sensores e atuadores [11]. As pesquisas em robótica cresceram rapidamente e, ao longo do tempo, as áreas de desenvolvimento também foram se diversificando. Além do setor industrial, berço da robótica, há atualmente aplicações nos segmentos educacional, médico, militar e até mesmo aplicações para soluções corriqueiras como no caso de robôs domésticos.

Os robôs industriais surgidos na década de 1960, inicialmente hidráulicos ou pneumáticos, evoluíram para as máquinas com controle por comando numérico [12] e depois para equipamentos eletrônicos avançados, como os da Figura 1.1. Com o tempo, dominaram as linhas de produção, sendo utilizados na fabricação de automóveis, telefones, eletrodomésticos e afins.

Além disso, a utilização de robôs no contexto educacional já é uma realidade. São exemplos dessa aplicação o *Robovie* e o popular LEGO *Mindstorms*, utilizados com crianças para ensino de linguagem e tecnologia, e o robô NAO da *Softbank Robotics* (antiga *Aldebaran Robotics*), utilizado para familiarizar estudantes universitários aos conceitos e aplicações de visão computacional [13, 14, 15]. O robô LEGO *Mindstorms* é mostrado na Figura 1.2.

Atualmente, existem também robôs que auxiliam em cirurgias médicas, sendo utilizados em variadas áreas da medicina, incluindo cirurgias cerebrais, uma vez que requerem altos níveis de exatidão e precisão que tal dispositivo é capaz de oferecer [3]. É possível ver a utilização de um sistema robótico sendo utilizado em uma cirurgia na Figura 1.3.

Todas essas áreas em que a robótica vem sendo inserida têm em comum a interação com um ambiente complexo e dinâmico. Portanto, a forma como os robôs extraem as informações do ambiente é muito importante para que a tecnologia seja bem aproveitada. A utilização de

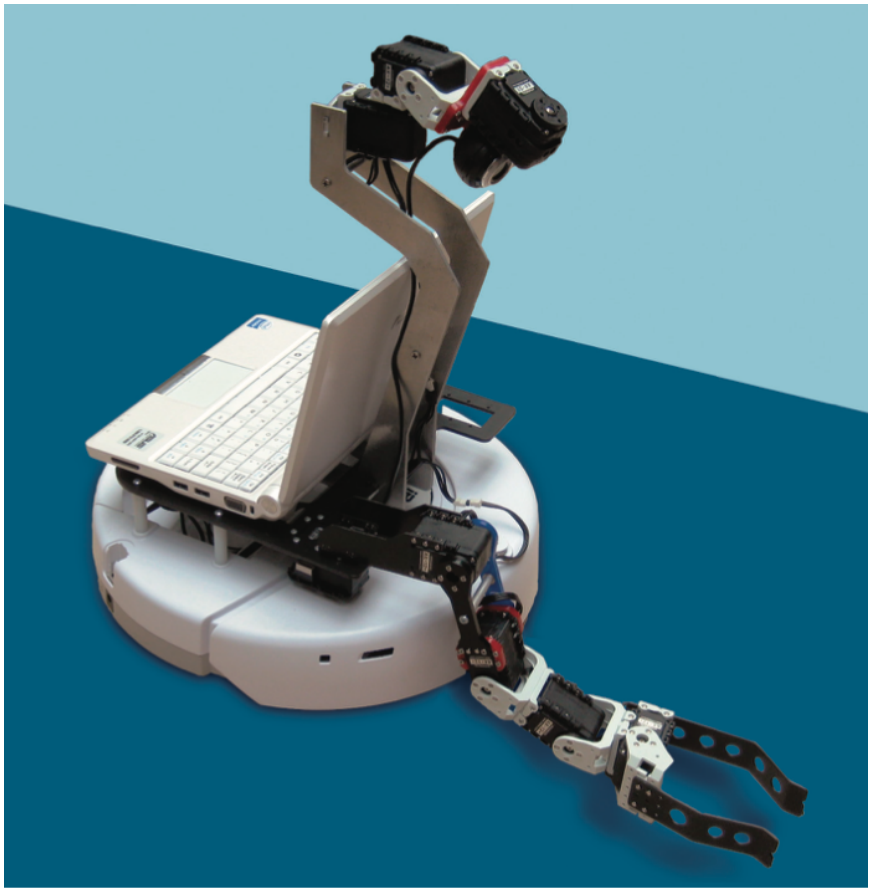


Figura 1.1: Protótipo de robô com braço robótico. Fonte[1]

Figura 1.2: Robô LEGO *Mindstorms*. Fonte[2]

sensores relevantes para cada aplicação individual é a forma que o dispositivo apreende dados para a compreensão do espaço em que está inserido. A diversidade de sensores disponíveis para aplicações robóticas é altíssima, podendo mensurar pressão, temperatura, aceleração, rotação, níveis de som e de luz.

Figura 1.3: Robô assistente sendo utilizado em cirurgia. Fonte [3]

Entre os sensores mais utilizados estão as câmeras, pois permitem a extração de uma grande quantidade de informações do ambiente. Porém, elas exigem um tratamento mais complexo para que sejam extraídos elementos realmente úteis à aplicação [3]. Isso significa que não basta a entrada do dado da câmera, ele precisa ser processado para que gere de fato uma informação de qualidade, que tenha valor e possa ser utilizada. A inteligência artificial (IA) foi alicerce para o desenvolvimento dos primeiros algoritmos de extração de informações de imagens [3]. A visão computacional surgiu como um campo científico específico para o desenvolvimento de tais algoritmos. Várias das técnicas que surgiram inicialmente com a IA, atualmente, estão bem consolidadas e já não são mais consideradas como IA [3].

Com a finalidade de incentivar a elaboração de pesquisa e desenvolvimento em áreas como robótica e inteligência artificial, surgiu em 1997 a iniciativa *RoboCup*, cuja proposta consiste em ter, em 2050, um time de robôs que, no futebol, vença o time campeão da Copa Mundial da Federação Internacional de Futebol (FIFA) daquele ano. Embora o uso do futebol não pareça ter grande impacto, completar o desafio significaria um marco importante para a área e para a humanidade. Para atingir a meta definida até 2050, há anualmente a formulação de novos desafios na área acadêmica. A mudança das regras da competição faz com que o jogo de futebol de robôs se assemelhe cada vez mais ao tradicional [4].

A complexidade do jogo de futebol traz muitos atrativos para o desenvolvimento científico no âmbito da competição e é inclusive comparada com o jogo de xadrez desde a concepção da *RoboCup*, em razão da existência de um problema padronizado com regras bem definidas, permitindo trabalhos de pesquisa focados. Frente ao xadrez, existem características do jogo de futebol que fazem muita diferença para o desenvolvimento de tecnologias neste escopo, como o ambiente bem mais dinâmico, o acesso limitado a informações e o controle distribuído entre vários agentes [16]. Esses fatores tornam o desafio ainda maior, uma vez que abarca várias áreas nas quais a pesquisa deve acontecer a fim de que o objetivo final seja atingido.

Com o crescimento de outras áreas de aplicação da robótica, o evento se estendeu para outros desafios dentro dos domínios *RoboCup@Home*, *RoboCupIndustrial*, *RoboCup Junior* e *RoboCupRescue*. Esses outros domínios focam, respectivamente, em aplicações domésticas, industriais, educacionais e de busca e resgate. Embora não tenham relação direta com a meta de 2050, eles também são importantes no incentivo ao desenvolvimento de pesquisa em tecnologia e inteligência artificial.

Ativa desde 2000, a *RoboCup Rescue* é uma competição que surgiu dentro do contexto de desastres naturais que assolaram o Japão na década de 1990 [17]. Seus objetivos envolvem aumentar o conhecimento acerca das possíveis variáveis em situações de busca e resgate, diminuir os riscos a que são expostos os profissionais de resgate, além de promover pesquisa e desenvolvimento na área, que tende a ter muita atuação robótica futuramente. As duas ligas que compõem a competição e guiam os participantes em direção ao objetivo são a liga robótica e a liga de simulação. Na liga *Robot*, existem robôs físicos que devem procurar por vítimas em ambientes experimentais, projetados para simular catástrofe. Mobilidade, sensoriamento e mapeamento são objetivos da programação desses robôs [17]. Na liga *Simulation*, o ambiente e o desenvolvimento das situações de desastre, como sugere seu nome, são completamente simulados para que a programação de agentes robóticos possa ser utilizada eventualmente para atuação em situações reais.

A RoboCup@Home e RoboCup Industrial recorrem ao uso de robôs nos contextos doméstico[1] e industrial[2], respectivamente, e a RoboCup Junior tem seu foco em futebol de robôs para estudantes de nível fundamental e médio e com idades entre 13 e 19 anos[3][4].

A *RoboCup Soccer* não apenas deu origem à iniciativa *RoboCup*, mas é também sua competição principal. Desde a sua formulação, a existência de ligas já era uma realidade e elas evoluem conforme a tecnologia existente e o estado do desenvolvimento da competição [16]. Elas são voltadas para a concretização do objetivo definido pela iniciativa no momento de sua criação e estão ilustradas na Figura 1.4. Atualmente, existem cinco ligas na *RoboCup Soccer*[4]:

- *Humanoid*: A liga é exibida na Figura 1.4(a) e é composta por robôs autônomos com braços e pernas em proporções definidas para que correspondam com o aspecto humanoide. Os robôs competem entre si em times de tamanhos variados, dependendo da subdivisão da liga. A categoria *KidSize* tem times de quatro jogadores, a *TeenSize* tem dois jogadores por time e a *AdultSize* tem apenas um jogador contra o outro. Desafios técnicos da liga envolvem a movimentação do robô e a manutenção do equilíbrio durante o chute.

- *Middle Size*: Nessa liga, os robôs jogam com uma bola da FIFA de tamanho regular, como é visível na Figura 1.4(b). Até ano passado, os times da liga eram formados por cinco jogadores, mas as regras atualizadas de 2020 trazem times de onze jogadores [18]. Algumas especificações físicas são feitas para cada membro do time, mas é uma liga relativamente flexível com o projeto de *hardware*, dando liberdade à equipe livre para desenvolver seus robôs.

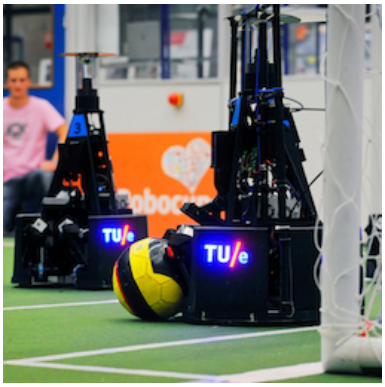
[1] https://athome.robocup.org/
[2] https://atwork.robocup.org
[3] http://junior.robocup.org/

- *Small Size*: Liga focada em sistemas multiagentes e na colaboração entre esses agentes em um ambiente dinâmico. Os times têm oito jogadores cujas dimensões são limitadas pelas regras. As informações do campo são captadas por um sistema de visão padrão comum a todos os participantes, localizado acima do campo. Elas são então repassadas para os times, que são coordenados e controlados por um computador central de cada equipe. Robôs utilizados pela liga são mostrados na Figura 1.4(c).

- *Standard Platform*: Nesta liga, exibida na Figura 1.4(d), todos os robôs usados são iguais, o NAO da *SoftBank Robotics*. Embora seja permitida a comunicação entre os robôs, a tomada de decisão deve ser individual. O ambiente do jogo é bem definido pelo comitê da competição e evolui anualmente, ficando cada vez mais similar a um campo de futebol tradicional e trazendo desafios que incentivam a pesquisa em visão computacional, comunicação e inteligência artificial.

- *Simulation*: Liga que não possui robôs físicos e permite a pesquisa focada em inteligência artificial a fim de melhoria da estratégia de jogo. Existem subdivisões para simulações bi e tridimensionais.

(a) Liga *Humanoid*



(b) Liga *Middle Size*

(c) Liga *Small Size*

(d) Liga *Standard Platform*

Figura 1.4: Ligas da *RoboCup Soccer*. Fonte[4]

A *Standard Platform League* (*SPL*) definiu um robô padrão para todas as equipes desenvol-

vedoras a fim de garantir que todos partam da mesma base de *hardware*, deixando claro que o objetivo desta liga é o desenvolvimento de *software*. Jogando em times formados por cinco robôs, cada um deles deve ser capaz de tomar decisões individuais, embora possa haver comunicação entre eles. Conforme as regras mais atuais disponíveis, o campo de futebol é verde e tem linhas e traves brancas, além de apresentar também marcações similares às do futebol humano, apesar de terem uma única área delimitada próxima ao gol, em vez das duas comuns ao jogo convencional[19]. A bola de futebol é preta e branca, similar à do futebol tradicional. O robô escolhido para ser usado na *SPL* é o NAO, da *SoftBank Robotics*.

O NAO, mostrado na Figura 1.5, é um robô comercial que tem foco em pesquisa e educação. Embora ele seja amplamente utilizado no contexto da *RoboCup*, existem publicações acadêmicas que exploram seu potencial para outras aplicações, como as educacionais ou que dizem respeito à interação homem-máquina, por exemplo [20, 21]. Com uma estrutura humanoide, o NAO tem diversos sensores e atuadores que o permitem interagir bem com muitos ambientes, com outros robôs similares e com seres humanos. O dispositivo da *Softbank Robotics* foi escolhido para substituir a plataforma de estrutura quadrúpede *Sony Aibo*, exibido na Figura 1.6, promovendo o estudo com base em um objeto mais próximo da forma humana e, por conseguinte, mais próximo do contexto de um jogo real.

Entre os diversos sensores e atuadores que o robô possui, dentro do contexto do jogo de futebol o sensor de visão é extremamente relevante em razão de ser a forma de o agente encontrar a bola, os gols, os outros jogadores e outros elementos importantes para a estratégia do jogo. Dessa forma, a visão computacional é um campo de estudo chave no desenvolvimento de algoritmos de futebol de robôs. Fazer com que o robô tenha as informações de elementos fixos do campo pode levar a um melhor desempenho durante a partida. Tais informações podem servir de parâmetro para definição da localização do robô dentro do campo durante o jogo e, portanto, melhorar a estratégia de jogo.

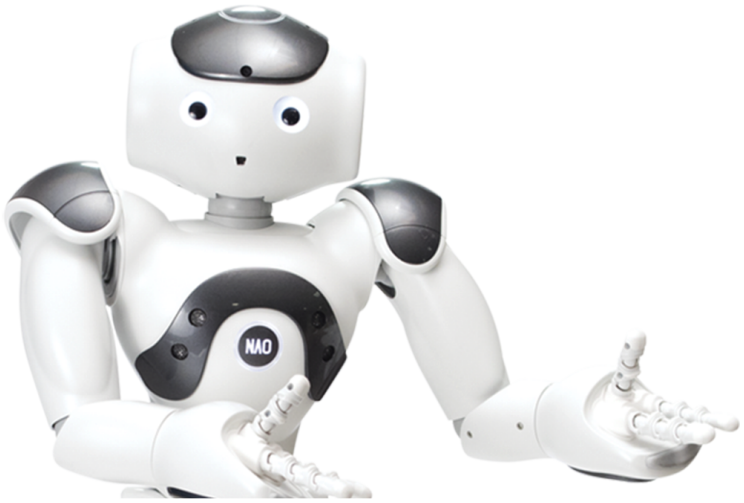


Figura 1.5: Robô NAO da *Softbank Robotics*. Fonte[5]

Figura 1.6: Robô AIBO da *Sony*.Fonte[6]

1.2 Definição do Problema

Este trabalho é desenvolvido com base na plataforma NAO, no contexto de uma partida de futebol da *RoboCup SPL*. Para cumprir com as tarefas de um jogo de futebol, é importante que o robô seja capaz de obter dados do ambiente. Com a câmera, pode ser feita a obtenção de sinais e, com o seu correto processamento utilizando técnicas de visão computacional, pode-se fazer a identificação de *features* na imagem para que realmente se obtenha uma informação útil relativa ao ambiente.

Esse trabalho foca no desenvolvimento e aplicação de algoritmos de visão computacional para a extração de informações importantes do campo de futebol, como mostrado na Figura 1.7. As demarcações em vermelho são as características que se deseja encontrar nas imagens que serão analisadas. Denominadas L, T e X, elas se caracterizam pelo encontro de duas linhas. A *feature* L é definida pelo encontro das extremidades de duas linhas. A *feature* T é definida pelo encontro da extremidade de uma linha com a parte intermediária de outra linha. Já a *feature* X é definida pelo encontro de partes intermediárias de duas linhas.

1.3 Objetivo Geral

O objetivo geral do trabalho é desenvolver um algoritmo para que o robô NAO possa identificar com sucesso elementos do campo e permitir que outros módulos utilizem essa informação para o desenvolvimento de estratégias de jogo.

1.4 Resultados Obtidos

Os resultados obtidos para esse trabalho são promissores, chegando a resultados com 62% de acerto. Em conjuntos de imagens obtidos externamente, a taxa de acerto tem espaço de melhora

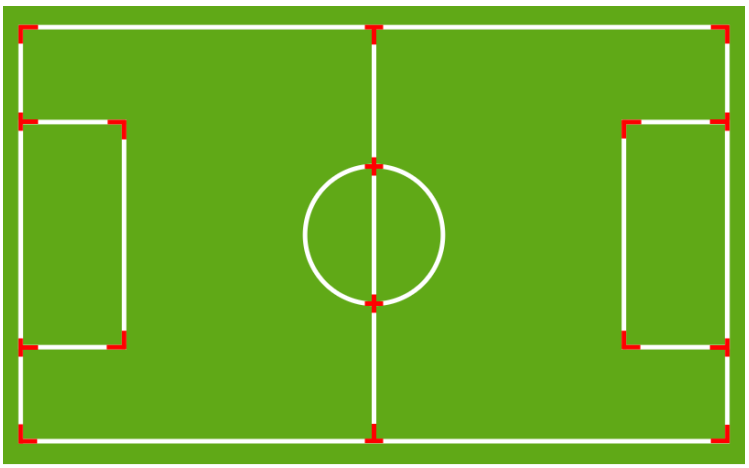


Figura 1.7: Campo de futebol SPL destacando *features* que são alvo da detecção. Fonte: Autor

com o uso de técnicas de interpolação, que tem potencial de melhorar a detecção de elementos nas imagens.

1.5 Estrutura do Trabalho

O trabalho que se segue é dividido em Fundamentos, Desenvolvimento, Resultados e Conclusões. No capítulo 2, de Fundamentos, alguns conceitos relativos a robótica e visão computacional são apresentados . No Capítulo 3, de Desenvolvimento, há uma explicação da metodologia aplicada para a realização do trabalho e para o desenvolvimento do código utilizado. No Capítulo 4, de Resultados, são apresentados os produtos dos experimentos, além de uma discussão acerca deles. No Capítulo 5, de Conclusões, um resumo do trabalho é feito, junto com comentários finais sobre os resultados obtidos, além da sugestões para possíveis trabalhos futuros.

Capítulo 2

Fundamentos

Este capítulo apresenta conceitos fundamentais para o desenvolvimento deste trabalho. A Seção 2.1 contém algumas definições de robótica e as especificações do robô humanoide utilizado para a pesquisa. Na Seção 2.2, os conceitos relativos a Visão Computacional são explicados e exemplificados. Finalmente, a Seção **??** apresenta algumas definições que concernem a sistemas em tempo real, que são escopo do trabalho apresentado.

2.1 Robótica

2.1.1 Definições

O termo robótica, utilizado na última década de forma bastante deliberada, tem sua origem etimológica datada há quase um século. O termo tcheco que remete a trabalho forçado foi utilizado em uma peça de teatro, com relação a máquinas humanoides que executavam trabalhos humanos [22, 3]. A representação constante de robôs como seres humanoides, especialmente em obras de ficção científica, ajuda a popularizar e mistificar o termo e seu significado.

O conceito, originado antes mesmo da concepção do primeiro computador em 1946, só foi bem consolidado na década de 1980 quando definições foram dadas pela *International Organization for Standardization* (ISO), pela *British Robot Association* (BRA) e pela *Japanese Industrial Standards* (JIS)[1]. A BRA define um robô como um "dispositivo reprogramável feito para manipular e transportar partes, ferramentas ou instrumentos especializados de manufatura usando movimentos programados variáveis para a performance de tarefas industriais específicas", em tradução livre [23]. A JIS tem uma definição distinta, que defende que um robô é um "sistema mecânico com movimentação flexível análoga à de organismos vivos ou que combina a movimentação com funções inteligentes que respondem ao desejo humano", em tradução livre [24]. É possível notar que essas definições surgiram dentro do escopo de robôs industriais e atualmente existem conceitos mais abrangentes, até mesmo pelo aumento das áreas de aplicação da robótica.

[1] ABREU, P. Robótica Industrial. Acesso em 09 de Março de 2020. Disponível em <`paginas.fe.up.pt/~aml/maic_files/introd.pdf`>

É possível definir um robô como um sistema físico e autônomo que pode não apenas reconhecer o próprio ambiente, mas também agir sobre ele a fim de atingir determinados objetivos [3]. A ISO 8373[2] traz uma especificidade ainda maior a essa definição e diz que um robô é um mecanismo programável que atua em no mínimo dois eixos, tendo grau de autonomia e mobilidade dentro de seu ambiente a fim de completar tarefas. Essas duas definições trazem uma visão que não é apenas voltada a robôs industriais, mas também a robôs de serviço. Esse fato evidencia a evolução da robótica no cotidiano e não mais apenas como um artifício da ficção científica ou como um instrumento industrial voltado para a produtividade.

O desenvolvimento da robótica está fortemente relacionado a três campos de estudo: a teoria de controle, a cibernética e a inteligência artificial [3]. A teoria de controle consiste no estudo matemático das propriedades de sistemas automatizados. Essa é uma área de estudo que já estava bem consolidada quando a robótica começou a ser mais amplamente explorada. Já a cibernética é o estudo da aplicação em sistemas artificiais da comunicação e controle de processos de modelos biológicos [3]. Um aspecto importante para a cibernética é a interação do sistema com o ambiente, ponto fundamental da definição de um robô.

A inteligência artificial, por sua vez, é o estudo de tecnologias capazes de reconhecer o ambiente e recorrer a ações que aumentem as chances de sucesso de completar a tarefa para a qual foram desenvolvidas [25]. A inteligência artificial também definiu diversas abordagens dentro da robótica e problemas de resolução complexa tiveram suas soluções implementadas tendo como base algoritmos da inteligência artificial.

É importante definir a autonomia e a mobilidade. A primeira consiste no fato de o sistema ser capaz de tomar decisões sozinho [3]. Já a última baseia-se na capacidade de locomoção do robô. Relembrando e detalhando um pouco mais o histórico de desenvolvimento da robótica, os robôs industriais normalmente tinham suas plataformas fixas e consistiam em braços ou plataformas que objetivavam a manipulação de objetos. Com o tempo, robôs começaram a ter rodas, trilhos e até mesmo pernas. Embora seja possível dividir robôs em móveis e manipuladores, essa separação parece fazer cada vez menos sentido com o crescimento do número de robôs que são capazes de não só se locomover por um ambiente como também de manusear objetos [3].

A ampliação dessas aplicações abrem espaço para diversas áreas acadêmicas desenvolverem seus estudos. A programação de um robô para que ele seja capaz de se locomover por um ambiente, tomar as decisões pertinentes à sua aplicação e desempenhar sua tarefa com precisão é algo que exige muito esmero e estudo. Esta pesquisa baseia-se num robô humanoide para desenvolver tecnologias que auxiliem a sua tomada de decisão.

2.1.2 Especificações do NAO

As versões 4 e 5 do NAO, dispositivo da *Softbank Robotics*, têm um processador *Intel ATOM* Z530, enquanto a versão 6 do robô usa um processador *Intel ATOM* E3845[26, 7]. Dentro do contexto deste trabalho, o desenvolvimento se baseia no processamento de imagens captadas pelo

[2]https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:en

NAO. Dessa forma, a especificação técnica de suas câmeras é extremamente importante e as especificações de hardware de suas versões são apresentadas na Tabela 2.1.

Tabela 2.1: Especificações de hardware das versões v4 e v6 do NAO

		NAO v4 [26]	**NAO v6 [7]**
Placa-mãe	Processador	ATOM Z530	ATOM E3845
	Número de núcleos	1	4
	Número de *threads*	2	4
	Frequência Base (*GHz*)	1.60	1.91
	Cache	512 KB	2 MB
	RAM	1 GB	4 GB
Câmera	Modelo	MT9M114	OV5640
	Resolução (Mp)	1.22	5
	Imagem de saída	1280x960 (30fps)	640x480 (30fps) 2560x1920 (1fps)
	Campo de visão diagonal	72.6°	67.4°
	Campo de visão horizontal	60.9°	56.3°
	Campo de visão vertical	47.6°	43.7°

Composto de diversos sensores como unidades inerciais, sonares, sensores de posição de cada junta, além de sensores táteis e de contato, o NAO também possui infravermelho, LEDs e sensores de pressão *FSR* (*force-sensing resistor*)[26, 7]. As duas câmeras idênticas que o dispositivo possui são posicionadas no mesmo eixo vertical de sua cabeça, contrariando a impressão inicial de que as câmeras se posicionam no que parecem os olhos do robô humanoide. Na verdade, a posição das câmeras se dá conforme a Figura 2.1 e nas Figuras 2.2 e 2.3 podemos ver os campos de visão que o dispositivo pode enxergar, com os ângulos de campo de visão da versão 4 do NAO.

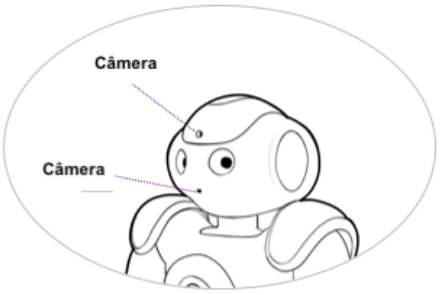


Figura 2.1: Posição das câmeras no robô NAO. Adaptado de [7]

2.2 Visão computacional

Visão computacional é um campo de estudo que envolve a extração de conhecimento através de imagens, sendo possível utilizá-lo para fazer alguma inferência acerca do mundo físico [27]. A gama de ferramentas utilizadas na visão computacional envolve o processamento de imagens,

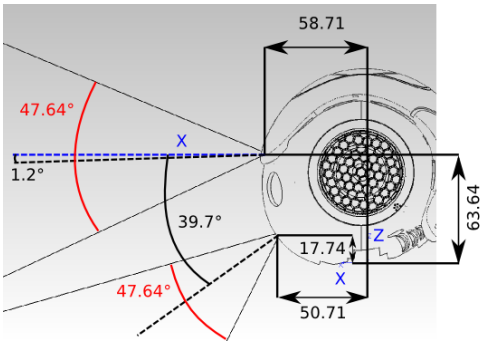


Figura 2.2: Visão lateral do campo visual da plataforma NAO. Fonte [8]

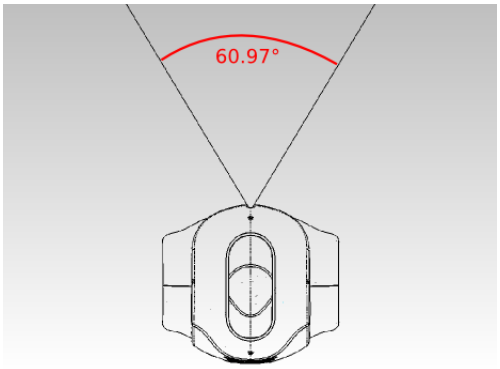


Figura 2.3: Visão superior do campo visual da plataforma NAO. Fonte[8]

o que normalmente significa transformá-las de forma que algum método computacional consiga extrair as informações de forma bem-sucedida. O processamento de imagens recebe uma imagem como entrada e tem uma outra imagem na saída. Ele envolve aplicação de técnicas, comumente já consolidadas, para transformar ou destacar determinada característica. A importância desse processo consiste na potencialização para aplicação de outros métodos. Já a visão computacional tem como entrada uma imagem e, como saída, uma informação que pode ser usada posteriormente para tomada de decisão. A câmera é um dos sensores com maior potencial de trazer muitas informações acerca do ambiente [28], mas essa informação precisa ser extraída da imagem capturada, papel que é desempenhado pelos algoritmos de visão computacional.

Dessa forma, a visão computacional pode ser vista como os olhos de um sistema computacional. É composta de métodos que podem ser utilizados para fazer um sistema robótico móvel desviar de um obstáculo ou levar um braço robótico a descartar uma peça defeituosa em uma linha de produção, por exemplo. Esse procedimento acontece com a detecção de *features*, que são elementos únicos que caracterizam parte de um objeto e, portanto, carregam informação [29].

Aplicações comuns da visão computacional são identificação de rostos humanos, análise de exames médicos, detecção de patologias e análise de peças em linhas de produção. De forma análoga ao processo em sistemas biológicos, a informação de que a peça na linha de produção tem um defeito pode ser obtida pela visão, mas o seu descarte depende da tomada da decisão e da efetiva retirada do item. Dessa forma, a visão computacional é raramente utilizada sozinha, sendo

comumente associada a outros campos de estudo, como a inteligência artificial, que seria capaz de tomar a decisão e mandar sinais de controle para atuadores específicos do robô.

Identificar elementos visuais através das cores é do instinto humano e é um método robusto quando se considera a movimentação de objetos, já que a cor não muda conforme movimentos de rotação ou translação [30, 31]. No entanto, computacionalmente falando, o uso de cores não é trivial [32] e o desafio de representação de cores no computador começa pela necessidade de representação digital das cores. A fim de padronizar e facilitar tal representação, têm-se os modelos de cores, amplamente conhecidos e aceitos [31]. O uso de determinado modelo depende da aplicação desenvolvida e pode influenciar diretamente em sua performance, e os dois mais comumente aplicados à visão computacional são o RGB e o HSV.

2.2.1 Modelos de cores

2.2.1.1 RGB

O modelo RGB, que remete às cores vermelho, verde e azul (*red, green* e *blue*), descreve cada cor existente como uma combinação dessas três. Computacionalmente falando, cada cor pode variar seu tom de vermelho, verde e azul entre 0 e 255, o que gera uma possibilidade de mais de 16 milhões de cores existentes.

Essa representação remete à notação de cores HTML, que é composta por seis dígitos de 0 a F no sistema hexadecimal: cada par de dígitos representa um número de 0 a 255 para uma das três cores. A manipulação e ajuste de cores nas imagens RGB é concentrada na variação dos valores de três parâmetros entre 0 e 255, representando a intensidade de cada uma das três cores. O modelo RGB pode ser visualizado conceitualmente no formato de cubo, conforme a Figura 2.4, em que observa-se que as cores vermelha, verde e azul são vértices equidistantes desse cubo. Em outros dois vértices, encontram-se o preto e o branco (oculto na imagem) que indicam, respectivamente, a ausência e a presença das três cores principais. Nos vértices restantes, encontram-se cores que são compostas por cada uma das combinações de pares entre as três primárias.

2.2.1.2 Modelo de cores HSV

O modelo HSV referencia *hue, saturation* e *value*, que em português são, respectivamente, tonalidade, saturação e valor (de brilho) de uma determinada cor. O valor do primeiro parâmetro pode variar de 0 a 360 enquanto os outros dois são representados em porcentagem. O modelo HSV é bastante utilizado na visão computacional, especialmente aplicada à robótica, principalmente porque seu componente de brilho torna essa representação extremamente robusta no que tange à determinação e diferenciação de cores [33]. Essa característica de robustez do modelo é bastante interessante e é aproveitada no contexto do presente trabalho, em que condições de iluminação variam bastante. O uso do espaço HSV permite que o algoritmo continue achando os elementos pertinentes, sendo menos dependente a uma condição constante e boa de iluminação.

Analogamente ao modelo RGB, que é representado por um cubo, o modelo HSV também pode

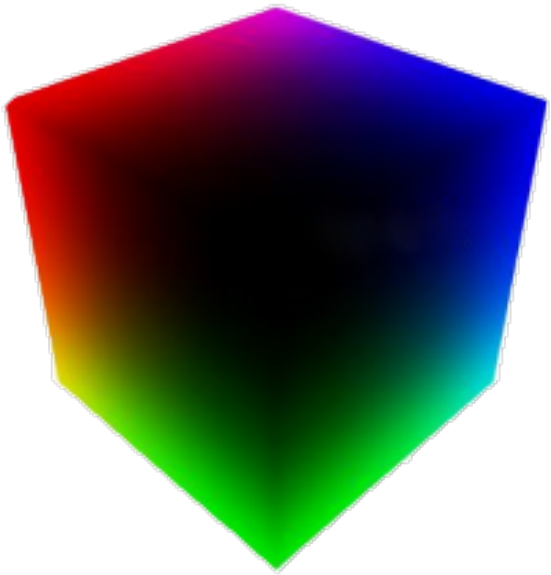


Figura 2.4: Cubo de cores RGB com vértice branco oculto. Fonte: Adaptado de [9]

ser representado por uma figura geométrica tridimensional na forma de cone. É possível vê-lo na Figura 2.5. A tonalidade (H) acompanha o perímetro das seções circulares do cone, por isso seu valor varia de 0 a 360, que representa um ângulo. A saturação (S) é definida pela distância desde o centro da seção circular. Por último, o valor de brilho (V) é representado pela altura do cone.

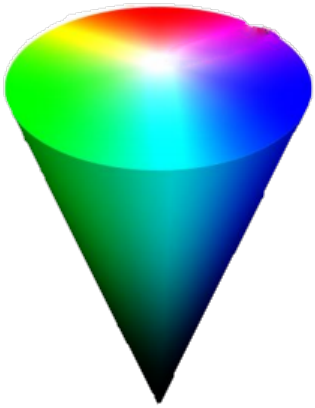


Figura 2.5: Cone de cores HSV. Fonte: Adaptado de [9]

2.2.1.3 Conversão de modelos de cores

É necessário fazer uma conversão constante entre os modelos RGB e HSV. Os três parâmetros H, S e V são mapeados a partir do R, G, B conforme um modelo matemático descrito conforme as equações 2.1, 2.2 e 2.3 [30].

$$H = \begin{cases} 0 & se \quad max(R,G,B) = min(R,G,B) \\ 60 \cdot \frac{G-B}{max(R,G,B)-min(R,G,B)} & se \quad max(R,G,B) = ReG \geq B \\ 60 \cdot \frac{G-B}{max(R,G,B)-min(R,G,B)} + 360 & se \quad max(R,G,B) = ReG < B \\ 60 \cdot \frac{B-R}{max(R,G,B)-min(R,G,B)} + 120 & se \quad max(R,G,B) = G \\ 60 \cdot \frac{R-G}{max(R,G,B)-min(R,G,B)} + 240 & se \quad max(R,G,B) = G \end{cases} \tag{2.1}$$

$$S = \begin{cases} 0 & se \quad max(R,G,B) = 0 \\ \frac{max(R,G,B)-min(R,G,B)}{max(R,G,B)} & se \quad max(R,G,B) \neq 0 \end{cases} \tag{2.2}$$

$$V = max(R,G,B). \tag{2.3}$$

2.2.2 OpenCV

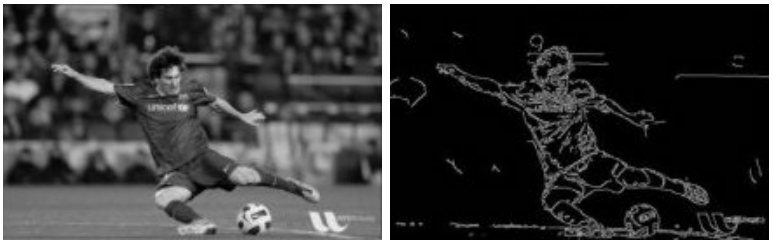
OpenCV[3] é uma biblioteca de visão computacional *open-source* composta por diversos métodos. Nativamente do C++, atualmente está disponível também para as linguagens Java e *Python*, podendo também ser utilizada com o MATLAB. Possui mais de 2500 algoritmos que podem ser usados para uma grande variedade de aplicações[3]. Com uma larga comunidade, é a ferramenta ideal para o desenvolvimento do presente trabalho, que utiliza o *OpenCV* aliado ao *Python*.

O *OpenCV* tem funções e métodos que facilitam a manipulação de imagens, auxiliam na detecção de linhas e na conversão das cores. Amplamente utilizada dentro do contexto da *SPL*, a biblioteca é um dos alicerces deste trabalho e de muitos outros. Ela é compatível com Linux, Windows e MacOS. Com grande foco em aplicações em tempo real, o OpenCV tem grande potencial de aproveitamento em processadores com múltiplos núcleos, além de ter módulos auxiliares para a área de aprendizado de máquina [34]. Essa característica é interessante porque permite escalabilidade no desenvolvimento de algoritmos, deixando-os adaptáveis para futuros módulos e funções que envolvam áreas de estudo correlatas. De origem russa, a biblioteca começou a ser projetada em 1999 e teve seu lançamento oficial em 2006[34]. Atualmente em sua segunda versão lançada, já contém mais de 2 mil algoritmos dentre os quais estão funções que auxiliam na segmentação de cores, detecção de elementos geométricos, conversão entre espaços de cores, além de diversas outras funções que são úteis no contexto do trabalho.

Funções valiosas do *OpenCV* são as de transformação morfológica, normalmente executadas em imagens binárias, cujos pixels são 1 ou 0. A erosão é uma operação capaz de alterar a aparência dos objetos em primeiro plano na imagem por meio da execução de varredura em que cada pixel de valor 1 é mantido apenas se todos ao seu redor tiverem o mesmo valor promovendo o desgaste do elemento em primeiro plano [35]. A dilatação consiste em uma transformação que, de forma análoga, executa uma varredura em que a condição para manutenção de um pixel de valor unitário é a existência de qualquer outro nas proximidades que também tenha valor 1 proporcionando o aumento da área do elemento em primeiro plano [35].

[3] http://opencv.org/

As funções de detecção de bordas e de aproximação de linhas, respectivamente denominadas *canny* e *houghLines*, são comumente usadas em conjunto. A primeira, nomeada em homenagem ao seu criador, executa quatro estágios com o objetivo de obter as bordas mais fortes na imagem. Essa operação é ilustrada na Figura 2.6. A segunda, a partir da imagem binária, escaneia e pontua possíveis retas presentes e conforme elas atinjam determinada pontuação usada como parâmetro do algoritmo, são retornadas para o corpo principal do programa usando parâmetros matemáticos que representam as linhas geometricamente [36].



(a) Imagem original

(b) Imagem com detecção de bordas

Figura 2.6: Imagens ilustrativas da função de detecção de bordas *canny*. Fonte: Adaptado[10]

2.2.3 Representação de linhas

Na visão computacional, é comum a representação de elementos geométricos por meio de seus parâmetros algébricos, como na geometria analítica. Segmentos de reta podem ser representados pelas coordenadas de seus pontos de início e fim. No entanto, nem sempre é utilizada a notação cartesiana, tão amplamente conhecida. Especialmente no que diz respeito a retas, o OpenCV por meio da sua função de detecção as representa usando a forma normal de *Hessian* [36].

Normalmente uma linha pode ser representada por sua forma reduzida, como na equação $y = a \cdot x + b$. No entanto, a forma normal de *Hessian* define uma linha utilizando outros dois parâmetros, ρ e θ. Eles representam, respectivamente, a menor distância entre a reta e a origem e o ângulo que a reta faz com o eixo y[36, 37]. Esses parâmetros podem ser visualizados na Figura 2.7.

Os parâmetros da forma reduzida da reta, a e b, são a cotangente do ângulo α e o ponto de cruzamento da reta com o eixo y, nessa ordem. O ângulo α e θ são complementares, isto é, somam 90^{o}. É possível, a partir da imagem, estabelecer relações diretas entre os pares de parâmetros (a, b) e (ρ, θ):

$$\begin{cases} a = cot(\theta) \\ b = \rho \cdot csc(\theta). \end{cases} \tag{2.4}$$

Para determinar se duas retas, r_1 e r_2, se cruzam, é necessária a solução para x e y do sistema de equações

$$\begin{cases} r_1 : y = a_1 \cdot x + b_1 \\ r_2 : y = a_2 \cdot x + b_2. \end{cases}$$

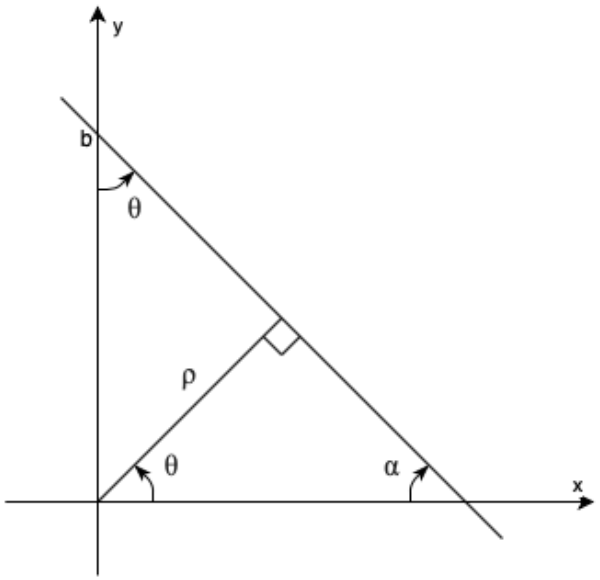


Figura 2.7: Representação de uma reta exibindo os parâmetros de Hessian. Fonte: Autor

O sistema pode ser reorganizado conforme

$$\begin{cases} -a_1 \cdot x + y = b_1 \\ -a_2 \cdot x + y = b_2 \end{cases}$$

para facilitar a representação por matrizes. O sistema, no seu formato de matrizes, é dado por

$$\begin{bmatrix} -a_1 & 1 \\ -a_2 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}.$$

Para a obtenção da solução, é necessário obter a inversa da matriz de coeficientes, obtida pelo produto do seu determinante pela matriz adjunta, que por sua vez é definida pela transposta da matriz de cofatores. O sistema, então, é desenvolvido como

$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{1}{a_2 - a_1} \cdot \begin{bmatrix} 1 & -1 \\ a_2 & -a_1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

e, em seguida, desenvolvendo as duas matrizes mais à direita, encontra-se a relação

$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{1}{a_2 - a_1} \cdot \begin{bmatrix} b_1 - b_2 \\ a_2 b_1 - a_1 b_2 \end{bmatrix}.$$

.

Com a matriz nesse formato, é possível obter as coordenadas (x, y) da interseção entre duas retas representadas na forma Hessiana se as substituições da equação 2.4 forem aplicadas no sistema, resultando em

$$\begin{cases} x = \frac{\rho_1 \cdot csc(\theta_1) - \rho 2 \cdot csc(\theta_2)}{cot(\theta_2) - cot(\theta_1)} \\ y = \frac{cot(\theta_2) \cdot \rho_1 \cdot csc(\theta_1) - cot(\theta_1) \cdot \rho_2 \cdot csc(\theta_2)}{cot(\theta_2) - cot(\theta_1)}. \end{cases} \tag{2.5}$$

.

Capítulo 3

Desenvolvimento

Este capítulo apresenta a metodologia de trabalho realizada nos ambientes competentes a este estudo, incluindo a ordem em que operações foram aplicadas nas imagens e os ajustes de parâmetros que foram executados a fim de aumentar a acurácia de detecção das *features*.

3.1 Obtenção de imagens

Como explicado no Capítulo 2, algoritmos de visão computacional têm imagens como entrada e dados como saída. Dessa forma, é necessária ao trabalho a coleta de imagens pertinentes ao seu escopo, o jogo de futebol de robôs. Idealmente, o desenvolvimento desta pesquisa se daria com o auxílio de um dispositivo NAO e um campo de futebol dentro dos padrões definidos pelo comitê da *RoboCup* para a *SPL*. O algoritmo teria como entrada as imagens coletadas pelo robô em tempo real e salvaria os resultados em estruturas de dados que facilitam a integração entre outros módulos de jogabilidade. A execução da pesquisa com o NAO em campo permitiria também a análise de desempenho do código.

No entanto, com jogos suspensos e sem acesso a um campo de futebol, uma solução alternativa foi concebida para o desenvolvimento do trabalho. O desenvolvimento de código pode ser feito usando arquivos de imagens preexistentes. No entanto, a princípio não foi encontrado nenhum conjunto de imagens, o que levou ao trabalho a ser iniciado usando imagens de simulação combinadas das câmeras superior e inferior do robô.

A equipe de competição *UnBeatables*, usando um programa de simulação denominado *V-REP*, havia projetado um campo de futebol seguindo as dimensões apropriadas e programado um robô NAO virtual para navegar por esse campo capturando imagens diversas com a perspectiva de suas duas câmeras. O *V-REP, Virtual Robot Experimentation Platform*, encontra-se atualmente descontinuado pela sua empresa desenvolvedora, *Coppelia Robotics*, que traz um novo programa para a simulação, denominado *CoppeliaSim*.

Ainda assim, as imagens obtidas pela equipe utilizando o V-REP foram de grande valia para a execução desta pesquisa. Analisando manualmente cada uma das 770 imagens correspondentes

à simulação, foram construídos dois arquivos de texto para serem usados como gabarito. Um dos arquivos possui com a quantidade de linhas presentes em cada imagem e o outro é composto pela localização e tipo das *features*, também discriminado por imagem. A elaboração deste gabarito serve não apenas para a avaliação da efetividade do algoritmo ao fim do trabalho, mas também para estabelecimento de um norte na definição dos parâmetros-chave do algoritmo.

O uso exclusivo de imagens simuladas não permitiria uma avaliação da efetividade do código a fim de determinar sua usabilidade de fato no campo. A presença de elementos fora do limite do campo, a iluminação variável e a diferença de textura no gramado são exemplos de elementos que existem nas imagens reais e que podem interferir na detecção e identificação de *features* pelo algoritmo. A ausência de um banco de imagens de jogos anteriores levou à procura de uma solução que fornecesse insumos para o algoritmo.

A plataforma *ImageTagger* permitiu a obtenção de um set de imagens bastante completo, mas com capturas de apenas uma câmera e sem informações detalhadas de qual robô foi utilizado para a captura. A ferramenta gratuita e *open source ImageTagger* foi criada por pesquisadores da Universidade de Hamburgo justamente com a finalidade de prover uma solução para criação e compartilhamento de conjuntos de imagens, com opções de marcações específicas para os elementos presentes e assim facilitar e promover o desenvolvimento de códigos na área de visão computacional [38].

A solução tem uma interface que permite a fácil marcação dos elementos, conforme exibida na Figura 3.1. É também possível publicar conjuntos de imagens para compartilhamento com a comunidade. A ferramenta utiliza ainda etiquetas que facilitam a identificação dos grupos de imagens compartilhados, permitindo que o usuário busque por coleções que são mais adequadas ao trabalho que executará. Com mais de 1500 usuários registrados na plataforma e mais de 250 conjuntos de imagens registradas, o *ImageTagger* oferece uma gama imensa de imagens que podem ser utilizadas para variadas aplicações no contexto da *RoboCup*.

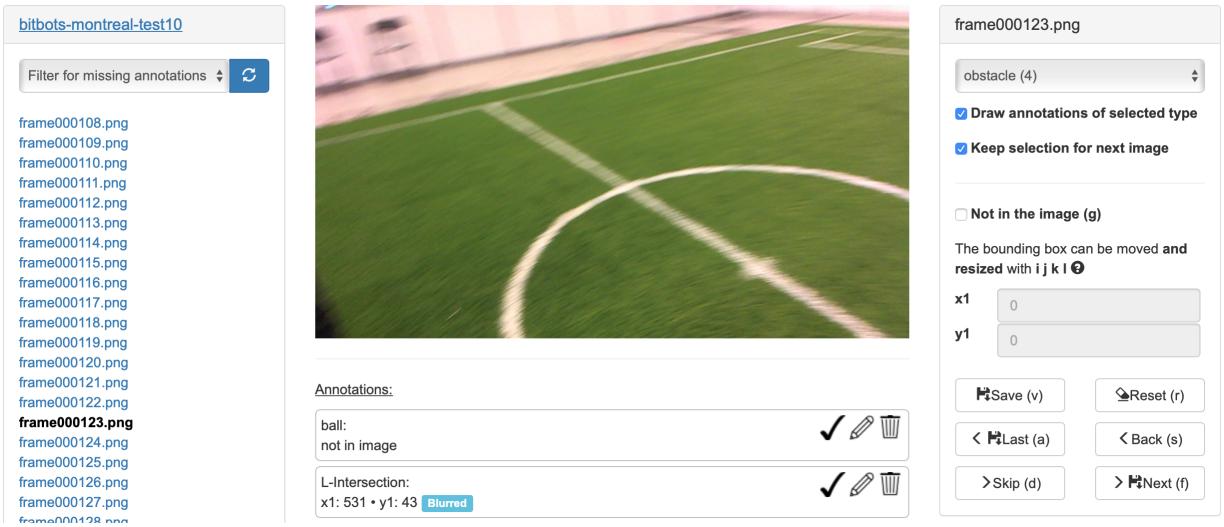


Figura 3.1: Interface da plataforma *ImageTagger*. Fonte: Captura de tela

O conjunto de fotos escolhido no *ImageTagger* foi o de código 363, nomeado *bitbots-montreal-*

test10. Pesquisando inicialmente por grupos que tivessem a etiqueta *lines*, o conjunto *bitbots-montreal-test05* pareceu promissor, mas não trazia tantos casos das *features* que são objeto de estudo deste trabalho. Outros dois conjuntos foram analisados, e no entanto, o *bitbots-montreal-test10*, apesar de não ter a etiqueta, se mostrou extremamente promissor, além de ter um número de imagens mais de três vezes maior do que os outros candidatos.

Após a seleção do grupo, as imagens foram analisadas uma a uma, com a ajuda dos pesquisadores da equipe *UnBeatables*. Usando a ferramenta *ImageTagger*, foi demarcado manualmente para cada imagem a coordenada das *features* presentes ou a sua ausência. Esse procedimento ocorreu tanto para as imagens simuladas como para as imagens reais. Mais de três mil imagens do conjunto foram analisadas e a partir dessa demarcação manual, um gabarito em texto foi exportado.

3.2 Detecção do campo

O primeiro passo para a detecção das *features* de campo é a detecção do próprio campo. Ao longo do jogo, especialmente com a câmera superior, o robô NAO vê parte do campo e parte do que está acima dele, como cenários vazios ou o público que assiste aos jogos. Faz-se necessário desconsiderar da extração de conhecimento qualquer informação que não diz respeito ao jogo. Nesse sentido, considera-se que a câmera inferior do robô vai estar vendo, pelo menos em parte, o campo de futebol e então essa imagem é utilizada para determinação do tom de verde. Dentro do espaço de cor RGB, foi programado o cálculo da proporção de verde na imagem, em comparação ao vermelho e azul, permitindo então a determinação dos limites mínimo e máximo de verde para o ambiente.

Após isso, numa abordagem *top-down*, é possível fazer uma varredura dos pixels coluna a coluna e determinar qual é o primeiro pixel verde. A partir dele até o limite inferior da imagem, todos serão considerados como sendo parte integrante do campo. Utilizando esse critério, o programa monta uma máscara para a imagem original que demarca os pixels pertencentes ao campo, como é possível ver na Figura 3.2(b), que exibe apenas o campo detectado a partir da imagem original correspondente, exibida na Figura 3.2(a).

A fase do algoritmo que faz a varredura coluna a coluna pode ser adaptada, a depender dos requisitos de performance do sistema, para que varra colunas intermitentes e construa a linha delimitadora do campo por interpolação. Dessa forma, a detecção do campo pode não ser tão exata, mas a resposta será mais rápida. Esse custo de oportunidade precisa ser avaliado em condições reais de jogo, o que não foi possível dentro do escopo de desenvolvimento deste trabalho. Ainda assim, o algoritmo já está preparado para fazer essa troca de exatidão por performance.

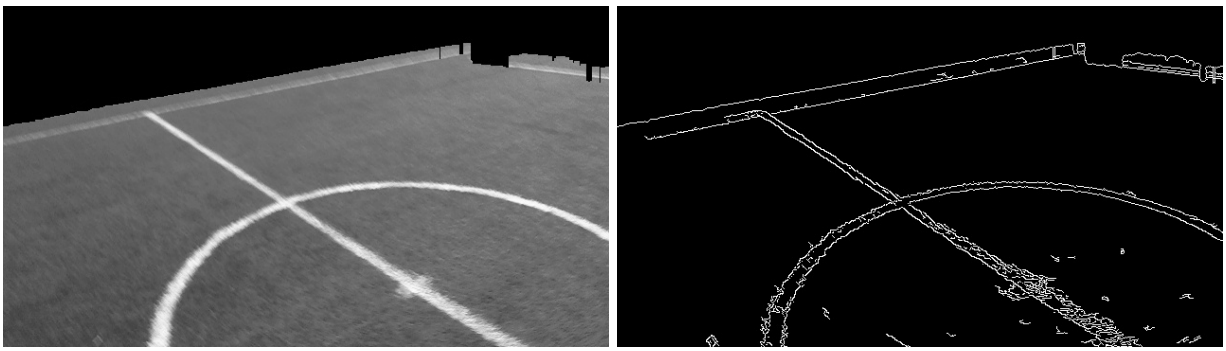
Considerando a detecção do campo como bem sucedida, é possível seguir para a detecção das linhas.

(a) Imagem Original (b) Campo detectado

Figura 3.2: Captura da câmera antes e depois da detecção do campo. Adaptado de ImageTagger

3.3 Detecção de linhas

O método para detecção de linhas começa com a conversão da imagem original para uma em preto e branco, seguida por transformações morfológicas de erosão e dilatação, respectivamente. Nessa ordem, essas operações são capazes de remover pequenos ruídos e linhas muito finas que tendem a atrapalhar a detecção das *features*. Uma operação de *canny* é realizada, a fim de delimitar as bordas [10]. Essas duas operações são mostradas nas Figuras 3.3(a) e 3.3(b), respectivamente. Posteriormente, é aplicada a função *HoughLines*, nativa do *OpenCV*, que exige um parâmetro de limite para consideração de linhas que atinjam determinada pontuação [36]. Os valores desse e de outros parâmetros foram definidos conforme metodologia de teste com um conjunto limitado de imagens que é detalhada na seção 3.5.

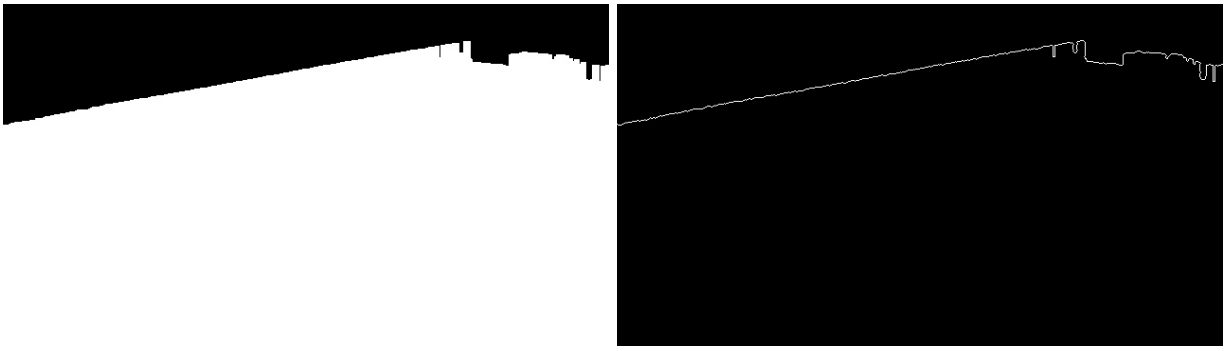


(a) Imagem preta e branca depois de erosão e dilatação (b) Imagem com detecção de bordas

Figura 3.3: Aplicação de transformações que auxiliam na detecção de linhas. Adaptado de ImageTagger

É possível ver na Figura 3.3(b) a definição do contorno da linha de lateral de campo e a linha de contorno do próprio campo. Não é surpreendente, portanto, que ao gerar as linhas, apareça uma que seja a delimitadora do campo. No entanto, essa linha deve ser desconsiderada pelo algoritmo. Nesse caso, a saída foi a detecção das linhas da máscara do campo. Exibida na Figura 3.4, a detecção de linhas é feita na máscara e depois as linhas resultantes, denotadas como delimitadoras, são comparadas com as linhas detectadas como linhas de campo. Se elas têm

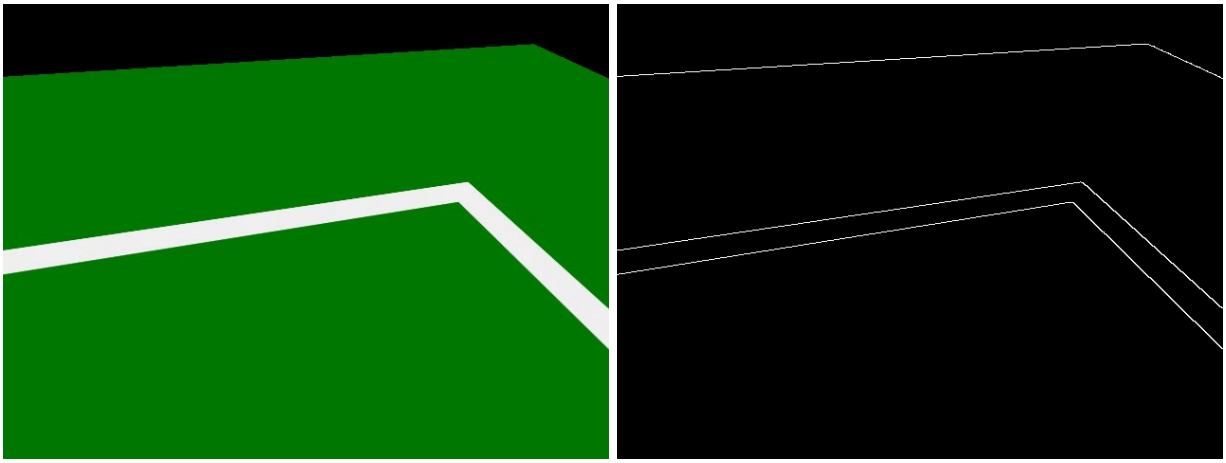
parâmetros ρ e θ similares, são descartadas.



(a) Máscara de detecção de campo (b) Bordas da máscara de detecção de campo

Figura 3.4: Imagens responsáveis pela detecção de linhas delimitadoras. Adaptado de ImageTagger

Existe ainda um outro fator a ser considerado. Por vezes, as linhas de campo podem ser detectadas duplamente pelo algoritmo *HoughLines*. Isso acontece porque elas podem parecer muito grossas, como ilustrado nas Figuras 3.5(a) e 3.5(b), geradas a partir de uma das imagens do conjunto de simulação. Para esses casos, foi elaborado um algoritmo de agrupamento de linhas. Após a eliminação de linhas delimitadoras de campo, as linhas que são consideradas muito similares são agrupadas. Dessa forma, ao se traçar as linhas de campo na imagem final, é possível ter um resultado conforme os exibidos nas Figuras 3.6(a) e 3.6(b).

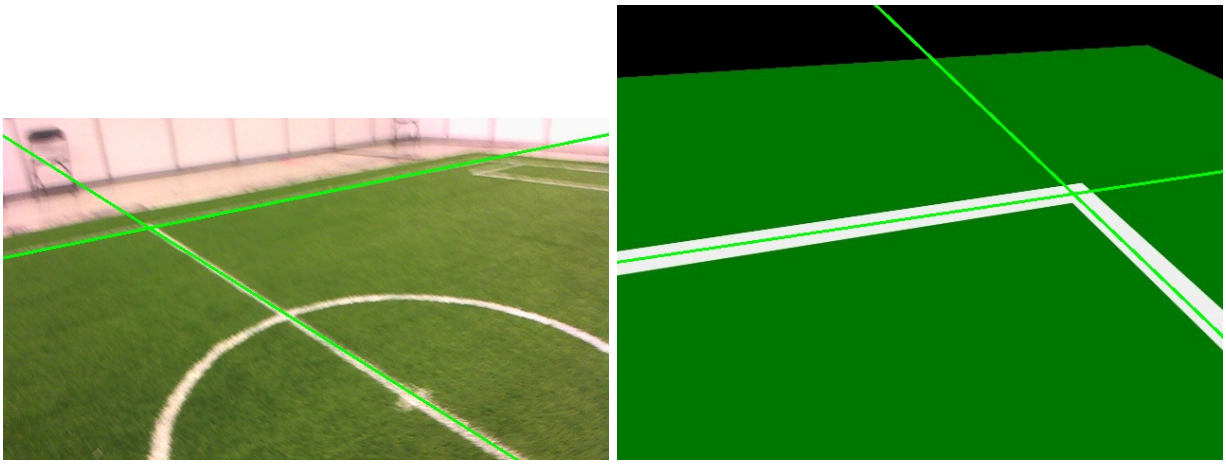


(a) Imagem original (b) Imagem com detecção de bordas

Figura 3.5: Imagens que mostram possível detecção dupla de linhas. Adaptado de ImageTagger

3.4 Interseção das linhas

Com as linhas de campo devidamente detectadas pelo algoritmo, o próximo passo é encontrar o ponto de interseção entre essas retas. Conforme descrito em 2.2.3, para cada uma delas existe um par de parâmetros ρ e θ, armazenado em estruturas de dados apropriadas. A interseção entre cada par de linhas é calculado conforme descrito na Equação 2.5 usando iterações para que cada



(a) Imagem real com linhas detectadas

(b) Imagem simulada com linhas detectadas

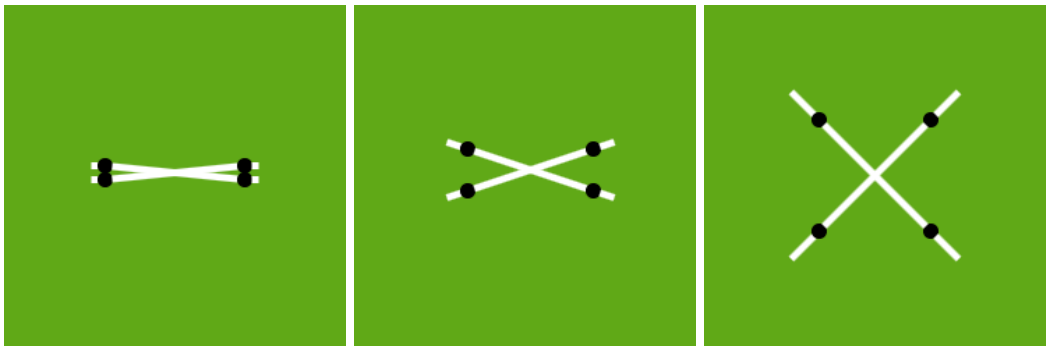
Figura 3.6: Imagens originais com as linhas encontradas traçadas. Adaptado de ImageTagger

linha encontrada seja comparada com as demais para determinação do ponto de encontro. Essas interseções precisam ser então validadas e classificadas.

Duas retas não paralelas sempre têm um ponto de encontro e, no escopo desse trabalho, as coordenadas de tal ponto podem estar fora dos limites da imagem. Essa não é uma interseção que ocorre na imagem, é o resultado de uma prolongação imaginária dos segmentos de reta. Por isso, a primeira etapa da validação é verificar se a interseção está localizada dentro dos limites da imagem e, caso esteja, ela precisa estar localizada sobre um pixel branco, uma vez que apenas interseções das linhas de campo são consideradas.

Para toda interseção detectada, calculam-se coordenadas de controle, usadas para validação e posterior classificação. São quatro pontos que se localizam sobre cada uma das duas retas, equidistantes do ponto de interseção entre elas por distância denotada como r. Os pontos que são posicionados em retas diferentes não podem estar muito próximos um do outro, já que podem simbolizar uma linha cujos parâmetros de agrupamento não foram suficientes para que fossem consideradas uma só. A Figura 3.7 ilustra os pontos de controle por meio de três exemplos. Quando os pontos de controle são muito próximos, como ilustrado na Figura 3.7(a), a interseção é descartada. Já nas Figuras 3.7(b) e 3.7(c) os pontos de controle estão longe o suficiente uns dos outros, o suficiente para a classificação de uma possível interseção.

O próximo passo é a classificação das interseções, ou seja, a determinação de fato da *feature*. Essa classificação se dá pela análise dos pontos de controle e suas proximidades, averiguando a existência de pixels brancos. A confirmação de que quatro pontos de controle estão sobre pixels brancos classifica a *feature* como sendo do tipo X. Três pontos de controle confirmados caracterizam *features* do tipo T. No caso de dois pontos de controle serem brancos, eles só caracterizam uma *feature* L caso esses pontos não sejam opostos. A etapa de classificação é ilustrada pela Figura 3.8. Quando a contagem dos pontos de controle é inferior a dois, a interseção correspondente é descartada.



(a) Pontos de controle extremamente próximos (b) Pontos de controle comuns (c) Pontos de controle em uma distância grande

Figura 3.7: Imagens originais com as linhas encontradas traçadas. Adaptado de ImageTagger

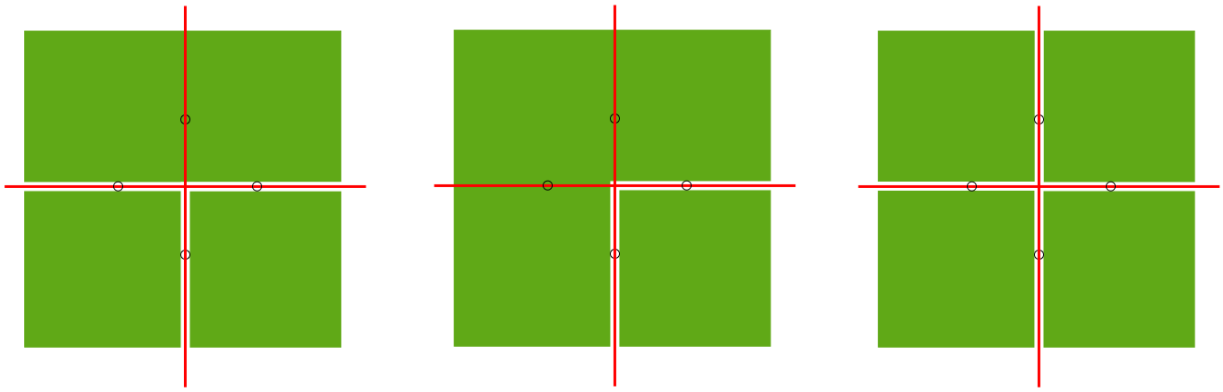


Figura 3.8: *Features* T, L e X com seus pontos de controle ilustrados. Fonte: Autor

3.5 Definição e ajuste de parâmetros

Os parâmetros utilizados ao longo do desenvolvimento do presente trabalho tiveram seus valores definidos a fim de maximizar a acurácia dos resultados com base nos gabaritos construídos. Um dos primeiros parâmetros utilizados no trabalho é o limite para detecção de linhas com a função *houghLines* do OpenCV. A partir da imagem com bordas, a função só retornará como linhas válidas as que obtiverem determinada pontuação, definida pelo parâmetro *threshold*[36].

Há também os parâmetros de agrupamento e similaridade de linhas, $\rho_{threshold}$ e $\theta_{threshold}$, que definem os limites para que duas linhas sejam consideradas semelhantes o suficiente para o seu subsequente agrupamento. Finalmente, existem parâmetros relativos aos pontos de controle, que definem a distância que esses pontos ficam da interseção e o raio de tolerância para que um pixel branco próximo seja o suficiente para considerar tal ponto como válido.

Os valores ótimos dos parâmetros foram definidos por execuções subsequentes do código usando conjuntos de imagens reduzidos e, no caso da simulação, apenas da câmera superior. Cada execução se dava com variações pertinentes desses parâmetros. O resultado é aferido e a escolha dos valores é feita de acordo com a maior quantidade de aferições corretas.

O parâmetro de limite de linhas encontradas foi determinado com variações de 10 e depois interseções menores entre os dois melhores candidatos. As taxas aferidas para o conjunto de simulação são exibidas na Tabela 3.2 e o valor base escolhido para esse parâmetro é 55. A taxa de acerto foi considerada igualando o número de linhas encontradas na imagem com a quantidade de linhas de um gabarito de contagem de linhas. De forma similar, os valores de limites para agrupamento de linhas similares com os parâmetros ρ e θ também foram testados nas imagens de simulação e seus resultados são resumidos na Tabela 3.3, em que se torna claro que os valores devem ser 45 para ρ e 0.3 para θ. Por último, o mesmo método é aplicado considerando os parâmetros referentes aos pontos de controle, que consistem na distância do ponto de controle até a interseção e no raio de tolerância para que aquele ponto de controle seja validado como branco para caracterização das *features*. Os valores escolhidos foram 30 para a distância e 5 para o raio, nas imagens de simulação.

Para as imagens reais, um procedimento similar foi executado e os valores ótimos para cada um dos cinco parâmetros testados foram definidos e são exibidos, junto com os de simulação, na Tabela 3.1. A diferença entre os procedimentos de definição dos valores ótimos entre os dois conjuntos de imagens é que, para as simuladas, há um arquivo de quantidade de linhas em cada quadro, permitindo um ajuste mais fino dos parâmetros de agrupamento de linhas. No caso das imagens reais, há apenas o gabarito de tipos e coordenadas das *features*, em razão da volumetria do conjunto.

Tabela 3.1: Valores ótimos dos parâmetros usados em cada um dos conjuntos de imagens

Parâmetro	**Valor utilizado (Simulação)**	**Valor utilizado (Real)**
Threshold	55	100
ρ	45	15
θ	0.3	0.2
Distância	30	30
Raio	5	3

3.6 Execução do código

O código de detecção das *features* foi executado no conjunto de simulação, que consiste em 385 pares de imagens, considerando as duas câmeras do robô. As imagens da câmera inferior foram usadas apenas para obtenção do tom de verde para detecção do campo, enquanto as da câmera superior passaram pelo escrutínio do código a fim de detectar as *features*. As imagens reais obtidas na ferramenta *ImageTagger* não continham pares com imagens das duas câmeras do robô. Assim, foi selecionada manualmente uma imagem para ser usada na definição do tom de verde a ser usado na detecção do campo nas imagens reais.

Como dito na Seção 2.2, a visão computacional tem como entrada uma imagem e como saída, uma informação. Desse modo, as informações referentes às *features* detectadas pelo programa

Tabela 3.2: Taxas de acerto referentes ao limite para encontrar linhas

Limite testado	**Acerto**
100	40.0%
90	47.0%
80	53.0%
70	58.0%
60	63.5%
59	67.0%
58	68.0%
57	68.0%
56	69.5%
55	71.0%
54	69.5%
53	68.0%
52	70.5%
51	69.0%
50	68.5%
40	58.5%
30	50.5%
20	38.5%

Tabela 3.3: Taxas de acerto referentes ao limite para agrupamento de linhas

Valores de $\rho_{threshold}$	**Valores de $\theta_{threshold}$**				
	0.1	**0.2**	**0.3**	**0.4**	**0.5**
20	60%	61%	61%	60%	63%
25	64%	66%	66%	64%	64%
30	64%	68%	69%	67%	66%
35	66%	71%	72%	69%	67%
40	67%	72%	73%	71%	69%
45	68%	73%	75%	73%	71%

Tabela 3.4: Taxas de acerto referentes aos parâmetros dos pontos de controle

Raio de tolerância	Valores da distância		
	1	2	3
1	19%	21%	20%
2	32%	33%	29%
3	42%	40%	37%
4	45%	44%	41%
5	48%	46%	43%

foram registradas em arquivos de texto. Em tais registros textuais são salvos o nome da imagem, o par de coordenadas (x,y) da *feature* encontrada e o seu tipo, em uma estrutura similar à dos gabaritos, que serão usados para a validação dos resultados.

Para geração das imagens exibidas no texto do presente trabalho, uma execução especial do programa foi realizada em que funções específicas do *OpenCV* foram utilizadas para marcação de texto na imagem com a letra correspondente ao tipo da *feature*, na coordenada em que ela foi encontrada.

Ademais, um código auxiliar foi programado de forma a fazer a comparação entre as *features* detectadas e os gabaritos existentes. Os valores são importados para estruturas de dados no código, ordenados e então comparados, considerando o nome dos arquivos, as coordenadas cartesianas, x e y, e o tipo da *feature*. Embora não seja parte da programação do NAO e nem integrante da estratégia de jogo, tal código é primordial para extração das estatísticas referentes à detecção das *features*. Tanto resultados gerais quanto estatísticas discriminadas por tipo de *feature* fazem parte da saída desse código. Neste código, são considerados como acerto:

- Se a *feature* encontrada na imagem é do mesmo tipo que a registrada no gabarito e se as coordenadas x e y coincidem com o registro correto, com uma tolerância de até 10 pixels em qualquer direção; ou
- Se nenhuma *feature* é encontrada na imagem e no gabarito não há registro para a imagem ou se no gabarito também consta que não há *feature*.

Capítulo 4

Resultados

Este capítulo traz uma visão geral dos resultados obtidos a partir da metodologia previamente descrita, além de discussão e análise detalhadas com a intenção de aumentar a compreensão do que foi obtido.

4.1 Imagens de simulação

As funções desenvolvidas neste trabalho foram testadas com as imagens de simulação e seus resultados salvos em arquivos de registros por sua vez processados pelo código de comparação. Este código é capaz de verificar o sucesso da aferição das *features*, comparando os registros gerados com o arquivo de gabarito através de índices de sucesso. Para a exibição das imagens com resultados neste trabalho, as *features* aferidas foram marcadas em vermelho com suas letras características nas imagens. Essa sinalização não foi realizada manualmente, mas sim usando a própria estrutura do código desenvolvido, marcando automaticamente os tipos de *features* nas coordenadas em que elas foram encontradas.

4.1.1 Exemplos de detecção L, T e X nas imagens simuladas

Alguns exemplos de *features* tipo L são exibidos nas Figuras 4.1, 4.2 e 4.3. Na Figura 4.3 há um caso interessante em que duas *features* L existem na imagem, mas apenas uma delas é detectada. A seção 4.4 traz discussões sobre essa e outras situações adversas. Nas Figuras 4.4, 4.5 e 4.6 é possível ver exemplos de *features* T que foram detectadas em imagens simuladas. De maneira similar à Figura 4.3, a Figura 4.6 também tem uma *feature* L que não foi detectada, evento que será discutido. As Figuras 4.7, 4.8 e 4.9 trazem casos de *features* X que foram detectadas pelo código.

Em todos os exemplos, é possível notar que as *features* encontradas estão muito bem localizadas. Esse é o caso geral para as imagens simuladas, em que a detecção tem um grau de exatidão singular.

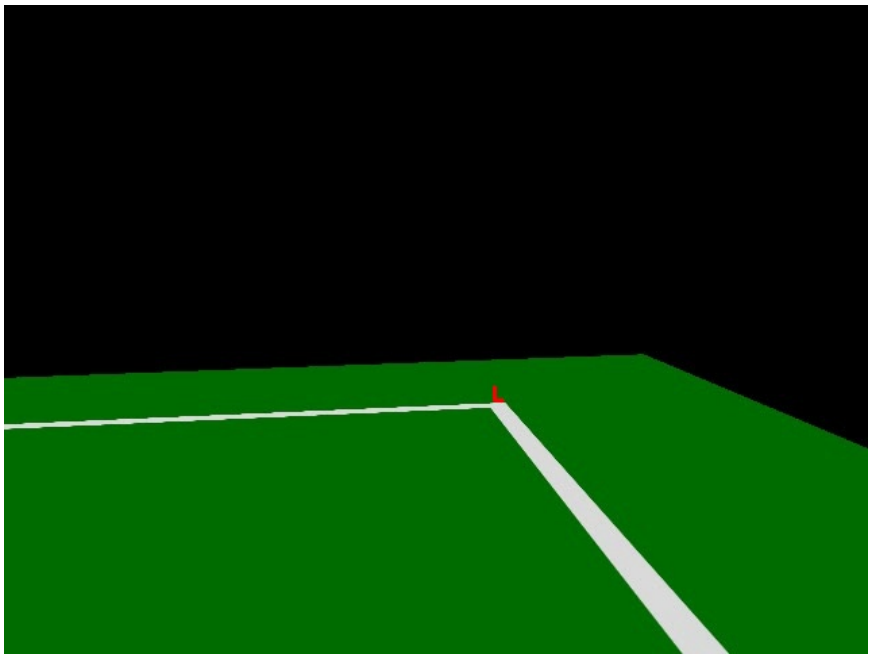


Figura 4.1: Detecção de *feature* L: Exemplo 1

Figura 4.2: Detecção de *feature* L: Exemplo 2

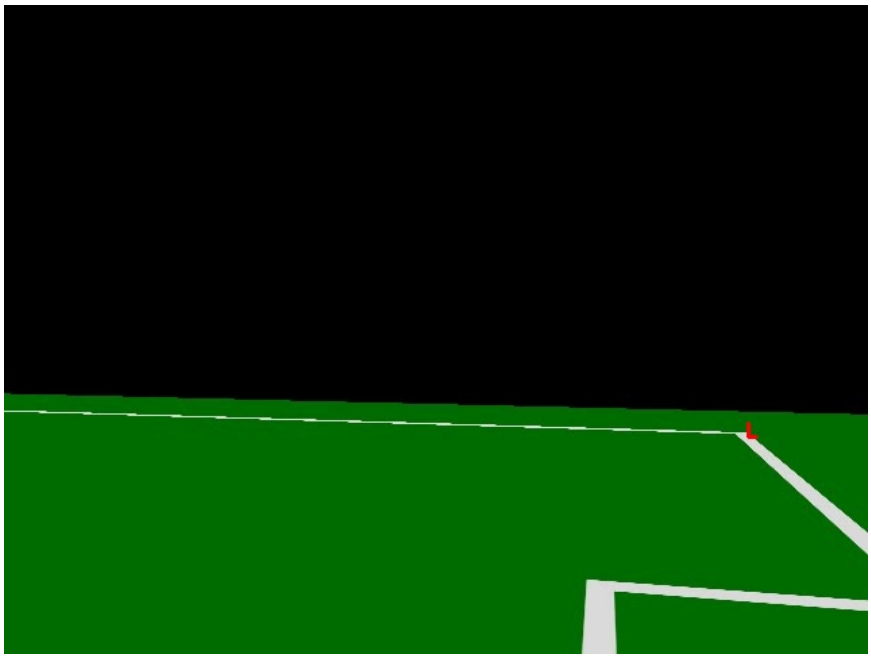


Figura 4.3: Detecção de *feature* L: Exemplo 3

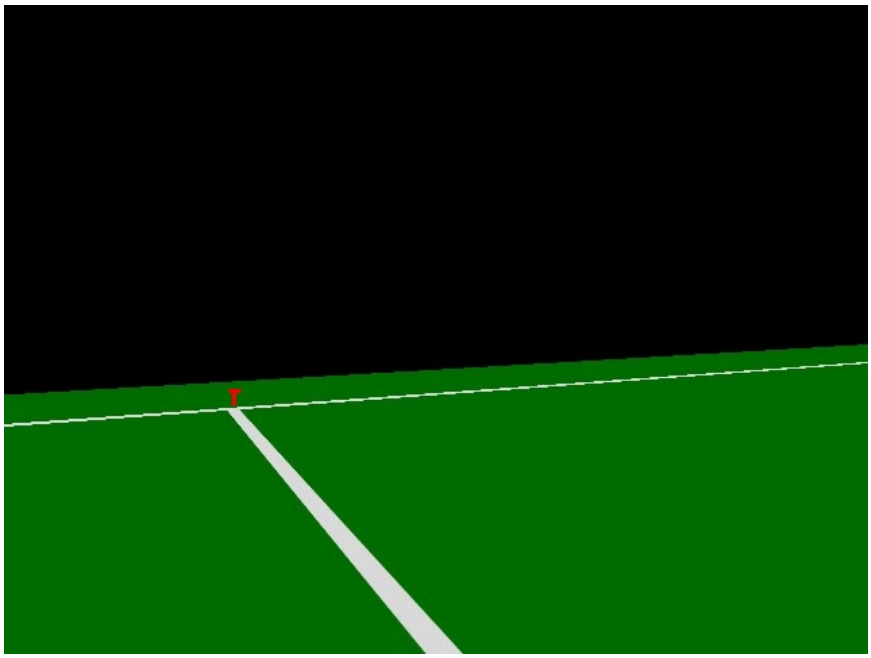


Figura 4.4: Detecção de *feature* T: Exemplo 1

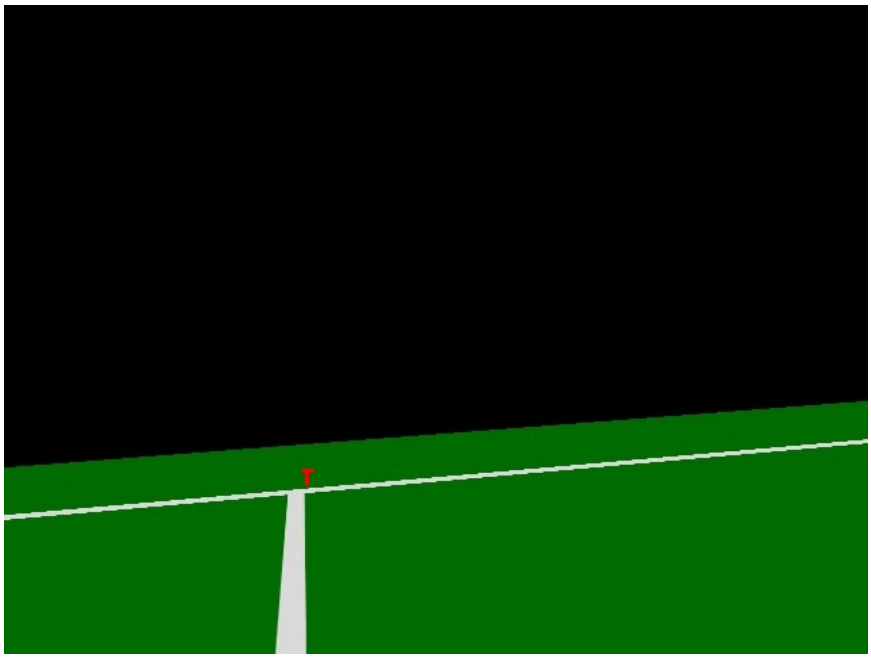


Figura 4.5: Detecção de *feature* T: Exemplo 2

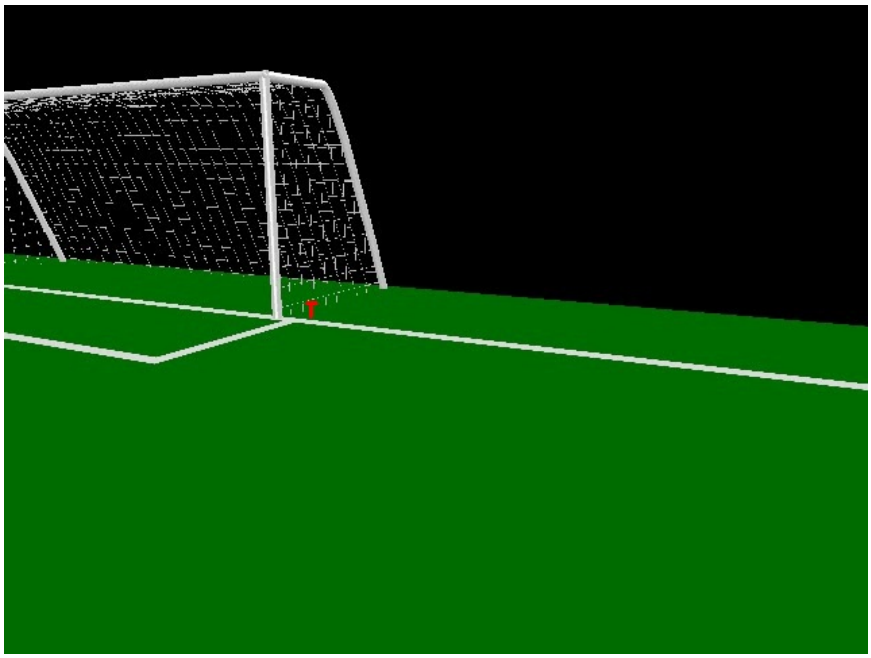


Figura 4.6: Detecção de *feature* T: Exemplo 3

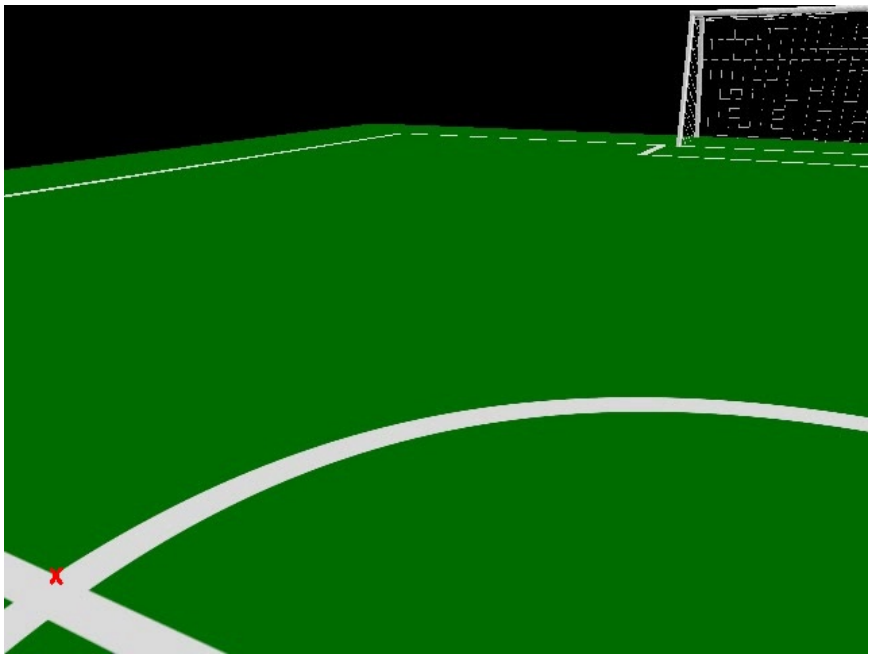


Figura 4.7: Detecção de *feature* X: Exemplo 1

Figura 4.8: Detecção de *feature* X: Exemplo 2

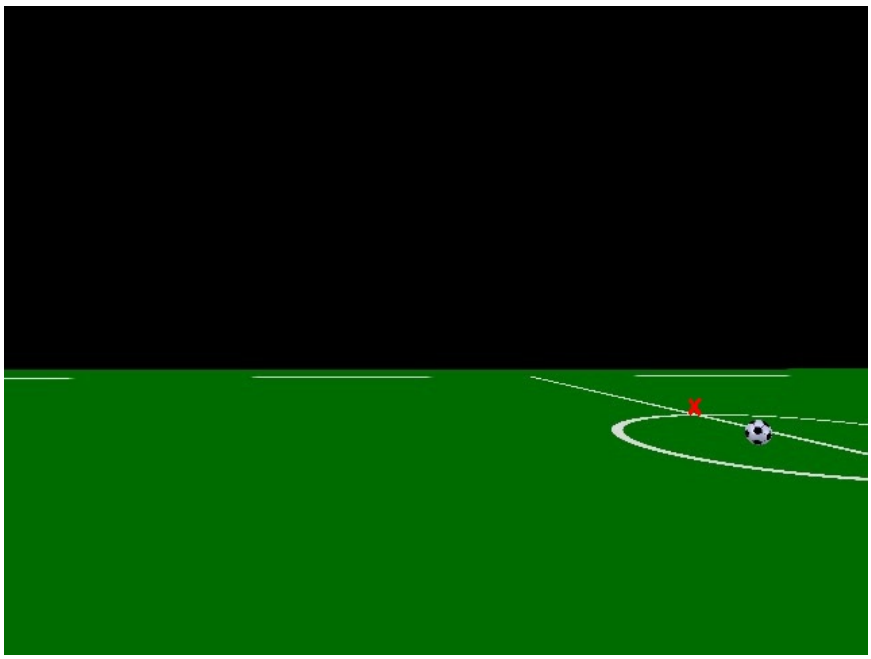


Figura 4.9: Detecção de *feature* X: Exemplo 3

4.1.2 Taxas de detecção nas imagens simuladas

As taxas de detecção de cada uma das *features* e a taxa de detecção total são apresentadas na Tabela 4.1. Vale ainda lembrar que, como mencionado na Seção 3.6, para as taxas totais, foram considerados como acertos também os casos em que tanto as imagens originais quanto o código não têm *features* detectadas.

Tabela 4.1: Estatísticas de detecção de *features* em imagens simuladas

Feature	**Taxa de acerto**
L	67.59%
T	64.96%
X	40.00%
Total	**62.60%**

4.2 Imagens reais

4.2.1 Exemplos de detecção L, T e X nas imagens reais

Com a metodologia idêntica ao conjunto de imagens de simulação e a alteração dos valores dos parâmetros, o código para as imagens reais foi executado. Elas têm características bem diferentes das simuladas, o que permitiu em muitos casos a detecção de *features* distintas nas mesmas imagens. Por outro lado, a identificação múltipla de uma única *feature* acontece com bastante frequência. A Figura 4.10 mostra a detecção bem sucedida de uma *feature* tipo L. Similarmente, a Figura 4.11 também exibe uma detecção bem sucedida, porém duplicada, estando

quase no mesmo lugar. Esse é um evento que se repete para outras *features*, como as tipo X, cuja detecção duplicada é visível nas Figuras 4.12 e 4.11. Exemplos de detecção de *features* tipo T são mostrados nas Figuras 4.14 e 4.15 em que aparecem detectadas junto com outras tipo X.

4.2.2 Taxas de detecção nas imagens reais

As taxas de detecção nas imagens reais são consideravelmente inferiores às da simulação e estão apresentadas na Tabela 4.2. Esses índices serão discutidos de forma mais aprofundada na seção 4.4. Nas imagens reais, é perceptível o movimento do robô durante a captura e esse é um fator determinante para perda de qualidade da identificação e detecção de *features*, já que as linhas aparecem borradas.

Tabela 4.2: Estatísticas de detecção de *features* em imagens reais

Feature	**Taxa de acerto**
L	32.65%
T	32.72%
X	11.84%
Total	**21.49%**

Figura 4.10: Exemplo de detecção de *feature* L

Figura 4.11: Detecção de *feature* L duplicada

Figura 4.12: Detecção de *feature* X duplicada

Figura 4.13: Detecção de *feature* X triplicada

Figura 4.14: Detecção de *feature* T: Exemplo 1

Figura 4.15: Detecção de *feature* T: Exemplo 2

4.3 Falsos positivos e falsos negativos

Foi feita a aferição da quantidade de falsos positivos e falsos negativos no que diz respeito à detecção das *features*, taxas apresentadas na Tabela 4.3. A quantidade de falsos negativos merece atenção, visto que representam imagens em que não houve detecção, mas existia pelo menos uma *feature*.

Tabela 4.3: Falsos positivos e falsos negativos da detecção de *features*

Grupo de imagens	**Falsos positivos**	**Falsos negativos**
Reais	7.70%	22.80%
Simuladas	5.26%	26.31%

4.4 Discussão

Uma taxa de acerto de mais de 60% nas imagens simuladas é extremamente satisfatória, especialmente considerando condições atenuantes dessas imagens. Uma destas é o fato de as linhas ficarem tracejadas quando longe da visão do robô, por uma questão da renderização da imagem do próprio programa de simulação. Isso certamente atrapalha a detecção das linhas, já que podem acabar sendo descartadas pelo limite da função *houghLines*. Mais ainda, esse tracejado pode atrapalhar na validação dos pontos de controle para caracterização da *feature*, eventualmente podendo levar a uma classificação incorreta. Nas Figuras 4.8 e 4.9, existem *features* não identificadas pelo programa justamente em razão desse tracejado e na 4.7, a tipo L não é nem sequer encontrada porque o programa não identifica a linha horizontal tracejada do fundo do campo de futebol.

Nas imagens simuladas, o gol é modelado em 3D de forma que as cordas de sua rede ficam extremamente retas, interferindo na detecção de linhas do programa, especialmente quando o robô está próximo ao gol, numa posição em que a rede é vista dentro do limite do campo. Nessas ocasiões, o robô encontra interseções incorretas, mesmo com operações de transformação morfológica. Isso porque quando a operação usa parâmetros suficientes para remover as linhas da rede, ela desconsidera linhas que ficam distantes do robô, de forma que tal solução não é muito apropriada. No entanto, em situações reais, a rede do gol, como vista pelo robô, não possui cordas tão retas a ponto de serem consideradas linhas de campo. Por isso, esse é um fator que não chegará a interferir em aferições reais, só atrapalha a estatística de acerto no caso da simulação.

É importante ainda trazer para discussão dos resultados o fato de que retas muito curtas raramente passam pelo valor de limite da função de detecção de linhas e, por isso, *features* que ficam muito próximas das bordas da imagem podem não ser identificadas, como é o caso da Figura 4.3.

Outro aspecto importante para a interpretação dos resultados é a queda na taxa de sucesso na detecção de *features* do tipo X. Isso acontece por dois motivos, sendo que o primeiro é o fato da quantidade desse tipo ser muito inferior comparado aos outros e o segundo é o fato de ele ser

caracterizado por uma linha reta e uma linha curva, que apesar de ser encontrada pelo método em muitos casos, não é a situação mais prevista por ele.

Em ambas coleções de imagens, ocorreu um baixo número de falsos positivos e alto número de falsos negativos. Isso é indicativo de que muitas *features* podem estar sendo descartadas, o que abre a possibilidade para a reconsideração dos critérios de validação de *features* e de comparação do resultado da detecção com o gabarito. Embora o critério de avaliação tenha passado por alguns testes de flexibilização, os resultados variam bruscamente com a alteração da tolerância para que a *feature* seja dada como correta. Pode ser interessante usar um critério que considere a detecção como certa conforme uma tolerância variável de acordo com a altura da *feature* na imagem, sendo que quanto mais embaixo, maior a tolerância e ela vai reduzindo conforme chega à parte superior da imagem, correspondendo à uma perspectiva de horizonte.

No que diz respeito exclusivamente a imagens reais, a taxa de acerto de 21% ficou longe do esperado. Mais de 500 imagens foram analisadas individualmente considerando as marcações de detecção em cada uma e o motivo que empurra a taxa para baixo é a visão ofuscada do robô, dada a sua movimentação constante. Dessa forma, constata-se que a qualidade das imagens foi fator determinante para o resultado insatisfatório.

É possível ver as imagens com linhas borradas nos resultados exibidos. As linhas de campo ficam borradas a ponto de as *features* ficarem duplicadas porque a função de agrupamento não chegou a agrupar as linhas. Isso pode ter acontecido em decorrência da falta de um gabarito de quantidade de linhas para o conjunto importado do *ImageTagger*, o que não permitiu um ajuste mais fino dos parâmetros de detecção e agrupamento das linhas. Especialmente nas Figuras 4.10 e 4.11 é possível notar essa característica que é tão presente nas imagens reais. Em especial na Figura 4.11 é perceptível o desaparecimento da linha lateral delimitadora do campo da visão do robô, de forma que a *feature* do canto do campo não foi encontrada por falta de identificação da linha.

A identificação duplicada das *features* muito próximas não é vista necessariamente como um problema nessa etapa, especialmente porque o uso de imagens com melhor qualidade tende a diminuir esse problema, baseado no fato de que as imagens simuladas não apresentaram tal condição. Uma solução possível para esse efeito envolve o desenvolvimento de funções de agrupamento dessas *features* considerando limites de proximidade entre pontos para que elas sejam consideradas iguais. Essa é uma proposta que exige estudo cauteloso para a definição de tal parâmetro, uma vez que dependendo do ângulo, o robô pode ver duas *features* que são distintas, mas próximas o suficiente para que o código as agrupe. Um exemplo desse ângulo de visão é mostrado na Figura 3.2(a), em que duas *features* L distintas têm coordenadas extremamente similares.

Na verdade, uma solução mais complexa, porém robusta para o problema, pode ser o uso de filtros de interpolação entre duas imagens, que tende a reduzir tal efeito e ainda melhorar a detecção das *features* usando o método que foi desenvolvido e descrito ao longo desse trabalho. Outra solução que traz robustez ao sistema de detecção de *features* é a abordagem adaptativa no que diz respeito à definição de parâmetros, usando como entrada para calibração informações obtidas dos sensores do NAO que sejam capazes de estimar sua posição. Dependendo da posição

e da visão que o robô tem do campo, os valores dos parâmetros terão valores ótimos distintos. Levar isso em consideração para executar a detecção das linhas e validar as suas interseções pode melhorar as taxas de acerto.

Capítulo 5

Conclusões

O objetivo principal deste trabalho de graduação é o desenvolvimento de um algoritmo de detecção e identificação de elementos de um campo de futebol de robôs a partir de imagens de agentes participantes do jogo. Utilizando programação *Python* e auxílio de funcionalidades da biblioteca *OpenCV*, diversas funções foram criadas para a identificação das chamadas *features* de campo.

A metodologia aplicada envolve a identificação do campo de futebol, desconsiderando todos os elementos pertencentes aos arredores, seguida pela identificação de linhas e cálculo dos seus pontos de interseção. A análise das coordenadas desses pontos de interseção foi feita utilizando pontos de controle que classificam as *features* em três categorias distintas: L, T e X. Os parâmetros utilizados nesse trabalho tiveram seus valores ótimos definidos com um método projetado com a intenção de maximização do desempenho do código.

Este trabalho foi testado utilizando imagens de simulação fornecidas pela equipe de competição de robôs da UnB, a *UnBeatables*. Considerando a importância de testar códigos em imagens reais e frente a dificuldade de obtenção própria, recorreu-se à plataforma alemã *ImageTagger*, criada em ambiente universitário para compartilhamento de bancos de imagens relativas ao futebol de robôs. O conjunto de imagens reais foi selecionado e mais de três mil imagens passaram por apuração manual para demarcação das *features* com auxílio da equipe *UnBeatables*. Tal marcação permitiu posterior análise da desempenho do código de identificação.

Os resultados para imagens simuladas teve um alto nível de satisfação, enquanto a taxa de acerto obtida com imagens reais diminuiu. Foram levantadas as razões para a diferença entre os resultados e propostas de intervenções foram sugeridas para perspectivas de pesquisas futuras.

Este trabalho apresenta uma contribuição valiosa no método de identificação de *features* de campo e resultados parcialmente consistentes. Ainda assim, existem muitas possibilidades para desdobramentos futuros. Alternativas envolvem o aprimoramento dos resultados atuais considerando as imagens reais ou a obtenção de imagens próprias para testagem do algoritmo. Dentre os caminhos que este trabalho deixa em aberto para futuro desenvolvimento de pesquisa, é possível enumerar:

1. Aplicação das técnicas sugeridas na seção 4.4 e reavaliação dos resultados

   As sugestões envolvem a obtenção de imagens próprias, estudo e implementação de algoritmo que agrupe *features* muito próximas e utilização de filtros de interpolação para redução imagens borradas e aumento da taxa de acerto do código implementado. Sugestões de melhoria também envolvem o uso de técnicas mais robustas de detecção de linhas, especialmente envolvendo *clusters*.

2. A inclusão de camadas para prever casos de falha, fazer filtragem de informações, e rastreamento das *features* detectadas pode ser interessante como trabalho futuro.

   O acompanhamento das coordenadas detectadas vinculadas a cada um dos tipos de *features* pode ser útil quando vinculado a algoritmos de predição e filtragem que possa as descartar caso sejam consideradas incorretas ou corrigir a tipagem de uma *feature* que pode ter sofrido uma classificação mal sucedida. A associação desse algoritmo com o código desenvolvido pode traz mais robustez à detecção.

3. A integração do código com o módulo de jogo do NAO, para que o dispositivo seja capaz de identificar as *features* em tempo real, conforme captura as imagens da própria câmera. Essa adaptação abre espaço para testes de desempenho do código considerando o processador do dispositivo.

4. A integração do código com as funções de localização que estão sendo implementadas em trabalho de graduação em desenvolvimento concomitante a este. Essa sugestão ainda se alia ao fato de que o ajuste de parâmetros pode ser realizado vinculado diretamente à posição do robô, visto que eles podem depender bastante da perspectiva de visão da câmera.

5. A reprogramação da calibração do verde para a detecção do campo de tempos em tempos, otimizando o tempo de processamento das imagens e reduzindo os requisitos de CPU do robô, uma vez que, como um sistema em tempo real, ele precisa adquirir os dados e tomar decisões em tempo hábil compatível com sua aplicação.

As possibilidades para a continuação deste trabalho são promissoras, com o potencial de trazer grandes contribuições para os campos de pesquisa em robótica, visão computacional e inteligência artificial.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] TOURETZKY, D. S. Preparing computer science students for the robotics revolution. *Communications of the ACM*, ACM New York, NY, USA, v. 53, n. 8, p. 27–29, 2010.

[2] BRASIL, L. *LEGO Mindstorms - EV3 - LEGO*. Disponível em: <https://www.legobrasil.com.br/lego-mindstorms-ev3/p#>.

[3] MATARIĆ, M. J. et al. *The robotics primer*. [S.l.]: Mit Press, 2007.

[4] FEDERATION, R. *RoboCup Federation official website*. Disponível em: <https://www.robocup.org>.

[5] ROBOTICS, S. *NAO the humanoid and programmable robot*. Disponível em: <https://www.softbankrobotics.com/emea/en/nao>.

[6] INC, S. E. *Sony aibo | autonomous companion robot*. Disponível em: <https://direct.sony.com/aibo-ERS1000W/>.

[7] ROBOTICS, S. *NAO Documentation*. Disponível em: <http://doc.aldebaran.com/2-8/home_nao.html>.

[8] ROBOTICS, S. *NAO - Video camera*. Disponível em: <http://doc.aldebaran.com/2-1/family/robots/video_robot.html>.

[9] CARDANI, D. Adventures in hsv space. *Laboratorio de Robótica, Instituto Tecnológico Autónomo de México*, Citeseer, 2001.

[10] DOCUMENTATION, O. *Canny Edge Detection - OpenCV-Python Tutorials*. Disponível em: <https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html>.

[11] SANCHEZ, T. G. *Artificial Vision in the Nao Humanoid Robot*. 1–116 p. Tese (Doutorado), 2009.

[12] WALLÉN, J. *The history of the industrial robot*. [S.l.]: Linköping University Electronic Press, 2008.

[13] MUBIN, O. et al. A review of the applicability of robots in education. *Journal of Technology in Education and Learning*, v. 1, n. 209-0015, p. 13, 2013.

[14] KANDA, T. et al. Interactive robots as social partners and peer tutors for children: A field trial. *Human–Computer Interaction*, Taylor & Francis, v. 19, n. 1-2, p. 61–84, 2004.

[15] MUBIN, O. et al. Improving speech recognition with the robot interaction language. *Disruptive science and Technology*, Mary Ann Liebert, Inc. 140 Huguenot Street, 3rd Floor New Rochelle, NY 10801 USA, v. 1, n. 2, p. 79–88, 2012.

[16] KITANO, H. et al. Robocup: Robot world cup. *IEEE Robotics & Automation Magazine*, IEEE, v. 5, n. 3, p. 30–36, 1998.

[17] AKIN, H. L. et al. Robocup rescue robot and simulation leagues. *AI magazine*, v. 34, n. 1, p. 78–78, 2013.

[18] COMMITTEE, R. T. *Middle Size Robot League Rules and Regulations for 2020*. Disponível em: <`https://www.robocup.org/system/sub_leagues/rules_papers/000/000/006/original/Rulebook_MSL2020_v21.4.pdf`>.

[19] COMMITTEE, R. T. *RoboCup Standard Platform League (NAO) Rule Book*. Disponível em: <`https://spl.robocup.org/wp-content/uploads/downloads/Rules2019.pdf`>.

[20] ANDREASSON, R. et al. Affective touch in human–robot interaction: conveying emotion to the nao robot. *International Journal of Social Robotics*, Springer, v. 10, n. 4, p. 473–491, 2018.

[21] AL-KHALIFA, H. S. et al. The experience of developing mr. saud educational system using nao humanoid robot. In: IEEE. *2017 6th International Conference on Information and Communication Technology and Accessibility (ICTA)*. [S.l.], 2017. p. 1–3.

[22] LEENES, R.; LUCIVERO, F. Laws on robots, laws by robots, laws in robots: regulating robot behaviour by design. *Law, Innovation and Technology*, Taylor & Francis, v. 6, n. 2, p. 193–220, 2014.

[23] KHODABANDEHLOO, K.; SAYLES, R.; HUSBAND, T. Safety integrity assessment of robot systems. *IFAC Proceedings Volumes*, Elsevier, v. 18, n. 12, p. 13–20, 1985.

[24] YONEMOTO, K. Robotization in japan–an examination of the socio–economic impacts of industrial robots. *International Journal of Technology Management*, Inderscience Publishers, v. 1, n. 1-2, p. 179–196, 1986.

[25] POOLE, D. I.; GOEBEL, R. G.; MACKWORTH, A. K. *Computational intelligence*. [S.l.]: Oxford University Press New York, 1998.

[26] ROBOTICS, S. *NAO Documentation*. Disponível em: <`http://doc.aldebaran.com/2-1/home_nao.html`>.

[27] PRINCE, S. J. *Computer vision: models, learning, and inference*. [S.l.]: Cambridge University Press, 2012.

[28] MA, Y. et al. *An invitation to 3-d vision: from images to geometric models*. [S.l.]: Springer Science & Business Media, 2012.

[29] NIXON, M.; AGUADO, A. *Feature extraction and image processing for computer vision.* [S.l.]: Academic press, 2019.

[30] SHUHUA, L.; GAIZHI, G. The application of improved hsv color space model in image processing. In: IEEE. *2010 2nd International Conference on Future Computer and Communication.* [S.l.], 2010. v. 2, p. V2–10.

[31] GONZALES, R. C.; WOODS, R. E. *Digital image processing.* [S.l.]: Prentice hall New Jersey, 2002.

[32] FORSYTH, D. A.; PONCE, J. *Computer vision: a modern approach.* [S.l.]: Prentice Hall Professional Technical Reference, 2002.

[33] CHENG, H.-D. et al. Color image segmentation: advances and prospects. *Pattern recognition*, Elsevier, v. 34, n. 12, p. 2259–2281, 2001.

[34] BRADSKI, G.; KAEHLER, A. *Learning OpenCV: Computer vision with the OpenCV library.* [S.l.]: "O'Reilly Media, Inc.", 2008.

[35] DOCUMENTATION, O. *Morphological Transformations - OpenCV-Python Tutorials.* Disponível em: <`https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html`>.

[36] DOCUMENTATION, O. *Hough Line Transform - OpenCV-Python Tutorials.* Disponível em: <`http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html`>.

[37] VINCE, J. *Geometry for computer graphics: formulae, examples and proofs.* [S.l.]: Springer, 2006.

[38] FIEDLER, N.; BESTMANN, M.; HENDRICH, N. Imagetagger: An open source online platform for collaborative image labeling. In: SPRINGER. *Robot World Cup.* [S.l.], 2018. p. 162–169.

ANEXOS

I. PROGRAMAS UTILIZADOS

Os códigos utilizados nesse trabalho encontram-se em repositório virtual e estão disponíveis para download[1]. As instruções para rodar o código encontram-se no próprio portal do GitHub. Todos os testes foram realizados em ambiente macOS com processador de 2.9 GHz Dual-Core Intel Core i5 e 8GB de memória RAM. O código funciona para as versões 2 e 3 do Python.

[1] https://gitlab.com/samuels15/naocv