Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

# Detecção de objetos Very Small Size Soccer utilizando redes neurais convolucionais YOLO

Ayssa Giovanna de Oliveira Marques

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Marcus Vinicius Lamar

Brasília
2025

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof. Dr. Marcelo Grandi Mandelli

Banca examinadora composta por:

Prof. Marcus Vinicius Lamar (Orientador) — CIC/UnB
Prof.ª Dr.ª Carla Maria Chagas e Cavalcante Koike — CIC/UnB
Prof. Dr. Flávio de Barros Vidal — CIC/UnB

**CIP — Catalogação Internacional na Publicação**

Marques, Ayssa Giovanna de Oliveira.

Detecção de objetos Very Small Size Soccer utilizando redes neurais convolucionais YOLO / Ayssa Giovanna de Oliveira Marques. Brasília : UnB, 2025.
51 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2025.

1. Aprendizado de Máquina, 2. Detecção de Objetos, 3. Visão Computacional, 4. Redes Neurais Profundas, 5. Futebol de Robôs

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

# Detecção de objetos Very Small Size Soccer utilizando redes neurais convolucionais YOLO

Ayssa Giovanna de Oliveira Marques

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Marcus Vinicius Lamar (Orientador)
CIC/UnB

Prof.ª Dr.ª Carla Maria Chagas e Cavalcante Koike
CIC/UnB

Prof. Dr. Flávio de Barros Vidal
CIC/UnB

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 19 de fevereiro de 2025

# Agradecimentos

Agradeço a minha família, meus pais e meus irmãos mais novos, pelo apoio e carinho durante a minha graduação. Agradeço aos meus amigos e pessoas importantes que passaram pela minha vida nesse período, que tornaram a vida acadêmica mais leve; aos professores que inspiraram ideias e sonhos, em especial ao meu orientador que foi fundamental na reta final dessa estrada.

# Resumo

O sistema visual humano analisa uma imagem e, com base em seu conhecimento prévio sobre as características de determinados objetos, identifica quase instantaneamente quais objetos estão presentes, suas localizações e como eles interagem entre si. Algoritmos rápidos e precisos para detecção de objetos podem se aproximar da eficácia da visão humana, permitindo que computadores dirijam carros sem sensores especializados, dispositivos de assistência transmitam informações em tempo real da cena para usuários humanos e desbloqueiem o potencial para sistemas robóticos de uso geral e responsivos. E por que não jogar futebol? Este trabalho propõe uma abordagem para a detecção de objetos em partidas de futebol de robôs da categoria IEEE *Very Small Size Soccer* (VSSS), utilizando redes neurais convolucionais da família *You Only Look Once* (YOLO), especificamente a versão YOLOv8-pose. O projeto envolveu a criação de um banco de imagens a partir de vídeos de partidas da IronCup 2020, onde foram anotados os pontos-chave dos robôs e da bola para treinamento do modelo. Foram realizados treinamentos com diferentes variantes do YOLOv8-pose (*nano, small, medium, large e extra-large*), variando o número de épocas para avaliar a precisão e a perda do modelo. O modelo YOLOv8x-pose demonstrou o melhor desempenho, com menor perda e maior precisão na detecção dos objetos e cálculo da orientação dos robôs. Os resultados mostraram que o sistema proposto, denominado *Ballnet Pose*, superou o *Main System* em termos de precisão na localização dos objetos e no cálculo da orientação dos robôs.

**Palavras-chave:** Aprendizado de Máquina, Detecção de Objetos, Visão Computacional, Redes Neurais Profundas, Futebol de Robôs

# Abstract

The human visual system analyzes an image and, based on its prior knowledge about the characteristics of certain objects, almost instantly identifies which objects are present, their locations and how they interact with each other. Fast and accurate object detection algorithms can approach the effectiveness of human vision, enabling computers to drive cars without specialized sensors, assistive devices to transmit real-time scene information to human users, and unlock the potential for general-purpose, responsive robotic systems. And why not play soccer? This work proposes an approach for object detection in IEEE *Very Small Size Soccer* (VSSS) matches using convolutional neural networks from the *You Only Look Once* (YOLO) family, specifically the YOLOv8-pose version. The project involved creating an image dataset from videos of matches from the IronCup 2020, where keypoints of the robots and the ball were annotated for model training. Training sessions were conducted with different variants of YOLOv8-pose (nano, small, medium, large, and extra-large), varying the number of epochs to evaluate the model's accuracy and loss. The YOLOv8x-pose model demonstrated the best performance, with the lowest loss and highest accuracy in object detection and robot orientation calculation. The results showed that the proposed system, named BallnetPose, outperformed the Main System in terms of both accuracy in object localization and robot orientation calculation.

**Keywords:** Machine Learning, Object Detection, Computer Vision, Deep Neural Networks, Soccer Robotics

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Lista de Abreviaturas e Siglas

**mAP** *Mean Average Precision.*

**MS** *Main System.*

**OKS** *Object Keypoint Similarity.*

**R-CNN** *Region-based Convolutional Neural Network.*

**SIFT** *Scale-Invariant Feature Transform .*

**SSD** *Single Shot Detector.*

**SVM** *Support Vector Machine.*

**VSSS** *Very Small Size Soccer.*

**YOLO** *You Only Look Once.*

# Capítulo 1

# Introdução

O termo inteligência artificial descreve sistemas que mimetizam funções humanas cognitivas associadas com tarefas até então exclusivas de seres humanos como: aprender, raciocinar e resolver problemas. Outra característica intrinsecamente humana é a capacidade de perceber o ambiente e agir de acordo com a informação adquirida, sendo a visão um dos principais meios pelo qual os seres humanos alcançam esse propósito. Através dela, obtemos informações tanto para a manipulação de objetos - pegá-los, agarrá-los, girá-los - quanto para a nossa movimentação no ambiente, evitando obstáculos.

A capacidade de usar a visão para esses propósitos está presente nos sistemas visuais mais primitivos dos animais, e embora a percepção de contexto pareça ser uma atividade de pouco esforço para os humanos, ela exige uma quantidade e complexidade significativa de computação [1]. É chamada de visão computacional a área de estudo cujo objetivo final é usar computadores para emular a visão humana, utilizando-se de uma variedade de fontes de informação; diferentemente das limitações humanas que se restringem a um intervalo de ondas eletromagnéticas, sistemas computacionais dispõem de representações digitais advindas de ultrassonografias, raios X, ultravioletas e outros mais [2].

Damos o nome de "agentes" às entidades que possuem a capacidade de atuar sobre o meio a partir de informação percebida do ambiente, e quanto aos dispositivos que esses agentes utilizam para obter determinada informação, nomeamos "sensores". Um agente humano tem olhos para perceber o ambiente de forma visual, um agente robótico tem câmeras. Ao adicionar autonomia a um agente, obtemos um robô. Como define Maja Matarić [3], um robô é "um sistema autônomo que existe no mundo físico, pode perceber o seu ambiente e agir nele para atingir um objetivo".

O sistema humano de visão captura uma imagem e, tendo conhecimento prévio das características de determinados objetos, quase que instantaneamente sabe quais estão presentes, onde estão e como interagem. Algoritmos rápidos e precisos para detecção de objetos podem aproximar-se da eficácia da visão humana, permitindo sistemas robóticos se

tornem responsivos e de uso geral. Algoritmos rápidos e precisos para detecção de objetos podem se aproximar da eficácia da visão humana, permitindo que computadores dirijam carros sem sensores especializados, dispositivos de assistência transmitam informações em tempo real da cena para usuários humanos e desbloqueiem o potencial para sistemas robóticos de uso geral e responsivos [4]. E por que não jogar futebol?

A RoboCup Federation propõe que até 2050 será possível que uma equipe totalmente autônoma de robôs humanoides vença um jogo contra o time campeão da última Copa do Mundo, utilizando as regras da FIFA [5]. No intuito de estimular a produção científica que permitirá a realização desse desafio, a organização promove competições, simpósios, congressos, e outros eventos. Uma das categorias de futebol de robôs promovida a é a IEEE *Very Small Size Soccer* (VSSS). Competições da RoboCup não são as únicas que contam com a modalidade, sendo comum que sejam utilizadas câmeras digitais para que os jogadores recebam informações do ambiente. No contexto de futebol de robôs *Very Small Size Soccer*, existem alguns trabalhos que propõem soluções com aprendizado de máquina profundo. No entanto, não foram encontrados exemplos que procurem detectar objetos em uma partida. Trabalhos como Medeiros et al. (2020) [6], Baldão et al. (2020) [7], Martins et al. (2021) [8], e Baldão et al. (2024) [9] investigam o desempenho de redes profundas para tarefas de tomada decisão, a maioria com foco em aprendizado por reforço.

A equipe UnBall, projeto de extensão da Universidade de Brasília que compete em eventos de robótica nacionais e internacionais, desenvolveu o sistema de *software* denominado *Main System* (MS) para obter as localizações de seus robôs e da bola em campo, e também se comunicar com seus robôs em partidas [10]. O *software* é desenvolvido em Python, tendo a maioria de suas funcionalidades de visão computacional implementadas utilizando a biblioteca OpenCV. Quatro etapas principais compõem o algoritmo do *Main System*: pré-processamento, segmentação, identificação e rastreamento. A entrada do sistema consiste em uma sequência de quadros que registra o estado de uma partida, advinda de uma câmera digital posicionada a 2 metros do campo onde ela acontece. Seguindo as etapas mencionadas, obtém-se: uma lista com coordenadas $(x, y)$ em metros do centro dos robôs identificados e suas orientações em graus, e coordenadas $(x, y)$ do centro da bola, também em metros.

Este trabalho tem como objetivo extrair das imagens de entrada (sejam imagens estáticas ou *streaming* de vídeo) os mesmos dados obtidos através do *Main System*, utilizando redes neurais convolucionais, e contribuir com a literatura científica sobre a aplicação de modelos de inteligência artificial em sistemas de visão computacional para futebol de robôs *Very Small Size Soccer*.

Este trabalho encontra-se assim dividido: conceitos-chave relacionados ao tema Detecção de Objetos e Redes Convolucionais são apresentados no Capítulo 2. Em seguida,

os detalhes de uma partida *Very Small Size Soccer* são descritos no Capítulo 3.

A metodologia aplicada pode ser lida no Capítulo 4. Os resultados obtidos são apresentados no Capítulo 5 e as conclusões e trabalhos futuros no Capítulo 6.

# Capítulo 2

# Fundamentação Teórica

Detectar instâncias de objetos desenvolver estratégias, técnicas e modelos que os identifiquem e localizem. Como as diferentes tarefas de detecção têm objetivos e restrições diferentes, as dificuldades podem variar de acordo com o contexto encontrado em cada uma. Além de problemas comuns a outras áreas de visão computacional, como objetos sob diferentes pontos de vista, iluminações e variações entre objetos de mesma classe, os desafios na detecção de objetos incluem, mas não estão limitados a: orientação de um objeto e mudanças de escala (por exemplo, objetos pequenos), precisão da localização, oclusão, velocidade da detecção [11].

O progresso na detecção de objetos passou por dois períodos históricos: um "tradicional" – antes de 2014, e um período ancorado em avanços de aprendizado de máquina profundo – após 2014. Exemplos de técnicas do período tradicional que envolviam criação manual de características e métodos sofisticados para extrair informações das imagens incluem *Scale-Invariant Feature Transform* (SIFT)(1999) [12] e *Shape Contexts* (2000) [13], usadas para extrair características invariantes, enquanto classificadores como *Support Vector Machine* (SVM) [1] e AdaBoost [14] ajudavam a distinguir objetos de fundos. Esses métodos formaram a base da detecção de objetos antes da popularização do aprendizado de máquina profundo. Embora os detectores modernos baseados em aprendizado profundo tenham superado os métodos tradicionais em precisão, muitos conceitos dos detectores tradicionais, como modelos mistos e regressão de caixas delimitadoras, continuam a influenciar o campo [11].

## 2.1 Redes Neurais

As principais ideias por trás das redes neurais artificiais existem desde a década de 1940. McCulloch e Pitts [15] introduzem a noção de que a ativação de um neurônio dentro do cérebro representa o valor-verdade de uma proposição sobre o mundo exterior. Uma

vez que é possível implementar operações lógicas de negação, conjunção e disjunção por meio de conexões neurais e limiares apropriados, pode-se representar cada combinação lógica finita derivável das proposições elementares em uma rede neural. Após um período de poucos avanços, os trabalhos relacionados a redes neurais artificiais tiveram aumento significativo em quantidade e inovação; devido principalmente ao desenvolvimento de *hardware* que superou os gargalos de infraestrutura da época [16].

Em particular, o desenvolvimento e a popularização do algoritmo de *backpropagation* (retropropagação) teve um grande papel em impulsionar o campo de estudo em redes neurais. Dentre os trabalhos relevantes, destaca-se o artigo Representations by Back-Propagating Errors (1986) por Rumelhart, Hinton e Williams [17] , *backpropagation* onde apresentam treinamento eficiente de redes neurais multicamadas. O algoritmo calcula o gradiente da função de perda em relação aos pesos da rede, propagando os erros de saída retroativamente, camada por camada, ajustando os pesos para minimizar o erro.

No momento da data de entrega deste trabalho, têm-se duas abordagens principais para a implementação de algoritmos de detecção baseados em aprendizado profundo: detectores de um estágio — *one-stage* e de dois estágios — *two-stage*. Detectores de um estágio, como *You Only Look Once* (YOLO) [4] e *Single Shot Detector* (SSD) [18], realizam a detecção de objetos em uma única passagem pela rede, combinando a localização e a classificação em uma única etapa. Por outro lado, detectores de dois estágios, como R-CNN [19], dividem o processo em duas etapas: primeiro, geram propostas de regiões onde objetos podem estar presentes e, em seguida, classificam e refinam essas regiões.

## 2.2 Redes Neurais Convolucionais

Uma rede neural convolucional é um tipo de rede neural especializada para processamento de dados que possuem uma topologia de grade [20], sendo predominantemente formada por camadas convolucionais. Redes convolucionais são comuns no processamento de imagens, devido à natureza de uma imagem digital representada computacionalmente por matrizes de pixels. Elas processam regiões diferentes de uma imagem de forma independente e compartilham parâmetros relacionados entre essas regiões, ao longo de todo o processamento da imagem [21]. Como o nome sugere, camadas convolucionais realizam a operação matemática de convolução e, considerando uma imagem um vetor bidimensional $I$ de tamanho $m$ x $n$, podemos expressá-la através da Equação 2.1, sendo $i$ e $j$ inteiros positivos.

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{2.1}$$

A convolução $S$, é chamada de *feature map* ou mapa de características, e $K$ é um vetor multidimensional de parâmetros conhecido como *kernel*. É comum que a entrada de uma rede convolucional especializada em processamento de imagem seja uma imagem RGB. Um *kernel* 3×3 teria pesos 3×3×3 e seria aplicado aos três canais de entrada em cada uma das posições 3×3 para criar uma saída bidimensional que possui as mesmas dimensões da imagem de entrada (Figura 2.1). Para gerar vários canais de saída, o processo é repetido com diferentes pesos de *kernel* e os resultados anexados para formar um **tensor** tridimensional. Tensor pode ser entendido aqui como a generalização de uma matriz para dimensões arbitrárias. Nesse sentido, um vetor é um tensor unidimensional, uma matriz é um tensor bidimensional e um tensor tridimensional é uma grade tridimensional de números [21].

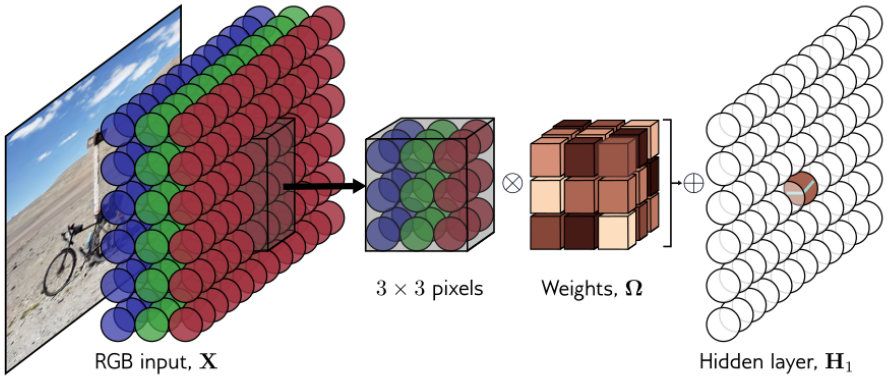



Figura 2.1: Convolução em uma imagem RGB.

Fonte: Reproduzido de [21].

Após uma convolução, acontece uma operação de *pooling*. Uma camada de *pooling* recebe cada saída do mapa de características da camada convolucional e prepara um novo mapa de características condensadas. Essa operação reduz as dimensões da entrada, diminuindo o número de parâmetros, e funciona de forma semelhante à camada convolucional, aplicando um filtro por toda a entrada. A diferença é que este filtro não tem nenhum peso. Em vez disso, ele aplica uma função de agregação nos valores da matriz de entrada, preenchendo uma matriz de saída a cada passo [22].

Existem vários tipos de agregação que podem ser usados nessa operação: *pooling* máximo — seleciona o pixel com valor máximo ao redor do pixel de referência de acordo com o tamanho do filtro, *pooling* médio — calcula o valor médio desses valores, *pooling* L2 — soma o quadrado dos valores [22][23]. Em todos os casos, o agrupamento ajuda a tornar a representação invariante para pequenas translações da entrada. Ser invariante a translações significa que para translações em pequena quantidade em uma entrada, os valores da maioria das saídas agrupadas tenham variações também pequenas [21].

Para muitas tarefas, o *pooling* é essencial para lidar com entradas de tamanhos variados. Por exemplo, na classificação de imagens de tamanho variável, a entrada da camada responsável pela classificação deve ter um tamanho fixo. Isso geralmente é alcançado com a variação no tamanho de um deslocamento entre regiões de *pooling* para que a camada de classificação sempre receba o mesmo número de valores resumidos, independentemente do tamanho da entrada [21].

Ao final de uma iteração, o algoritmo operante na rede convolucional escolhe um subconjunto aleatório dos dados de treinamento — *batch* — e calcula o gradiente a partir desses exemplos. Esse processo acontece para todas as amostras de treinamento. Uma única passagem por todo o conjunto de dados de treinamento é chamada de **época**.

Algoritmos iterativos de otimização são usados para encontrar o mínimo de uma função. No contexto de redes neurais, eles encontram parâmetros que minimizam a perda para que o modelo preveja com maior precisão os valores das entradas. A abordagem básica é escolher os parâmetros iniciais aleatoriamente, e depois fazer uma série de pequenas alterações que diminuam a média da perda. Cada mudança é realizada com base no gradiente da perda em relação aos parâmetros na posição atual [21].

Ao treinar modelos de aprendizado de máquina, procuramos os parâmetros que produzem o melhor mapeamento possível da entrada para a saída. Dado um conjunto de dados para treino com pares $x_i, y_i$ de entrada e saída, uma **função de perda** ou função de custo $L[\phi]$ retorna um único valor que descreve o afastamento entre as previsões do modelo $f[x_i, \phi]$ e as saídas $y_i$ correspondentes. Minimizar esse afastamento e, portanto, mapear as entradas de treino para as saídas o mais próximo possível é minimizar a **perda** que ocorre nas previsões do modelo [1].

## 2.3 Estimativa de pose

A estimativa de **pose**, também chamada de detecção de pontos de interesse ou pontos-chave, é uma tarefa que envolve a identificação da localização — posição e orientação — de pontos específicos em uma imagem. Os pontos-chave podem representar várias partes do objeto, como articulações, pontos de referência ou outras características distintivas. A estimativa de pose é uma boa escolha quando é necessário identificar partes específicas de um objeto numa cena e a sua localização em relação umas às outras [24].

Conceitos em estimativa de pose são úteis ao projeto desenvolvido durante este trabalho devido à necessidade do cálculo da orientação dos robôs em campo. Uma vez determinadas as coordenadas da frente e parte traseira do robô, o cálculo da sua orientação é trivial. Detalhes da implementação se encontram no Capítulo 4.

## 2.4 YOLO: You Only Look Once

*You Only Look Once* (YOLO), a arquitetura publicada por Joseph Redmon, Santosh Divvala, Ross Girshick e Ali Farhadi em 2016 propõe uma abordagem *one-stage* [4]. Até então, a principal técnica para detectar objetos em imagens era um problema de classificação extendido — *two-stage*; sendo necessário um segundo processamento de informação após a classificação, eliminando detecções duplicadas e outros refinamentos. Redes como *Region-based Convolutional Neural Network* (R-CNN) [19], apesar de trazerem maior eficiência em comparação com arquiteturas anteriores de redes convolucionais, ainda desperdiçam uma quantidade significativa de recursos devido à computação redundante de informação [11].

YOLO foi a primeira arquitetura a se utilizar de uma única rede convolucional para, simultaneamente, predizer caixas delimitadoras e probabilidades para as classes encontradas [4]. A generalização de objetos se torna mais tangível com a utilização de toda a informação da imagem durante o treinamento, ou seja, é possível não só codificar contexto mas também obter as características globais dos objetos em um único estágio.

O primeiro procedimento a ser feito na YOLO é a divisão da imagem de entrada em células com dimensões $S \times S$, e a especialização dessas unidades na detecção de um único objeto de centro $(x, y)$, caso $(x, y)$ esteja localizado dentro de sua área. Cada célula especialista prediz um número $B$ de caixas delimitadoras com um grau de confiança $P(Objeto) \times IoU(ref, pred)$, onde $P(Objeto)$ é a probabilidade da célula conter um objeto de interesse e $IoU(ref, pred)$ é o valor $IoU$ com base nos valores de referência e os valores preditos [4].

Sigla para *Intersect Over Union (IoU)* é uma métrica comumente utilizada para avaliar modelos de aprendizado de máquina para detecção de objetos. A Equação 2.2 expressa o cálculo de $IoU$, onde $Área(I)$ é a área da intersecção das entre a caixa delimitadora verdadeira e a prevista pela rede, e $Área(U)$ a sua união. A Figura 2.2 mostra visualmente o que ela significa.

$$IoU = \frac{Área(I)}{Área(U)} \tag{2.2}$$

Cada caixa delimitadora é descrita por 5 valores: $x, y, w, h$ e o grau de confiança que representa o IoU entre os valores previstos e de referência, onde $w$ é a largura da caixa e $h$ é a altura. As coordenadas $(x, y)$ representam o centro da caixa em relação aos limites da célula, e a largura e a altura são relativas à imagem inteira. Logo $x = \frac{x_{pixel}}{w}$ e $y = \frac{y_{pixel}}{h}$, cujos valores estão dentro de um intervalo entre 0 e 1.

Cada célula prevê apenas um conjunto de probabilidades de classe condicional C, $P(Classe_i|Objeto)$, independentemente do número de caixas B. No momento do teste,

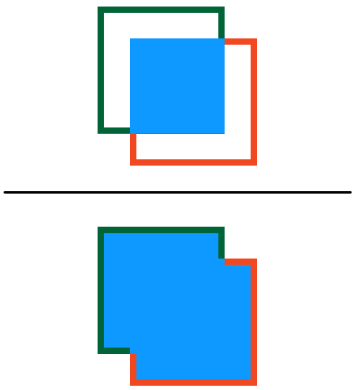


Figura 2.2: IoU - Intersecção sobre união

Fonte: Autor

multiplicam-se as probabilidades de classe condicional e os graus de confiança de caixa individual, o que nos dá valores de confiança específicos de classe para cada caixa. Esses valores codificam tanto a probabilidade dessa classe aparecer na caixa quanto o quão bem a caixa prevista se ajusta ao objeto.

O estudo comparativo de Terven e Cordova-Esparza [25] traz as principais modificações ao longo das primeiras versões de YOLO até YOLOv8 e YOLO-NAS. Entre os diferentes detectores de objetos, YOLO se destaca por equilibrar seu desempenho em termos de velocidade e precisão. Desde a sua criação, a família YOLO evoluiu através de múltiplas iterações, cada uma delas baseada nas versões anteriores para resolver limitações e melhorar desempenho.

Sua primeira versão publicada em 2016 [4], apesar de alcançar resultados mais rapidamente em comparação com as arquiteturas existentes, tinha desvantagens como: dificuldade em detectar objetos muito próximos uns dos outros e encontrar objetos com proporções não vistas nos dados de treinamento. Joseph Redmon e Ali Farhadi publicaram novo artigo no ano de 2017 descrevendo uma nova versão da YOLO — YOLOv2 — com modificações na arquitetura e melhoramentos, e no ano seguinte, apresentaram YOLOv3. Desde a sua quarta versão, a família de redes YOLO não tem mais autoria de Redmon e Farhadi. Os detalhes e melhorias que a versão YOLOv8, utilizada neste trabalho, não foram publicados em nenhum artigo oficial; ao invés, têm como base a documentação oficial, o repositório pertencente à empresa *Ultralytics* e páginas explicativas em seu website [24] [26].

YOLOv8 baseia-se na versão YOLOv5 também desenvolvida pela *Ultralytics*. Tem disponível cinco escalas de modelo: YOLOv8n (*nano*), YOLOv8s (*small*), YOLOv8m (*medium*), YOLOv8l (*large*), YOLOv8x (*extra large*); onde a largura e a profundidade

das camadas de convolução variam para atender aplicações específicas e requisitos de *hardware* [25]. O número de parâmetros para cada modelo pode ser visto na Tabela 2.1

Tabela 2.1: Relação modelo-número de parâmetros dos modelos YOLOv8-pose.

| **Modelo** | **Número de parâmetros** |
|---|---|
| YOLOv8n-pose | 3,3 M |
| YOLOv8s-pose | 11,6 M |
| YOLOv8m-pose | 26,4 M |
| YOLOv8l-pose | 44,4 M |
| YOLOv8x-pose | 69,4 M |

Outra característica importante da YOLOv8 é a variedade de tarefas que realiza, não apenas classificação e detecção de objetos mas também segmentação e estimativa de *pose*. Os nomes dos modelos variantes e as respectivas tarefas que desempenham podem ser vistos na Tabela 2.2.

Tabela 2.2: Relação variante-tarefa das variações da YOLOv8.

| **Variante** | **Tarefa** |
|---|---|
| YOLOv8n | Detecção |
| YOLOv8-seg | Segmentação |
| YOLOv8-pose | Estimativa de pose/pontos de interesse |
| YOLOv8-obb | Detecção com caixas delimitadores orientadas |
| YOLOv8-cls | Classificação |

Sua arquitetura pode ser dividida em três conjuntos principais de camadas que desempenham funções diferentes, nomeados ***backbone***, ***neck*** e ***head***; respectivamente análogas à coluna, pescoço e cabeça no corpo humano. Na Figura 2.3, observamos a estrutura simplificada das camadas.

*Backbone* é responsável por extrair características primitivas da imagem de entrada — convoluções seguidas de *pooling* —, capturando hierarquias entre essas características em diferentes escalas, (por exemplo, bordas e texturas). *Neck* atua como uma ponte entre *neck* e *head*, realizando operações de concatenação das saídas de camadas anteriores e integrando informações contextuais e espaciais — *pooling*. Já *head*, último conjunto de camadas da rede, é responsável por gerar as saídas da rede, como as caixas delimitadoras, valores de confiança da detecção, classes e localizações [28] [29].

YOLOv8 consiste em vinte e três camadas primárias, cada uma contendo subcamadas, que por sua vez contêm mais subcamadas. É composta por sete camadas "ConvModul",

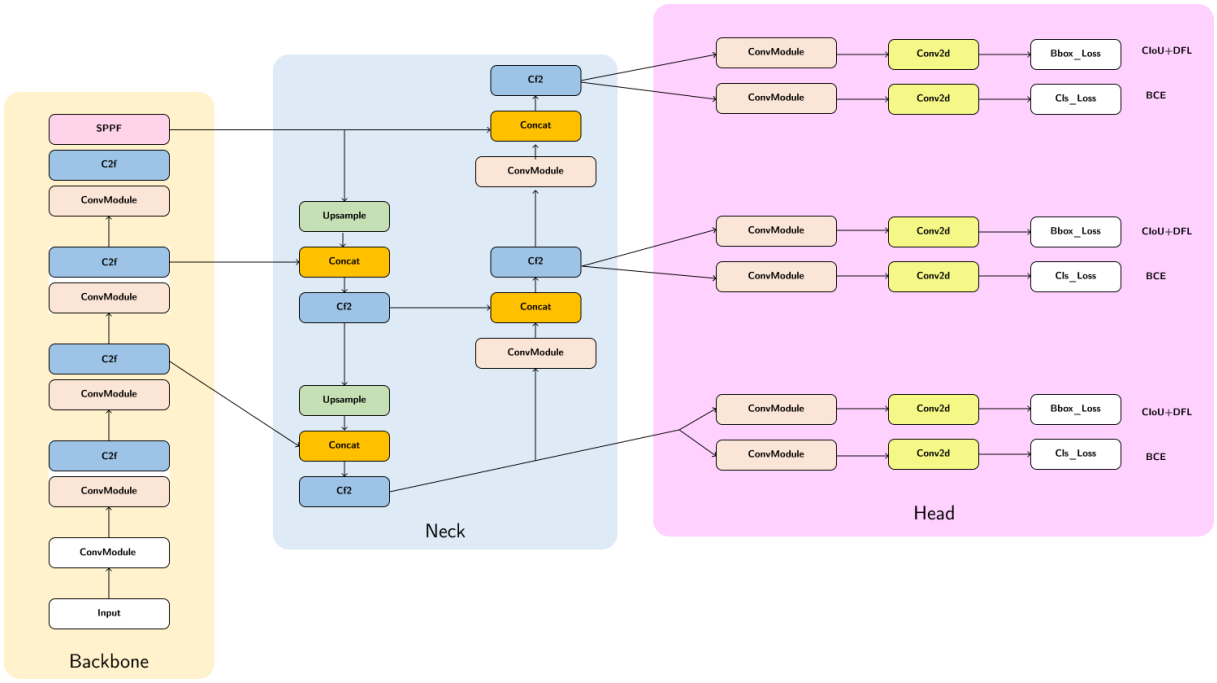



Figura 2.3: Arquitetura YOLOv8.

Fonte: Adaptado de [27]

oito camadas "C2f", uma camada "SPPF", duas camadas "Unsample", quatro camadas "Concat" e uma camada de detecção final, *head*, como mostra a Figura 2.3 [27].

ConvModul se refere ao módulo convolucional. Herdado de sua antecessora YOLOv5 com algumas alterações, o módulo C2f combina recursos de alto nível com informações contextuais para melhorar a precisão da detecção [25]. C2f é composto por dois módulos ConvModul e $n$ redes *DarknetBottleNeck*, onde $n$ é um parâmetro pré-definido da rede. SSPF se refere a *Spatial Pyramid Pooling* e visa acelerar o cálculo da rede agrupando características de diferentes escalas em um mapa de características de tamanho fixo [27]. Na arquitetura ilustrada na Figura 2.3, Upsample especifica uma operação de *upsampling* que duplica as dimensões espaciais da camada anterior. As operações Concat são então usadas para empilhar mapas de características em profundidade [26].

Para cálculo da perda na predição de caixas delimitadoras, YOLOv8 usa as funções $CIoU$ (uma variação de $IoU$) e $DFL$ (*Distribution Focal Loss*). Considerando $\gamma$ a coordenada (x ou y) correta da caixa delimitadora, $y_i$ e $y_{i+1}$ o valor antecessor e sucessor de $\gamma$, e $P$ a probabilidade prevista pela rede de $\gamma_i$ e $\gamma_{i+1}$, $DFL$ é definida pela Equação 2.3.

$$DFL(P_i, P_{i+1}) = -((\gamma_{i+1} - \gamma)log(P_i) + \gamma - \gamma_i)log(P_{i+1})). \tag{2.3}$$

Já para a perda na classificação, opta pela função de entropia cruzada — comumente aplicada para quantificar a diferença entre duas distribuições de probabilidade. Sendo $H$

a entropia, $N$ o número de amostras, $y_i$ a classe verdadeira da amostra $i$, e $p_i$ a classe prevista pela rede conforme a Equação 2.4.

$$H(y,p) = -\frac{1}{N}\sum_{i=1}^{N}[y_i log(p_i) + (1-y_i) log(1-p_i)] \tag{2.4}$$

YOLOv8 pode ser executada a partir de interface de linha de comando ou também pode ser instalada como um pacote pip — gerenciador de pacotes Python. Além disso, seu repositório agrega várias ferramentas auxiliares e integrações para rotulagem, treinamento e modificações; visto que é um projeto *open-source* sob a licença AGPL-3.0 (GNU *Affero General Public License* versão 3) [26].

# Capítulo 3

# *Very Small Size Soccer*

## 3.1 Cenário

Uma partida *Very Small Size Soccer* ocorre em um campo composto por uma chapa plana e rígida de madeira medindo 150cm × 130cm, em cor preta fosca e não-reflexiva com laterais medindo 5cm de altura por 2.5cm de largura, conforme mostrado na Figura 3.1. O campo deve ser localizado em ambiente interno, com boa iluminação para permitir o melhor reconhecimento de cores possível. Uma partida deve ser disputada por dois times, cada um constituído de, no máximo, três robôs com dimensões máximas de 7.5cm × 7.5cm × 7.5cm, sendo que um dos robôs pode ser o goleiro. Cada robô deve ser totalmente independente, com baterias e motores contidos em si. Somente comunicação sem fio é permitida para todos os tipos de interação entre o computador *host* e um robô [30].

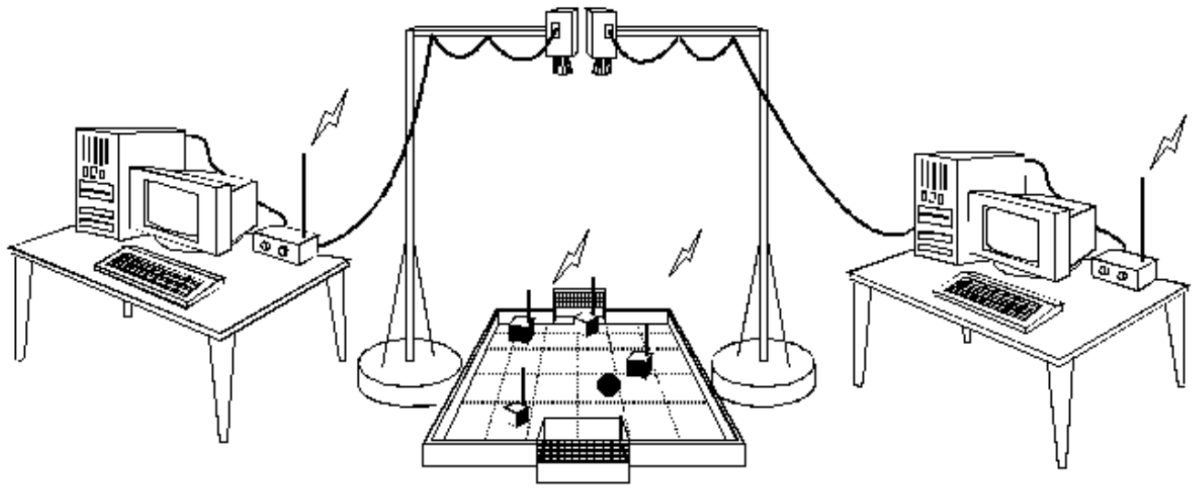


Figura 3.1: Sistema VSSS completo.

Fonte: Regras *Very Small Size Soccer* (VSSS) - LARC 2023 [30]

Um único computador pode ser usado por equipe, sobretudo dedicado ao processamento visual e para reconhecimento das posições. Em via de regra, a parte superior (camisa) dos robôs não deve conter as cores laranja, branco ou cinza e também não deve ser colorida com cores diferentes da cor preta e da cor do time. Uma cor, que geralmente alterna entre azul e amarelo, identificará times distintos e deverá ser posicionada em uma região do topo do robô. A cor de identificação de cada time poderá mudar de partida a partida, portanto a etiqueta usada deverá ser destacável [30].

Para identificar o robô e a bola em campo, um sistema de visão computacional é utilizado. A câmera (suspensa a uma altura não inferior a 2 m) é fixada acima do meio de campo, incluindo sua linha central, de tal forma que não tenha que ser movida depois da troca de lado após o meio tempo. Se ambos os times desejarem manter suas câmeras acima do círculo central do campo, elas deverão ser posicionadas lado a lado, equidistantes da linha central e o mais perto possível uma da outra [30].

Para além do tamanho dos robôs, competições VSSS, em via de regra, não possuem especificações rígidas de como projetar e confeccionar *hardware* e *software* dos sistemas robóticos das equipes participantes.

## 3.2 Características dos objetos

### 3.2.1 Robôs

Cada robô da UnBall utiliza identificação segundo a Figura 3.2. A equipe fixa um quadrado rosa de EVA, ou camisa do robô, com área próxima à da parte superior do robô (visível para a câmera), onde a forma geométrica inscrita no EVA diferencia os robôs por índices: 0, 1 e 2.

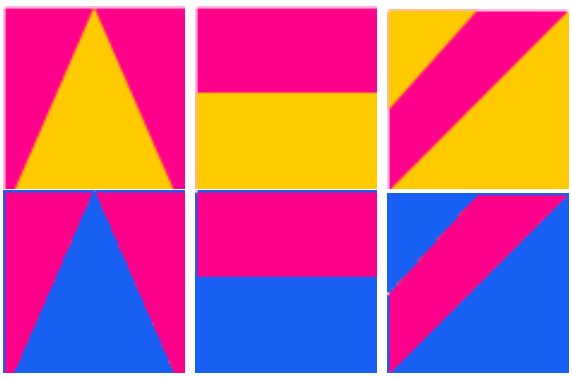


Figura 3.2: Camisas UnBall 0, 1 e 2, respectivamente, nas duas cores possíveis segundo as regras VSSS.

Fonte: Arquivos da UnBall

### 3.2.2 Bola

Utiliza-se uma bola de golfe que segue as regras estabelecidas para uma partida padrão VSSS [30]: cor laranja, com aproximadamente 42.7mm de diâmetro e 46g de massa.

Figura 3.3: Bola utilizada em partidas VSSS.

Fonte: Arquivos da UnBall

## 3.3 Sistema geral da UnBall

O sistema de *software* escolhido para investigação e comparação neste trabalho — *Main System* — é responsável por obter as coordenadas em campo dos robôs e da bola, e orientação dos robôs. Esses são os objetos de interesse na partida. Dentre os quatro módulos principais: Visão, Estratégia, Controle e Comunicação (Figura 3.4); o objeto de estudo aqui se volta para o subsistema de Visão. Sua entrada é uma sequência de quadros que advém da câmera posicionada a 2 metros de distância do campo da partida, conforme a Figura 3.1.

O *software* é desenvolvido em Python, tendo a maioria de suas funcionalidades implementadas utilizando a biblioteca OpenCV. Quatro etapas principais compõem o algoritmo de detecção do MS: pré-processamento, segmentação, identificação e rastreamento. Após o processamento das imagens, o subsistema de Visão entrega para o subsistema de Estratégia a *pose* (coordenadas no campo e orientação) dos robôs, coordenadas da bola, assim como suas taxas de variação (velocidade linear e velocidade angular).

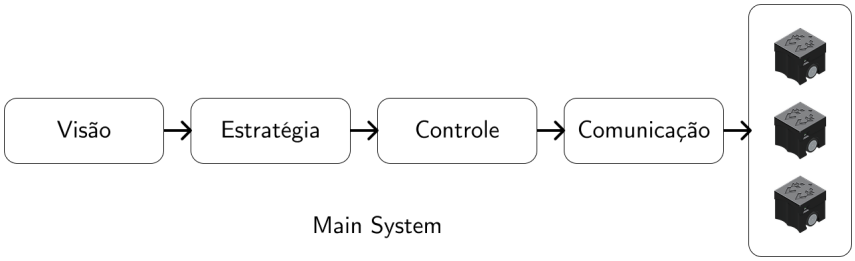



Figura 3.4: Módulos do *Main System* (MS).

### 3.3.1 Pré-processamento

O primeiro procedimento realizado em um quadro é o seu redimensionamento, onde é cortado de forma retangular ou via homografia (método `cv2.warpPerspective` da biblioteca OpenCV) a depender da opção selecionada pela interface gráfica. Os pontos de corte devem ser escolhidos pelo usuário para os dois tipos de corte. Logo em seguida, é feita a conversão do sistema de cores RGB para HSV.

(a) Corte retangular

(b) Corte por homografia

Figura 3.5: Visualização dos tipos de cortes do campo no *Main System* (MS).

### 3.3.2 Identificação e rastreamento

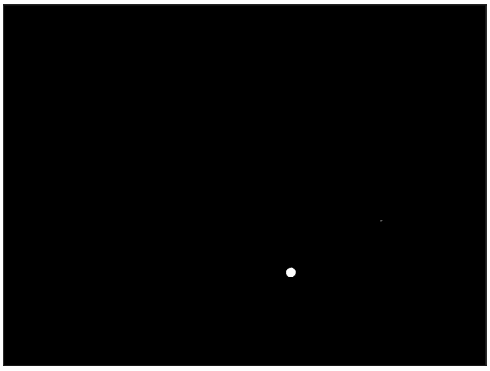
A segmentação é feita através da criação de máscaras (filtros) com intervalos de valores HSV das cores que identificam os objetos em campo. É usada a função `cv2.inRange` onde, dada uma imagem com sistema de cores HSV, se verifica o pertencimento de cada pixel ao intervalo determinado.

As localizações dos objetos são obtidas a partir das máscaras criadas na etapa anterior. Obtendo-se os contornos presentes na imagem, o fluxo de execução segue na ordem: tentativa de identificar a bola e tentativa de identificar os robôs do time aliado. A busca de contornos em uma imagem é uma boa estratégia para análise de forma, detecção e reconhecimento de objetos. Podem ser identificados por segmentos contínuos de quais-

quer tipos de objetos. Para seleção do objeto usamos a borda que, através de operações de abertura e dilatação, permitem a extração das características de objetos distintos do campo. A identificação da bola é a mais simples.

É usada a função `cv2.minEnclosingCircle` que recebe dois pontos como parâmetro e retorna um círculo com a área mínima que os envolve. Já a identificação dos robôs do time aliado exige uma quantidade maior de operações e uma lógica diferente. Iterando por cada elemento conectado, contorno, é feita uma tentativa de identificação de camisa através de uma função que retorna a posição em metros e a orientação em radianos da camisa com base no menor retângulo que se inscreve no contorno.

(a) Elementos em campo



(b) Segmentação da bola

(c) Robôs aliados

Figura 3.6: Segmentação no *Main System* (MS).

Detectada uma camisa, o sistema verifica se a camisa é do time aliado ou do time adversário. A Figura 3.6a mostra a segmentação de todos os elementos presentes no campo, na Figura 3.6b, a segmentação da bola, e na Figura 3.6c, temos a segmentação dos robôs do time aliado (UnBall).

As operações que buscam identificar se e qual delas está presente em uma camisa seguem na ordem: seleciona a forma principal, calcula o centro do contorno principal, define qual o polígono da figura principal, computa o identificador com base na forma e no número de contornos internos, e por último, calcula o ângulo com base no vetor entre o centro do contorno principal e o centro da camisa.

Dentre os contornos identificados em um quadro, todos que não forem identificados com formas características do time aliado, são categorizados como robôs do time inimigo. É então enviada uma mensagem para o módulo de Estratégia contendo novos valores para *pose* de robôs e bola, além de uma lista que indica se os Robôs e Bola foram identificados ou não.

# Capítulo 4

# Metodologia

Pela sua reputação em equilibrar, desempenho, velocidade e precisão [25], foi escolhida a arquitetura *You Only Look Once* (YOLO) em sua versão v8 para a detecção de objetos em uma partida IEEE *Very Small Size Soccer* (VSSS). Recebendo como entrada uma imagem do campo em uma partida VSSS, deseja-se obter as coordenadas no campo e sua orientação (no caso de robôs) dos objetos de interesse.

## 4.1 Banco de imagens

O primeiro passo em uma aplicação com aprendizado de máquina é obter um banco de dados hábil para treinamento. Na ausência de um banco de imagens existente para o problema, foi necessária a extração e preparação de imagens que representassem a entrada de um sistema de visão em uma partida VSSS. Foram usados vídeos gravados na IRONCup 2020 para obtenção de amostras de partidas jogadas pela UnBall com as camisas mostradas na Figura 3.2. Após a obtenção das imagens, foi realizado o pré-processamento que envolve: recorte, edição e redimensionamento. Como é preciso haver identificação dos robôs tanto com a cor amarela, quanto com a cor azul, foram escolhidos vídeos que contemplassem as duas situações.

Para a anotação das amostras, optou-se por utilizar a ferramenta *Roboflow Annotate* [31], onde existem diversos tipos de projeto focados em problemas diferentes de visão computacional. O tipo compatível com o projeto deste trabalho se chama *Keypoint Detection*. A ferramenta possui opções para registro das classes que serão identificadas no banco de dados a ser criado, onde cada classe possui um nome, uma cor associada e um esqueleto.

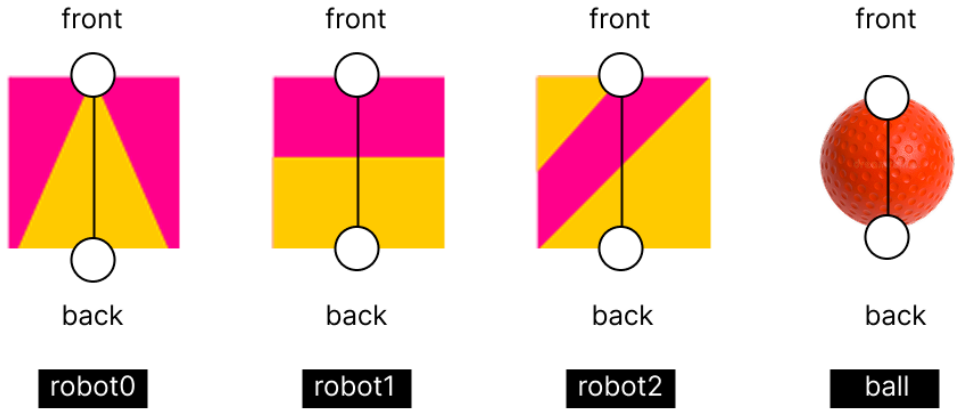



Figura 4.1: Esqueleto do robô e da bola.

Para todos os objetos, robôs e bola, o esqueleto (segmento de reta) desenhado pode ser visto na Figura 4.1, onde *front* representa a frente do objeto e *back* representa a parte traseira. Abaixo de cada representação se encontram os rótulos de cada objeto nas anotações, respectivamente: `robot0` para o Robô 0, `robot1` para o Robô 2, `robot2` para o Robô 2 e `ball` para Bola. Foram escolhidos tais pontos para que fosse possível calcular a orientação do objeto posteriormente à sua localização.

A orientação do robô, ângulo $\theta$ em relação ao eixo horizontal, pode ser calculada conforme a Equação 4.1.

$$\theta = \arctan \frac{y_{front} - y_{back}}{x_{front} - x_{back}} \tag{4.1}$$

A versão v1 do banco de imagens `ballnet_dataset` [32] é a utilizada nesse trabalho. Composto por 653 imagens no total, a separação das amostras segue descrita na Tabela 4.1. *Roboflow Annotate* [31] oferece diversas funcionalidades para enriquecer bancos de imagens, como pré-processamento automático e *data augmentations* (conjunto de técnicas que usa dados pré-existentes para criar novas amostras, são exemplos: rotação da imagem, adição de ruído, conversão do espaço de cores, entre outras), porém não foram utilizadas para a criação do banco.

Tabela 4.1: Separação das amostras no banco de imagens.

| **Amostra** | **Quantidade de imagens** |
|---|---|
| Treino | 437 |
| Validação | 131 |
| Teste | 85 |

Vale destacar que a orientação da bola não foi calculada. Por ser um objeto de aparência homogênea, não há frente nem trás; logo, foram escolhidas quaisquer semirretas entre duas extremidades da circunferência.

## 4.2 Treinamento do modelo

O treinamento foi realizado em computador com sistema operacional Windows 11, utilizando o interpretador Python CPython 3.12.8. *Hardware* composto por um Processador AMD Ryzen Threadripper 3970X, com 32 núcleos físicos e 64 núcleos lógicos, com até 4,5GHz, complementado por duas GPU NVIDIA GeForce RTX 3080 (arquitetura Ampere), com 10,74 GiB de memória dedicada e 8.704 núcleos CUDA, gerenciadas pela versão CUDA 12.6. O sistema conta com 192 GiB de memória RAM e um armazenamento total de aproximadamente 8 TiB.

Como afirmado na Seção 2.4, o modelo YOLOv8-pose possui cinco variantes com diferentes escalas (*nano, small, medium, large, extra-large*). Foram feitos treinamentos subsequentes para cada uma delas, com variação no número de épocas entre 100 e 1000 épocas em intervalos de 100 (100, 200, 300, assim por diante). Como referencial para avaliação dos treinamentos, observaram-se os valores de perda para as coordenadas dos pontos de interesse para cada um.

YOLOv8-pose calcula a perda para os pontos de interesse usando *Object Keypoint Similarity* (OKS) [26], função expressa na Equação 4.2, onde $d_k$ é a distância euclidiana ao quadrado entre o ponto de interesse $k$ previsto e o valor de referência; $s_k$ é a escala desse ponto, que surge da integração sobre uma gaussiana isotrópica bidimensional (idêntica em todas as direções) com desvio padrão de 0,25; *Area* é a área da caixa delimitadora para o objeto de interesse a qual o ponto pertence, usada para normalizar a distância $d_k$ em relação ao tamanho do objeto; e $\epsilon$ é um valor adicionado para evitar divisão por zero. $\delta(v_k > 0)$ é uma função indicadora que retorna 1 se o ponto de interesse está visível ou oculto, e 0 caso contrário.

$$\text{OKS} = 1 - \frac{\sum_{k=1}^{K} \exp\left(-\frac{d_k^2}{2s_k^2 \cdot (\text{Area}+\epsilon)}\right) \cdot \delta(v_k > 0)}{\sum_{k=1}^{K} \delta(v_k > 0) + \epsilon} \tag{4.2}$$

## 4.3 Obtenção de dados no *Main System*

Para comparação e avaliação posteriores ao treinamento da rede, o mesmo conjunto de teste foi submetido à detecção do MS. Dada a primeira imagem do conjunto, após a

configuração ser determinada e todos os elementos identificados, o sistema foi modificado para ler automaticamente a próxima imagem do conjunto.

Tabela 4.2: Valores dos parâmetros HSV para a camisa amarela

| **Parâmetro** | **Intervalo de valores para a camisa amarela** | | |
|---|---|---|---|
| | **Segmentação de elementos** | **Segmentação da bola** | **Segmentação do time** |
| Hue | [5:179] | [0:21] | [2:76] |
| Saturation | [45:255] | [141:255] | [121:255] |
| Value | [203:255] | [177:255] | [0:255] |

Tabela 4.3: Valores dos parâmetros HSV para a camisa azul

| **Parâmetro** | **Intervalo de valores para a camisa azul** | | |
|---|---|---|---|
| | **Segmentação de elementos** | **Segmentação da bola** | **Segmentação do time** |
| Hue | [0:179] | [0:21] | [88:115] |
| Saturation | [85:255] | [141:255] | [0:255] |
| Value | [206:255] | [177:255] | [0:255] |

Os valores apresentados na Tabela 4.2 e na Tabela 4.3 são utilizados como parâmetros de exclusão dos objetos cujas cores estão fora do intervalo descrito em [H:S:V mínimo, H:S:V máximo], ou seja, temos apenas os objetos cujas cores estão dentro do intervalo.

# Capítulo 5

# Resultados

## 5.1 Treinamento

Realizados os treinamentos para cada uma das escalas disponíveis para a YOLOv8-pose (*nano, small, large* e *extra large*) alterando-se o número de épocas, começando com 100 e terminando em 1000, com um intervalo de 100 épocas entre cada treinamento, e considerando como critério o menor valor de perda nos pontos de interesse, observou-se que o modelo que obteve melhor desempenho foi YOLOv8x-pose (*extra large*).

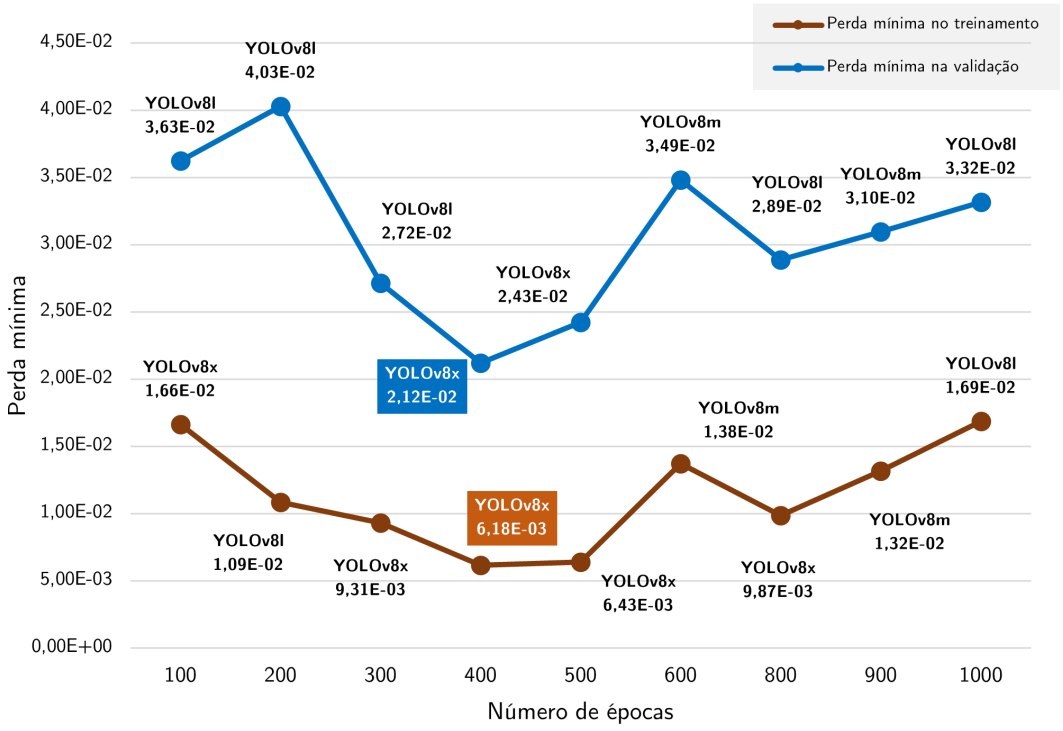



Figura 5.1: Perda mínima no treinamento e validação por número de épocas.

O gráfico da Figura 5.1 mostra a perda mínima dos pontos de interesse (frente e trás) obtida em cada treinamento e a Figura 5.2 ilustra o comportamento dessa perda durante o treinamento por 400 épocas para o modelo YOLOv8x-pose. Escolhido como o modelo principal para resolução do problema apresentado, o modelo está disponível em repositório público na plataforma *Hugging Face*[33].

Uma das métricas para medir a eficiência de um modelo para detectar objetos é a média da precisão média — *Mean Average Precision* (mAP). A precisão quantifica a proporção de verdadeiros positivos entre todas as previsões positivas. Como aqui os pontos de interesse são o foco, um caso de verdadeiro positivo é alcançado quando o modelo prediz corretamente um ponto de interesse, e falso positivo quando prediz erroneamente tal ponto, caso o tenha identificado.

Quando se calcula a mAP para diferentes limiares de OKS — variando de 0,50 a 0,95 e aumentando em 0,05 por vez — chamamos essa métrica de **mAP50-95** [24]. Isso significa que, ao processar e prever as localizações para todo o conjunto de treinamento, o modelo calcula mAP considerando verdadeiros positivos casos onde o valor de OKS para cada objeto detectado é igual ou maior 0,50; 0,55; 0,60 e assim por diante; até 0,95.

Ao final do treinamento da YOLOv8x-pose por 400 épocas, obteve-se o valor de mAP50-95 igual a 0,99. Esse valor indica um bom desempenho da *Ballnet Pose* na estimativa de pose.

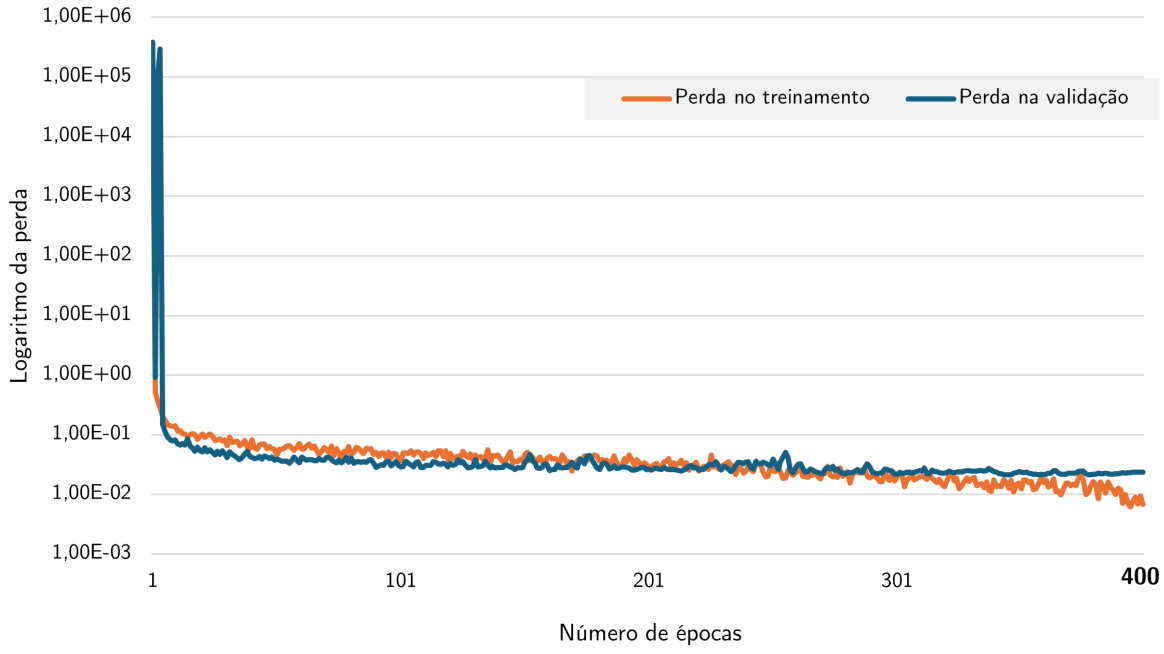



Figura 5.2: Perda para o modelo YOLOv8x em escala logarítmica.

Junto com o arquivo do modelo, também foi disponibilizado o módulo `BallnetPose`, onde é definida a classe `BallnetPose` e o método principal é `detect`, codificado em Python que implementa a inferência das coordenadas dos objetos, juntamente com o cálculo das orientações dos robôs. A saída é formatada como mostrado no código em *Listing 1*. Detection se refere a uma tupla nomeada, assim como Robot e Ball. O código *Listing 2* descreve a definição dos objetos.

**Listing 1** Saída esperada do método `detect` da classe `BallnetPose`.

```
[Detection(
  robots=[
    Robot(id=1.0, center_x=100, center_y=601,
    orientation=49.86),
    Robot(id=2.0, center_x=560, center_y=419,
    orientation=45.76),
    Robot(id=0.0, center_x=60, center_y=603,
    orientation=50.19)
  ],
  ball=Ball(center_x=192, center_y=393)
)]
```

**Listing 2** Definições dos objetos em Python.

```
Detection=namedtuple('Detection', ['robots', 'ball'])
Robot=namedtuple('Robot', ['id','center_x', 'center_y',
'orientation'])
Ball=namedtuple('Ball', ['id','center_x', 'center_y'])
```

## 5.2 Acurácia

Para comparar o desempenho da detecção de ambos os sistemas apresentados — *Main System* e *Ballnet Pose* — calculou-se o erro médio para as coordenadas $(x, y)$ em pixels do centro dos objetos em relação às coordenadas $(x_0, y_0)$ de referência, como expresso na Equação 5.1.

$$\text{Erro}_{\text{centro}} = \frac{|(x - x_0) + (y - y_0)|}{2} \tag{5.1}$$

Também foi calculado o erro absoluto no cálculo da orientação $\theta$ em radianos dado o respectivo $\theta_0$, ao longo do conjunto de imagens de teste com 85 imagens, vide Equação 5.2.

$$\text{Erro}_{\text{orientação}} = |\theta_0 - \theta| \tag{5.2}$$

Vemos pelas Figura 5.3 e Figura 5.4 que *Ballnet Pose* desempenha, em média, melhor que o MS na estimativa do centro dos robôs e também na estimativa da orientação.

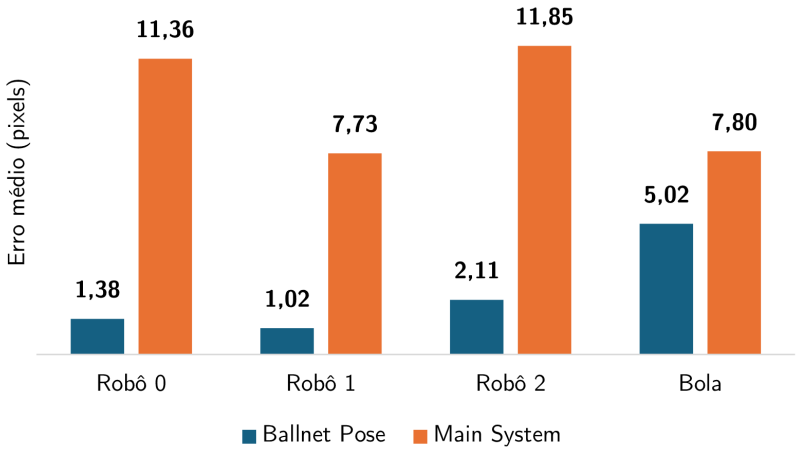



Figura 5.3: Erro médio do centro dos objetos.

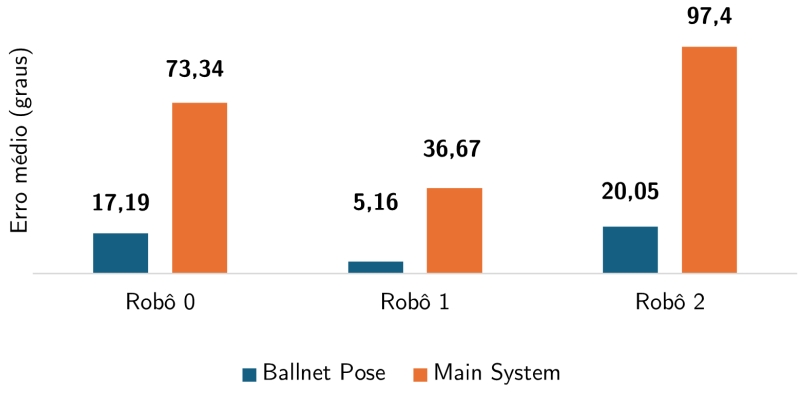


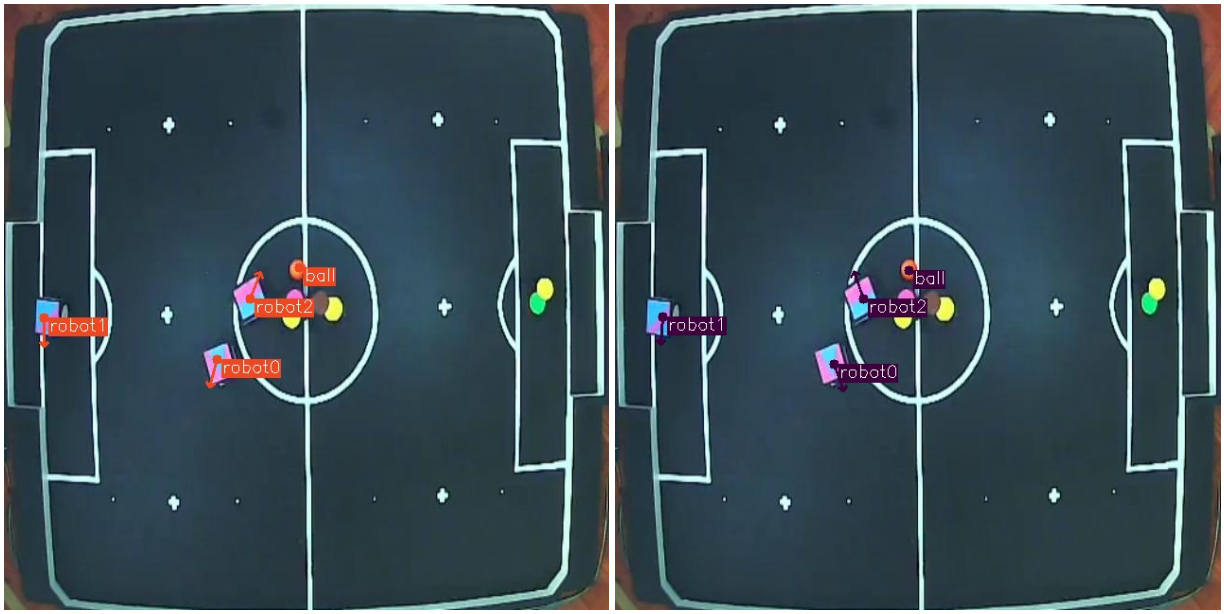

Figura 5.4: Erro médio da orientação dos robôs.

Ambos os sistemas têm dificuldade em localizar o Robô 2 e facilidade em detectar o Robô 1. Enquanto o maior erro médio para a *Ballnet Pose* acontece na detecção da bola,

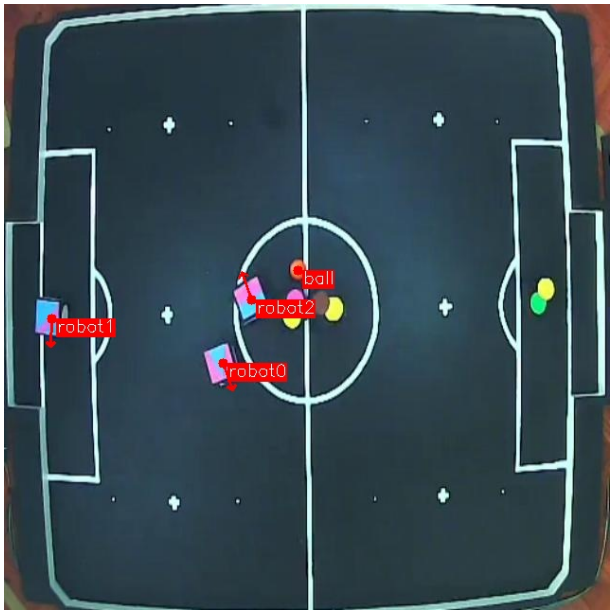
para o MS, a localização do centro do Robô 2 é a que mais se distancia do centro de referência.

A Figura 5.4 mostra que a *Ballnet Pose* erra a orientação do Robô 0 em média por 17,19 graus; 5,16 graus quanto à orientação Robô 1; e 20,05 graus quanto à orientação Robô 2. Observa-se que para o MS esses valores são, respectivamente: 73,34 graus para o Robô 0; 36,67 graus para o Robô 1; e 97,40 graus para o Robô 2.

Um comparativo visual pode ser observado na Figura 5.5, onde foram encontradas "boas" detecções dos sistemas, vide Figura 5.5a e Figura 5.5b, e uma imagem de referência com as detecções corretas em Figura 5.5c.



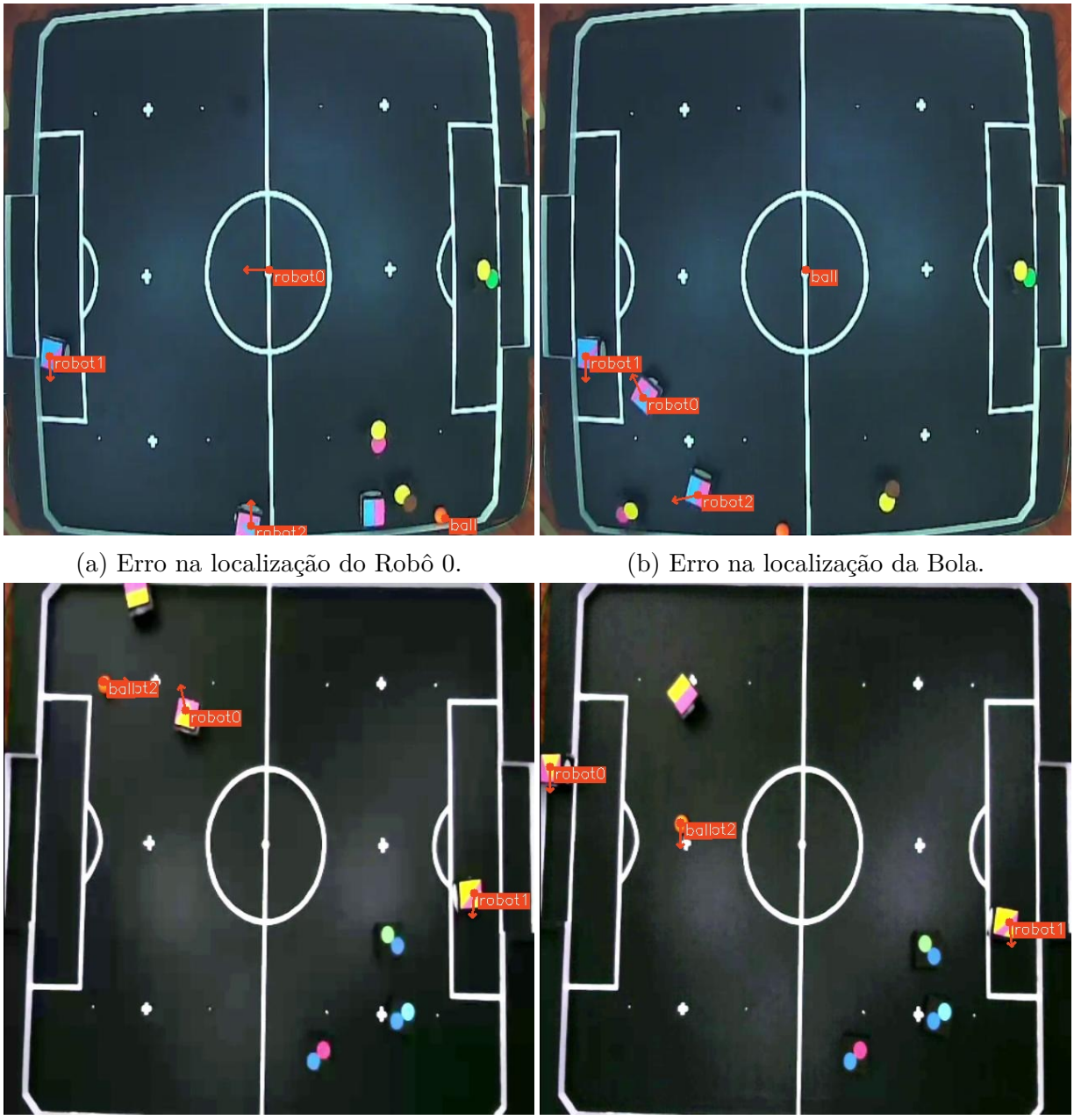

(a) *Main System* (MS) (b) *Ballnet Pose*




(c) Referência

Figura 5.5: Comparação entre a detecção do *Main System* (MS) e *Ballnet Pose*.

O método para determinar se uma detecção foi boa ou não consistiu em observar as discrepâncias entre localizações e direções inferidas e as de referência. Caso a diferença não fosse visualmente grande, a detecção foi considerada boa. Enquanto o MS obteve localizações do centro próximas da referência, a orientação difere significativamente da orientação correta para os Robôs 0 e 2. Em geral, *Ballnet Pose* se aproxima da referência tanto em relação às coordenadas dos centros quanto em relação às orientações dos robôs.




(a) Erro na localização do Robô 0.

(b) Erro na localização da Bola.

(c) Erro na identificação do Robô 2.

(d) Erro na identificação do Robô 2.

Figura 5.6: Erros significativos na detecção do Main System.

O MS atribuiu localizações erradas para a Bola (Figura 5.6b) em 1 imagem do conjunto de teste, não identificou o Robô 0 em outra (Figura 5.6a), atribuindo para ele o rótulo de Robô 2 erroneamente. O sistema atribuiu à bola o rótulo de Robô 2 em duas imagens (Figura 5.6c e Figura 5.6d). *Ballnet Pose* não detectou a Bola em uma das imagens de teste Figura 5.7a e não identificou o Robô 2 em outra Figura 5.7b.

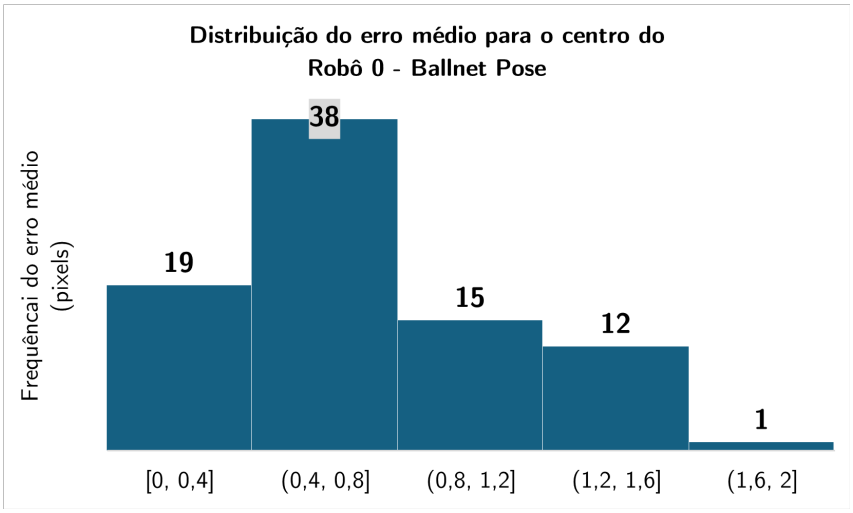

(a) Erro na detecção da Bola. (b) Erro na detecção do Robô 2.


(c) Erro na orientação do Robô 2. (d) Erro na orientação do Robô 2.

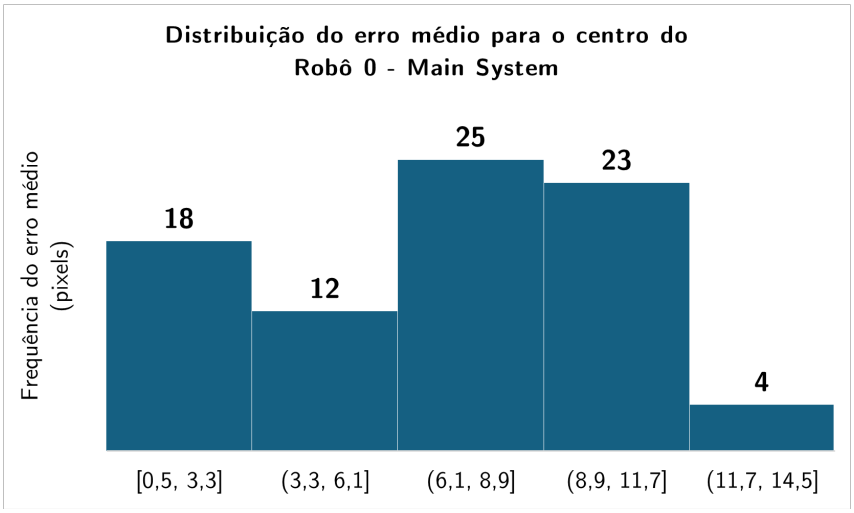
Figura 5.7: Erros significativos na detecção da Ballnet Pose.

Na Figura 5.8 pode-se observar que a maioria dos erros na *Ballnet Pose* para o centro do Robô 0 — 38 detecções — se dão no intervalo entre 0,4 e 0,8 pixels, não ultrapassando 2 pixels de erro ao longo das imagens de teste. Já para o MS, em geral os erros são maiores que na *Ballnet Pose*, chegando a ter quatro erros entre o intervalo 11,7 e 14,5. Uma observação quanto a duas detecções da *Ballnet Pose*, vistas na Figura 5.7c e Figura 5.7d, é: o centro do Robô 2 foi localizado de forma satisfatória, porém a orientação divergiu significativamente em relação à referência devido ao corte da imagem, que oculta parte

do Robô.



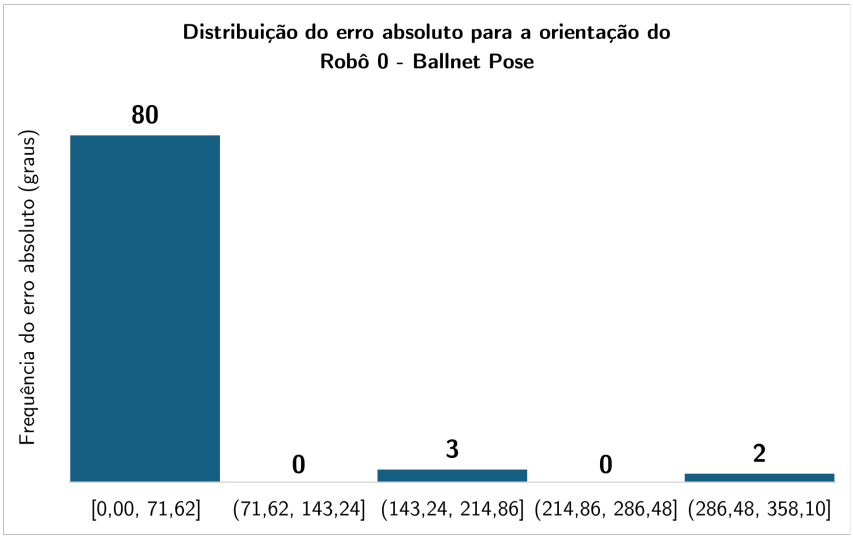

(a) *Ballnet Pose*



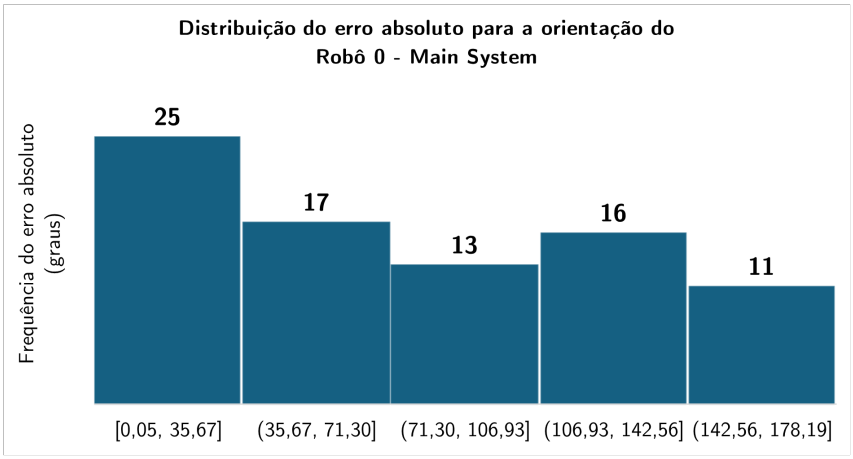

(b) *Main System* (MS)

Figura 5.8: Frequência do erro médio para o centro do Robô 0.

A Figura 5.9 mostra as distribuições para a orientação do Robô 0 nos sistemas apresentados. Cerca de 80 detecções na *Ballnet Pose* estavam entre 0 e 71,62 graus; tendo 5 detecções acima desse intervalo. No MS, assim como nas localizações do centro do Robô, os erros nas orientações também são maiores do que os erros da *Ballnet Pose*; sendo que 25 detecções estavam entre 0,05 e 35,67 graus e o restante está distribuído nos intervalos (35,67; 71,30], (71,30;106,93], (109,93; 142,56], e (142,56; 178,19].



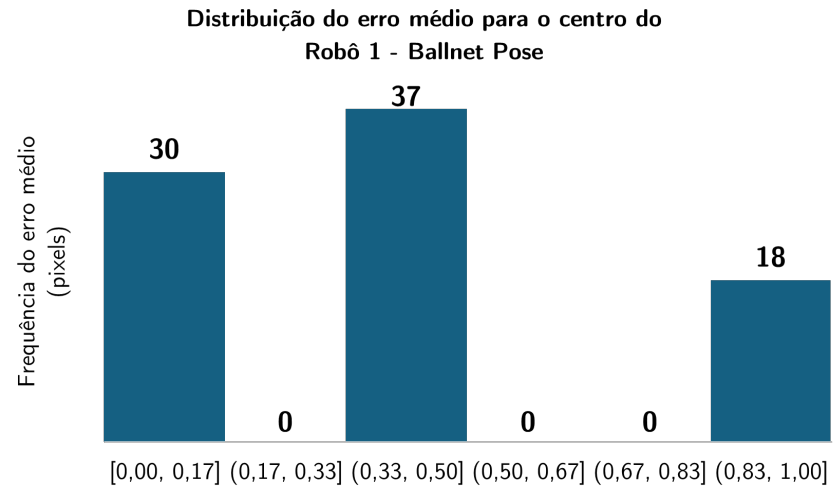

(a) *Ballnet Pose*



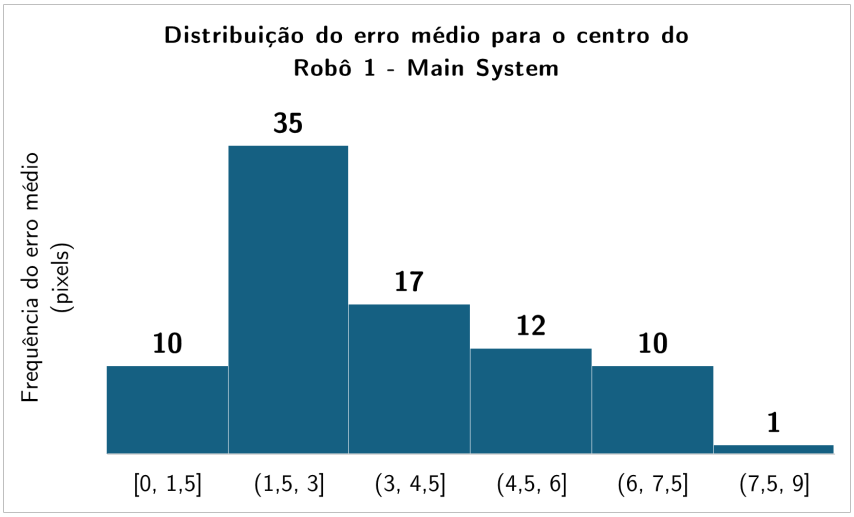

(b) *Main System* (MS)

Figura 5.9: Frequência do erro absoluto para a orientação do Robô 0.

De forma similar às detecções do Robô 0, *Ballnet Pose* comete erros que não ultrapassam o valor de 1 pixel para o centro do Robô 1 (Figura 5.10a). Diferente das detecções do Robô 0, para o MS, não houve erros maiores que 10 pixels, onde a maioria se encontra no intervalo entre 1,5 e 3 pixels (Figura 5.10b).



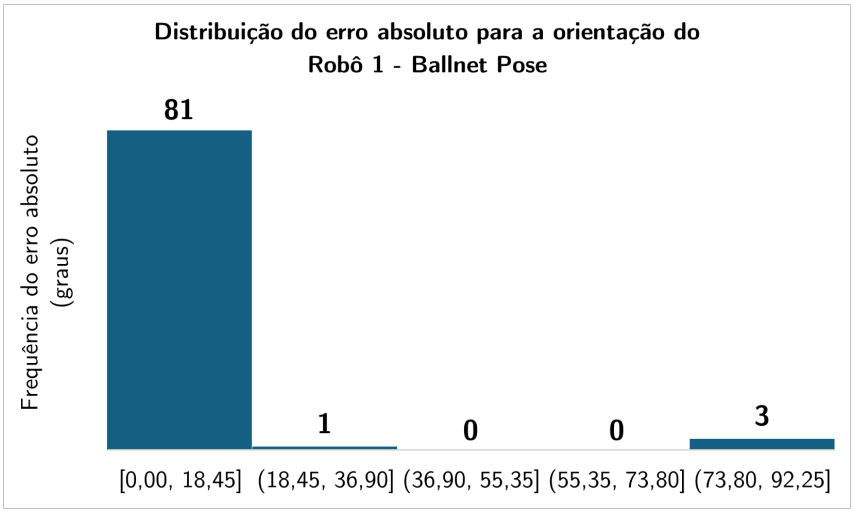

(a) *Ballnet Pose*



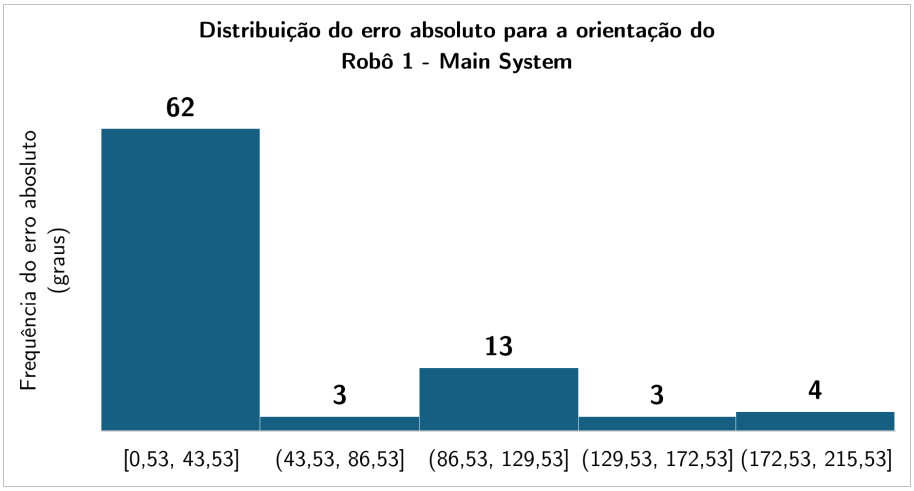

(b) *Main System* (MS)

Figura 5.10: Frequência do erro médio para o centro do Robô 1.

As distribuições para as orientações do Robô 1 podem ser vistas na Figura 5.11. Podemos ver que, para as orientações do Robô 1, *Ballnet Pose* concentra a maioria dos erros no intervalo [0; 18,45] — 82 valores, com 3 valores entre 73,80 e 92,25 graus, e 1 valor entre 18,45 e 38,90 graus. Para o Robô 1, o MS concentra 62 valores entre 0,53 e 43,53 graus, e seus maiores valores estão entre 172,53 e 215,53 graus.



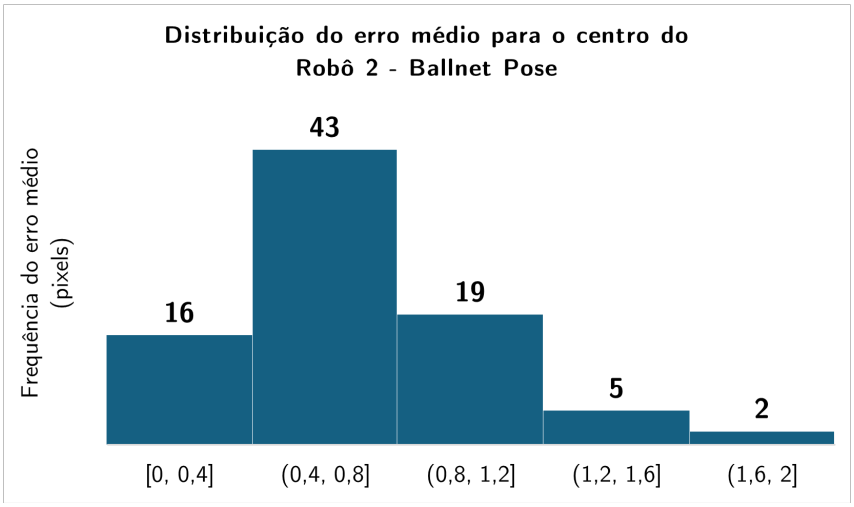

(a) *Ballnet Pose*



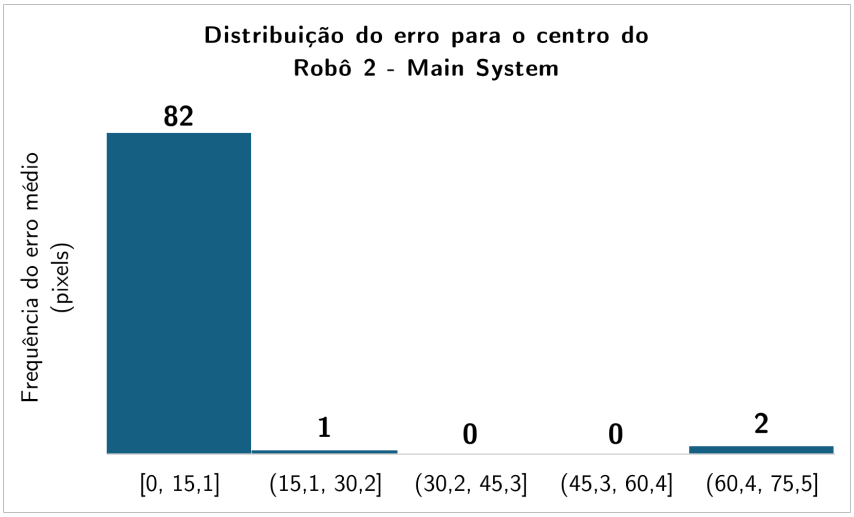

(b) *Main System* (MS)

Figura 5.11: Frequência do erro absoluto para a orientação do Robô 1.

Os maiores erros em relação às coordenadas do centro do Robô 2 ocorrem nas detecções do MS (Figura 5.12b), chegando a ter 2 valores acima de 50 pixels. Isso ocorre nas duas imagens onde o sistema confundiu a bola com o Robô 2, vide Figura 5.6c e Figura 5.6d. Os erros para o centro do Robô 2 nas detecções da *Ballnet Pose* são melhor distribuídos do que para os centros dos Robôs 0 e 1, mas ainda se concentram em sua maioria — 43 detecções — em um intervalo com limite superior menor que 1 pixel.



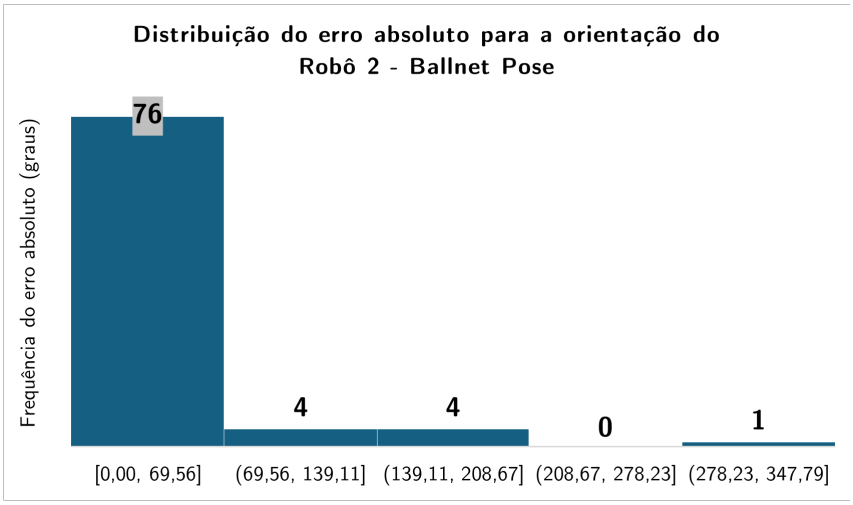

(a) *Ballnet Pose*



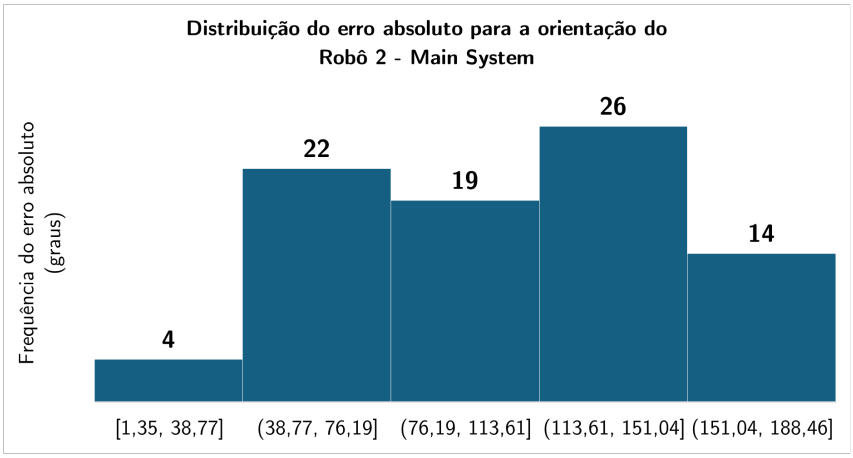

(b) *Main System* (MS)

Figura 5.12: Frequência do erro médio para o centro do Robô 2.

Pode-se observar pela Figura 5.13a que a *Ballnet Pose* comete erros maiores para as orientações do Robô 2 do que para as orientações dos Robôs 0 e 1, chegando a ter 1 detecção no intervalo (278,23; 347,79] graus e 8 detecções entre 69,56 e 139,11 graus. Comparando com a distribuição no MS — Figura 5.13b —, vemos que o valor máximo dos erros nas orientações calculadas pela *Ballnet Pose* é maior, e a maioria — 26 detecções — se encontra no intervalo (113,61; 151,04].




(a) *Ballnet Pose*

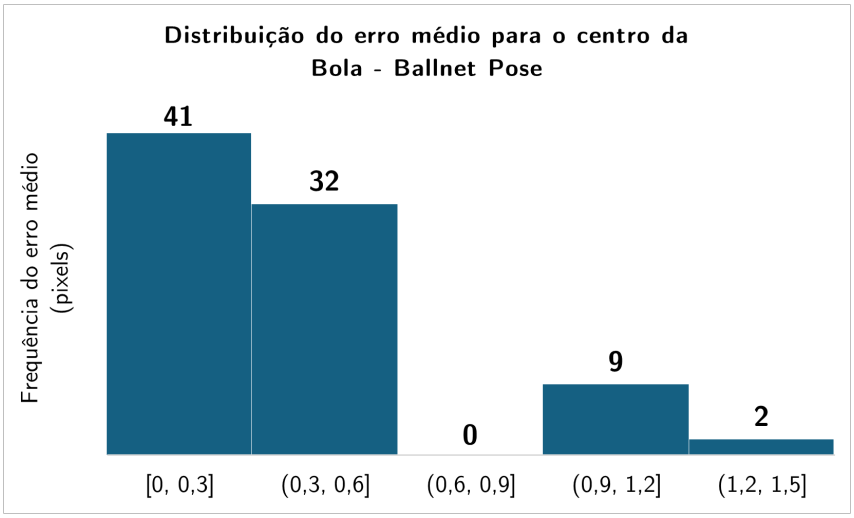



(b) *Main System* (MS)

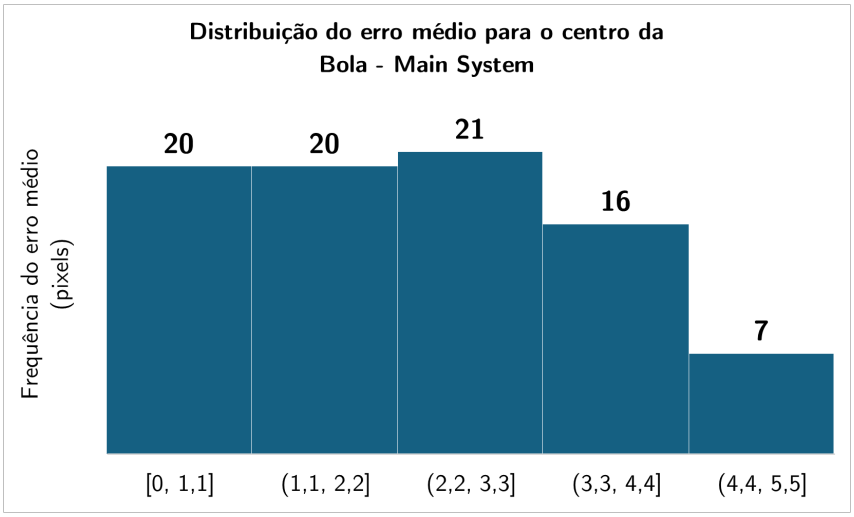
Figura 5.13: Frequência do erro absoluto para a orientação do Robô 2.

Em relação às coordenadas do centro da Bola, *Ballnet Pose* apresenta menores erros do que o MS (Figura 5.14). É interessante observar que, dentre os objetos de interesse, o MS apresentou menores valores de erro para a Bola Figura 5.14b.

As comparações realizadas avaliam as detecções com base em métricas relacionadas às dimensões das imagens. No entanto, esses resultados não são suficientes para avaliar a detecção de objetos em escala real no campo, uma vez que outros fatores devem ser considerados, como a distorção dessas dimensões devido às características de hardware das câmeras utilizadas durante as partidas.




(a) *Ballnet Pose*




(b) *Main System* (MS)

Figura 5.14: Frequência do erro médio para o centro da Bola.

# Capítulo 6

# Conclusão

A popularidade de modelos de inteligência artificial também traz o questionamento se essa abordagem é mais eficiente do que métodos tradicionais na detecção de objetos. Cada tarefa e cenário específicos trazem desafios diferentes e, portanto, a resposta está aberta a depender da aplicação. Detectar objetos em um campo *Very Small Size Soccer* (VSSS) pode ser menos complicado que detectar objetos em um campo de futebol convencional, mas ainda assim possui obstáculos. Propor técnicas e abordagens que se inspiram no estado da arte no campo de Visão Computacional pode ser um passo para alcançar melhores detecções e proporcionar partidas focadas em avaliar as estratégias para além do sistema de visão.

A partir dos resultados obtidos, é possível concluir que o modelo YOLOv8x-pose foi o que trouxe menor perda dentre as variantes de YOLOv8 disponíveis, quando treinado no banco de imagens formado por quadros de vídeos de partidas da UnBall gravadas na IRONCup 2020. Esse modelo detecta os robôs da UnBall e a bola de uma partida VSSS melhor que o sistema *Main System* (MS), atualmente usado pela equipe UnBall. Podemos concluir que o modelo proposto localiza o centro dos objetos com menor erro do que o algoritmo implementado no MS e, também, calcula a orientação dos robôs com maior precisão.

As comparações feitas avaliam as detecções quanto às métricas relativas às dimensões das imagens. Esses resultados não são suficientes para avaliar a detecção dos objetos no campo em escala real, visto que existem outros fatores a serem considerados como a distorção dessas dimensões devido às características de *hardware* das câmeras usadas nas partidas.

Existem muitas outras percepções ocultas à primeira vista a serem exploradas a partir dos dados obtidos. Novas conclusões podem surgir a partir de pontos de vista diferentes do autor, a geração de novas hipóteses e suas investigações ficam abertas. As sugestões visam tornar o projeto *Ballnet Pose* um contribuinte do universo que envolve futebol de

robôs, especificamente da categoria VSSS.

## 6.1 Trabalhos Futuros

O trabalho realizado nessa monografia soluciona o problema de detectar os objetos de interesse numa partida VSSS, porém a avaliação em cenário real fica aberta como sugestão em trabalhos futuros. Até a entrega da monografia, o módulo *Ballnet Pose* apenas lê arquivos (imagens ou vídeos) e gera um novo arquivo de saída. Para a sua utilização em cenário real é necessário adicionar a detecção em tempo real a partir de um dispositivo — *webcam*.

A ampliação do banco de imagens é uma das sugestões de continuidade do projeto. Adicionar amostras de diversas partidas torna possível um maior grau de generalização do modelo, permitindo que os robôs da UnBall e bola sejam identificados em uma variedade maior de cenários. Para além das camisas UnBall, aumentar a generalização no sentido de identificar robôs de outras equipes é um objetivo interessante a se propor também.

# Referências


[1] S.J. Russell, P. Norvig, e E. Davis. *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall, 2010. 1, 4, 7

[2] R.C. Gonzalez e R.E. Woods. *Digital Image Processing, Global Edition*. Pearson Education, 2018. 1

[3] Maja J Matari´c. *The robotics primer*. Library of Congress Cataloging-in-Publication Information, 2007. 1

[4] Joseph Redmon, Santosh Divvala, Ross Girshick, e Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016. 2, 5, 8, 9

[5] RoboCup Federation. A brief history of robocup. `https://www.robocup.org/a_brief_history_of_robocup`. [Acessado em 06-10-2024]. 2

[6] Thiago Filipe de Medeiros, Marcos R. O. de A. Máximo, e Takashi Yoneyama. Deep reinforcement learning applied to ieee very small size soccer strategy. In *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)*, pages 1–6, 2020. 2

[7] Thayna Pires Baldão, Marcos R. O. A. Maximo, e Cecilia de Azevedo Castro Cesar. Decision-making for 5x5 very small size soccer teams. In *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)*, pages 1–6, 2020. 2

[8] Felipe B. Martins, Mateus G. Machado, Hansenclever F. Bassani, Pedro H. M. Braga, e Edna S. Barros. rsoccer: A framework for studying reinforcement learning in small and very small size robot soccer. In Rachid Alami, Joydeep Biswas, Maya Cakmak, e Oliver Obst, editors, *RoboCup 2021: Robot World Cup XXIV*, pages 165–176, Cham, 2022. Springer International Publishing. 2

[9] Thayna Pires Baldão, Marcos R. O. A. Maximo, e Takashi Yoneyama. Reinforcement learning applied to very small size soccer decision-making, trajectory planning and control in penalty kicks. In *2024 Brazilian Symposium on Robotics (SBR), and 2024 Workshop on Robotics in Education (WRE)*, pages 115–120, 2024. 2

[10] UnBall. Unball - futebol de robôs. Disponível em `https://unball.github.io/`. [Acessado em 20/02/2025]. 2

[11] Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, e Jieping Ye. Object detection in 20 years: A survey. *Proceedings of the IEEE*, 111(3):257–276, 2023. [Acessado em 14-12-2024]. 4, 8

[12] D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, 1999. 4

[13] Belongie e Malik. Matching with shape contexts. In *2000 Proceedings Workshop on Content-based Access of Image and Video Libraries*, pages 20–26, 2000. 4

[14] Yoav Freund e Robert E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In Paul Vitányi, editor, *Computational Learning Theory*, pages 23–37, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg. 4

[15] Warren S. McCulloch e Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5:115–133, 1943. 4

[16] Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, e Xindong Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 30(11):3212–3232, 2019. 5

[17] David E. Rumelhart, Geoffrey E. Hinton, e Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. 5

[18] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, e Alexander C. Berg. *SSD: Single Shot MultiBox Detector*, page 21–37. Springer International Publishing, 2016. 5

[19] Ross Girshick, Jeff Donahue, Trevor Darrell, e Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014. 5, 8

[20] Ian Goodfellow, Yoshua Bengio, e Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. 5

[21] S.J.D. Prince. *Understanding Deep Learning*. MIT Press, 2023. 5, 6, 7

[22] IBM. O que são redes neurais convolucionais? Disponível em: `https://www.ibm.com/br-pt/topics/convolutional-neural-networks`. [Acessado em 09-02-2025]. 6

[23] Data Science Academy. Deep learning book. Disponível em: `https://www.deeplearningbook.com.br/`. [Acessado em 09-02-2025]. 6

[24] Ultralytics. Ultralytics yolo docs. Disponível em: `https://docs.ultralytics.com/`. [Acessado em 9-02-2025]. 7, 9, 24

[25] Juan Terven, Diana-Margarita Córdova-Esparza, e Julio-Alejandro Romero-González. A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. *Machine Learning and Knowledge Extraction*, 5(4):1680–1716, 2023. 9, 10, 11, 19

[26] Glenn Jocher, Ayush Chaurasia, e Jing Qiu. Ultralytics yolov8. `https://github.com/ultralytics/ultralytics`, 2023. [Acessado em 12-02-2025]. 9, 11, 12, 21

[27] Rui-Yang Ju e Weiming Cai. Fracture detection in pediatric wrist trauma x-ray images using yolov8 algorithm. *Scientific Reports*, 13(1):20077, 11 2023. 11

[28] Jacob Solawetz Francesco. What is yolov8? a complete guide, 2024. Disponível em: `https://blog.roboflow.com/what-is-yolov8/`. [Acessado em 06-11-2024]. 10

[29] Gaudenz Boesch. Yolov8: The ultimate guide, Dezembro 2024. Disponível em: `https://viso.ai/deep-learning/yolov8-guide/#elementor-toc__heading-anchor-9`. [Acessado em 12-02-2025]. 10

[30] LARC. Regras ieee very small size soccer (vsss) - série a, 2023. Disponível em: `https://lars-larc2024.ucsp.edu.pe/rules/regrasVSS23.pdf`. [Acessado em 06-27-2024]. 13, 14, 15

[31] Roboflow. Roboflow annotate: Label images faster than ever. Disponível em `https://roboflow.com/annotate`. [Acessado em 20/02/2025]. 19, 20

[32] Ayssa Giovanna de Oliveira Marques. Ballnet dataset, 2025. Disponível em: `https://universe.roboflow.com/ayssag/ballnet_dataset`. [Acessado em 20-02-2025]. 20

[33] Ayssa Giovanna de Oliveira Marques. Repositório ballnetpose. Disponível em `https://huggingface.co/ayssag/BallnetPose`. [Acessado em 20/02/2025]. 24