UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
ENG. DE CONTROLE E AUTOMAÇÃO

ARTUR MARTINI DA ROSA - 00243716

SIMULAÇÃO COMPUTACIONAL DE UM DISPOSITIVO ROBÓTICO COM SISTEMA DE VISÃO BASEADO EM DEEP LEARNING

Porto Alegre
2021

ARTUR MARTINI DA ROSA - 00243716

SIMULAÇÃO COMPUTACIONAL DE UM DISPOSITIVO ROBÓTICO COM SISTEMA DE VISÃO BASEADO EM DEEP LEARNING

Trabalho de Conclusão de Curso (TCC-CCA) apresentado à COMGRAD-CCA da Universidade Federal do Rio Grande do Sul como parte dos requisitos para o obtenção do título *Bacharel em Eng. de Controle e Automação.*

ORIENTADOR:
Prof. Dr. Heraldo José de Amorim

CO-ORIENTADOR:
Dr. Vitor Camargo Nardelli

Porto Alegre
2021

ARTUR MARTINI DA ROSA - 00243716

SIMULAÇÃO COMPUTACIONAL DE UM DISPOSITIVO ROBÓTICO COM SISTEMA DE VISÃO BASEADO EM DEEP LEARNING

Trabalho de Conclusão de Curso (TCC-CCA) apresentado à COMGRAD-CCA da Universidade Federal do Rio Grande do Sul como parte dos requisitos para o obtenção do título *Bacharel em Eng. de Controle e Automação.*

Orientador:_____
Prof. Dr. Heraldo José de Amorim, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul - Porto Alegre, Brasil

Banca Examinadora:

Prof. Dr. Heraldo José de Amorim, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul - Porto Alegre, Brasil

Prof. Dr. Marcelo Götz, UFRGS
Doutor pela Universidade de Paderborn - Parderborn, Alemanha

Prof. Dr. Rafael Antônio Comparsi Laranja, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul - Porto Alegre, Brasil

_____
Mário Roland Sobczyk Sobrinho
Coordenador de Curso
Eng. de Controle e Automação

Porto Alegre, novembro de 2021.

RESUMO

A industria moderna busca maior flexibilidade e eficiência através da aplicação de novas tecnologias em seus processos. A utilização de dispositivos robóticos em ambiente industrial é comum para automatizar partes de linhas de produção e atingir maior desempenho. Todavia, o custo de implementação de robôs é alto e sua integração com a planta de manufatura não trivial. Dentro desse contexto, o presente trabalho tem como objetivo explorar o uso de simulação computacional para construção de um *digital twin* de um sistema robótico. Um sistema físico composto por um robô KUKA iiwa e uma câmera RGB é utilizado como referência para o desenvolvimento do sistema digital deste trabalho. A simulação é caracterizada em uma tarefa de inspeção de peças automatizada, realizada por um sistema de visão computacional baseado em aprendizado de máquina. A construção do *digital twin* é feita em simulação com ferramentas de software e métodos de aprendizado de máquina. Os principais testes do projeto foram realizados em ambiente simulado e apresentam resultados promissores na tarefa de detecção de peças. Também são analisadas imagens de peças reais como uma primeira avaliação para integração com o sistema físico. Na detecção em imagens reais foram atingidos resultados comparáveis ao estado da arte no problema de detecção de objetos.

Palavras-chave: Visão de Máquina, Redes Neurais Convolucionais, Simulação computacional, Robótica, Dados sintéticos.

ABSTRACT

Modern industry aims for greater flexibility and efficiency through implementation of new technologies in their processes. The use of robotic devices is commum in industrial manufactoring as a mean to automate and achieve greater performance. However, the implementation of robots has a high cost and is non-trival. Thus, this work aims to explore the use of computer simulation to build a digital twin of a robotic system. A physical system is used as reference for development of the digital system. The physical system consists of a KUKA iiwa robot and an RGB camera. The simulation is based in an automated inspection routine of parts performed by a computer vision system with machine learning techniques. The development of the digital twin is accomplished with software tools and machine learning technics. The main tests of this project were carried out in a simulated environment and present promising results in the parts detection task. Real images are also used for validation as a first step for integration with the physical system. The detection in real images obteined results comparable to the state of art in object detection problem.

Keywords: Machine Vision, Convolutional Neural Networks, Computer Simulation, Robotics, Synthetic Data.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

3D	*Threedimensional*	(Tridimensional)
CAD	*Computer Aided Design*	(Desenho auxiliado por computador)
CPS	*Cyber Physical Systems*	(Sistema Ciberfísico)
DH	*Denavit-Hartenberg*	
DL	*Deep Learning*	(Aprendizado Profundo)
DR	*Domain Randomization*	(Aleatorização de domínio)
DT	*Digital Twin*	(Gêmeo digital)
ML	*Machine Learning*	(Aprendizado de Máquina)
ROS	*Robot Operating System*	(Sistema Operacional de Robô)
URDF	*Unified Robot Description Format*	(Formato unificado de descrição de robôs)
USD	*Universal Scene Description*	(Descrição Universal de Cena)

1 INTRODUÇÃO

Linhas de montagem modernas exigem sistemas de produção flexíveis capazes de se adaptar rapidamente a mudanças de demanda e a personalização de produtos (KOUSI et al., 2019). Essas necessidades implicam na utilização de novas estratégias de produção, automatização de processos e reconfigurações de plantas industriais. Alguns sistemas que possuem alto potencial de automação e flexibilidade são os robóticos, que podem ser reprogramados e equipados com diferentes ferramentas e sensores para realizar tarefas diversas.

Inserido dentro do contexto industrial, um dispositivo robótico deve ser capaz de interagir com objetos do seu ambiente de trabalho. Uma forma de atingir essa interação é através da programação manual de coordenadas fixas dos objetos conhecidos. Entretanto, os objetos alvo do robô não são necessariamente fixos. Com isso, surge a necessidade da integração de sensores ao manipulador robótico, permitindo ao sistema obter dinamicamente as coordenadas dos objetos de trabalho. O sensoriamento pode ser feito com câmeras e constituir um sistema de visão de máquina como exemplificado no trabalho de Akinola et al. (2021), que processa imagens para detectar objetos no ambiente de trabalho do robô.

Métodos clássicos de detecção de objetos em imagens são baseados na construção manual de filtros para extrair características dos objetos na imagem, como bordas, curvas e cor. Tais métodos são limitados devido a variação dos objetos nas imagens, em função da perspectiva e iluminação (KRÜGER et al., 2017). Para contornar isso, técnicas de aprendizado de máquina (ML do inglês, *Machine Learning*) são aplicadas nas tarefas de detecção de objetos. LeCun et al. (1998) apresenta uma classe de algoritmo de ML, as redes neurais convolucionais (CNN do inglês, *Convolutional Neural Networks*), capaz de atingir elevados índices de precisão na tarefa de detecção de objetos em imagens, sem a construção manual de filtros. Trabalhos recentes de OpenAI (2019) e Akinola et al. (2021) utilizam um sistema de visão baseado em ML para guiar as ações de manipuladores robóticos e obtém sucesso em tarefas complexas, o que indica as técnicas de ML como uma solução robusta para processamento de imagens. Entretanto, para desenvolver aplicações de visão de máquina baseada em ML é necessário um alto volume de dados, nesse caso imagens, para treinamento dos algoritmos. Além disso, a utilização de robôs para testes em ambiente industrial é onerosa, visto que esses dispositivos possuem um custo elevado. Com isso, o tempo de operação dos robôs em linhas de produção deve ser maximizado, o que dificulta desenvolvimentos paralelos com esses dispositivos.

Diante desse contexto, o presente trabalho propõe a aplicação de simulação computacional para o desenvolvimento de um sistema de visão de máquina baseado em ML. A simulação desenvolvida utiliza ferramentas do estado da arte para criar dados de treinamento dos algoritmos de ML e, também, para simulação da cinemática do robô e sua integração com o sistema de visão. O objetivo desse sistema de visão é guiar um robô antropomorfo em uma tarefa hipotética de inspeção de peças mecânicas. Para tanto, é utilizado como referência um robô real modelo KUKA iiwa com uma câmera fixada ao seu efetuador. Em resumo, o objetivo geral do trabalho é a construção de um gêmeo digital (*Digital Twin*) desse sistema físico.

A partir da simulação desenvolvida foram realizados testes do sistema de visão com diferentes peças e condições de iluminação variadas, como meio de avaliar seu desempenho. Dessa forma, o presente trabalho contribui como um estudo de ferramentas de simulação, no contexto de visão de máquina baseada em ML para sensoriamento de um robô industrial.

2 REVISÃO BIBLIOGRÁFICA E ESTADO DA ARTE

O presente capítulo tem como objetivo apresentar as principais tecnologias e conceitos utilizados na fundamentação e desenvolvimento deste trabalho.

2.1 Digital Twin

Um *Digital Twin* (DT) é uma cópia digital de um elemento físico simulado em computador. Dentro do ambiente de uma manufatura a cópia digital pode assumir diferentes formas e escalas, por exemplo: um único processo, um módulo ou até uma planta de produção inteira.

O conceito de DT, ou equivalente digital, de um produto físico foi introduzido em 2003 no contexto de gerenciamento de ciclo de vida de produtos (GRIEVES, 2015). Posteriormente esse conceito recebeu aprofundamento técnico dentro da indústria aeroespacial dos Estados Unidos da América, e foi exposto ao público geral em um trabalho sobre rotas tecnológicas de Glaessgen et al. (2012). A inspiração do conceito originou-se de uma missão do programa Apollo da NASA, onde foi construída uma cópia física, denominada *twin*, de um veículo espacial. A finalidade dessa cópia era servir como objeto de testes enquanto o veículo original realizava a missão (BOSCHERT et al., 2016).

A estrutura de um DT é composta por três macro componentes: um objeto físico, sua cópia digital e a conexão entre ambos. Um detalhamento dessa estrutura foi apresentado por Shafto et al. (2010) como uma simulação de um sistema que utiliza modelos físicos, sinais de sensores e histórico de operações para imitar o sistema real. Nesse aspecto, um DT se assemelha a um Sistema Ciberfísico (CPS do inglês, *Cyber-physical System*) que também tem como objetivo integrar ambientes físico e virtual. Entretanto, o trabalho de (TAO et al., 2019) expõe uma distinção quanto ao foco dos conceitos. Os CPS focam em fundamentações teóricas e tecnologias, enquanto os DT são uma aplicação prática dos CPS e uma forma pragmática de realizar a integração entre físico e virtual.

As aplicações de DT são geralmente dentro do contexto industrial nas tarefas de integração de sistemas, diagnósticos de operações e predição da condição de operação de ativos. Trabalhos recentes desenvolvidos pela empresa BMW em parceria com a NVIDIA apresentam um DT completo de uma linha de produção automotiva (CAULIFIELD, 2021). Através dessa cópia a montadora de carros foi capaz de testar uma nova frota de robôs de logística, para transporte de materiais na fábrica. Um segundo exemplo de aplicação é o trabalho da empresa Kinetic Vision (VISION, 2020), onde a automação de um

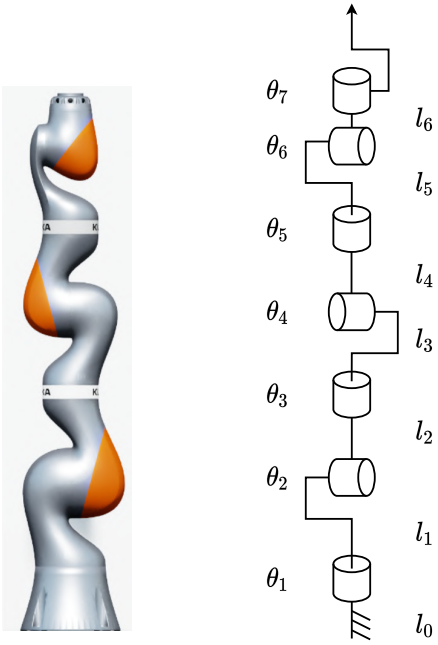
processo de inspeção por tomografia computadorizada foi construída e testada via DT .

2.2 Robótica

A robótica é uma área multidisciplinar, cujo objetivo é construir mecanismos capazes de automatizar tarefas e processos. A aplicação de robôs é comum em ambientes industriais nas tarefas de transporte, inspeção e manipulação de peças e ferramentas.

Devido à natureza deste trabalho os conceitos de cinemática de robôs fixos e seu sensoriamento via câmera são brevemente examinados. Os robôs fixos são classificados de acordo com suas características construtivas. Observam-se dois elementos mecânicos principais na sua geometria, os elos e as juntas e como eles estão organizados. O robô analisado neste trabalho é do tipo fixo articulado com sete juntas rotacionais em série, uma ilustração do robô de estudo, marca KUKA modelo LBR iiwa (KUKA, 2021), com indicação das suas juntas θ_i e elos l_i é apresentada na Figura 1.

Figura 1: *Robô articulado KUKA LBR iiwa e diagrama das suas sete juntas rotacionais θ_i e elos l_i.*



Fonte: Elaborada pelo autor.

2.2.1 Cinemática

A cinemática estuda os movimentos do robô sem considerar as forças geradoras de tal movimento. Dentro da cinemática estuda-se a posição, velocidade, aceleração e demais derivadas de ordem superior (CRAIG, 1986). Isso é feito analisando as propriedades geométricas do robô e construindo suas equações cinemáticas.

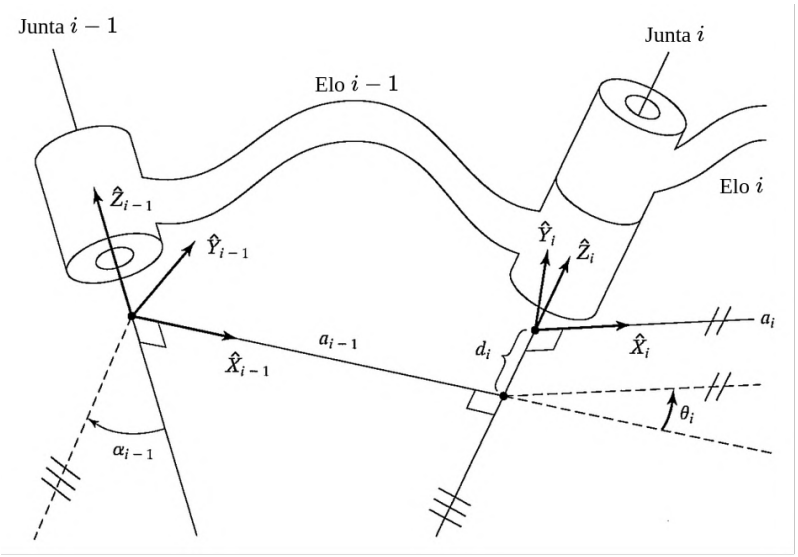
São diferenciados dois tipos de cinemática nesse contexto, a cinemática direta e a inversa. A cinemática direta é utilizada para obter a posição $\vec{P_e}$ e orientação $\vec{\Omega_e}$ do efetuador final do robô, para um estado de juntas $\vec{\theta}$ conhecido. Já a inversa utiliza uma posição e

orientação conhecida do efetuador final para calcular um conjunto de estados de juntas que resultam na posição do efetuador.

Uma forma de descrever a cinemática de robôs é através de quatro parâmetros e seus valores por elo do sistema, essa convenção é usualmente chamada de notação de Denavit-Hartenberg (DH) (CRAIG, 1986) e estabelece um procedimento para atribuição dos eixos de coordenadas cartesianas $(\hat{X}, \hat{Y}, \hat{Z})$ para cada junta do robô. Vale destacar que existem diversas notações para fixação das coordenadas, mas a de DH é a mais popular. Os parâmetros de DH e suas definições são listados abaixo e exemplificados na Figura 2.

- d_i: distância de $\hat{X}_{i-1}$ para $\hat{X}_i$ medida sobre o eixo $\hat{Z}_i$.
- α_i: ângulo de torção entre $\hat{Z}_i$ e $\hat{Z}_{i+1}$ medido pelo eixo $\hat{X}_i$.
- a_i: distância de $\hat{Z}_i$ para $\hat{Z}_{i+1}$ medida sobre o eixo $\hat{X}_i$.
- θ_i: ângulo de articulação entre $\hat{X}_{i-1}$ e $\hat{X}_i$ medido pelo eixo $\hat{Z}_i$.

Figura 2: *Sistemas de coordenadas de duas juntas genéricas $i-1$ e i.*



Fonte: Adaptada de (CRAIG, 1986).

A partir dos parâmetros de DH montam-se as matrizes de transformação homogênea T, que descrevem a transformação espacial entre os sistemas de coordenadas das juntas. Para um caso genérico a matriz T_i^{i-1} é definida conforme a Equação 2.1.

$$T_i^{i-1} = \begin{bmatrix} C\theta_i & -S\theta_i C\alpha_i & S\theta_i S\alpha_i & a_i C\theta_i \\ S\theta_i & C\theta_i C\alpha_i & -C\theta_i S\alpha_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

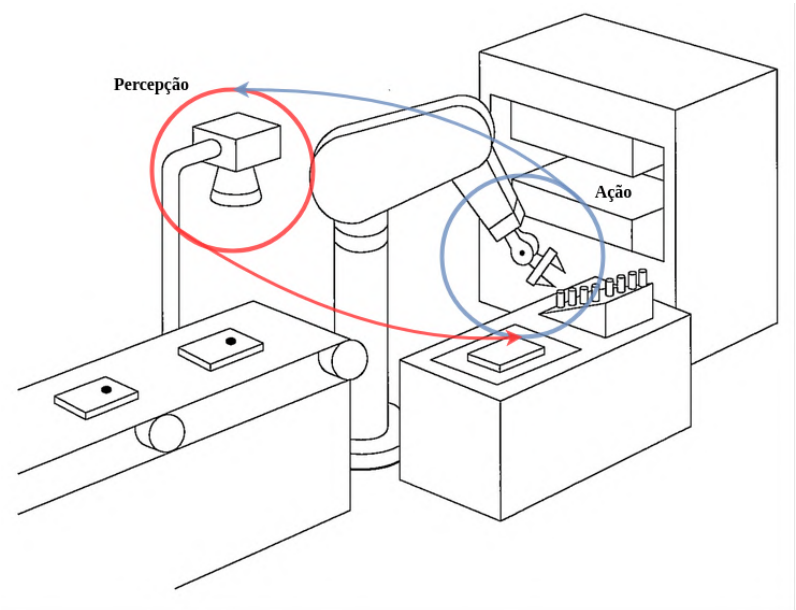
Para cada conjunto de parâmetros de DH é construída uma matriz de transformação, o produtos dessas matrizes permite a transformação de coordenadas entre duas juntas do robô.

A cinemática inversa envolve, em geral, a solução de um problema não linear, que pode possuir múltiplas soluções. Além disso, a existência de solução está condicionada à geometria do manipulador robótico e suas limitações mecânicas, que definem um espaço de trabalho onde o robô é capaz de se mover. Por exemplo, se a posição e orientação desejada para o efetuador do robô está fora da área de trabalho não existe solução para a cinemática inversa. As soluções para esse problema podem utilizar basicamente dois métodos: analítico através de identidades geométricas e inversão das equações de cinemática direta ou numéricos iterativos.

2.2.2 Sistemas de percepção e visão de máquina

Para um dispositivo robótico ser capaz de responder à mudanças no seu ambiente físico é necessário que ele possua um sistema de percepção. Tal sistema é um conjunto de sensoriamento e processamento de dados, cujo objetivo é obter informações para orientar as ações do robô. Em geral, as informações obtidas são descrições do ambiente na forma de coordenadas espaciais. Um exemplo descrito por Craig (1986) é a integração entre robô e um sistema de percepção baseado em visão computacional (Figura 3). Neste caso as coordenadas de um objeto de interesse, transportado por uma esteira, são calculadas via processamento de imagem e mapeadas no espaço de juntas do robô utilizando sua cinemática inversa.

Figura 3: *Exemplo de integração entre um robô articulado e um sistema de visão, que identifica objetos em uma esteira.*



Fonte: Adaptada de (CRAIG, 1986).

Além dos dispositivos câmera e robô existe o componente lógico, geralmente na forma de algoritmos de software, para realizar o processamento das imagens e obter informações que descrevam o ambiente ao robô. Tais algoritmos pertencem à área de visão de máquina. Essencialmente, essa área é composta por um conjunto de tecnologias e métodos voltados à automatização de processos a partir da análise de imagens (THEODORIDIS et al., 2006).

Quanto aos métodos utilizados na construção dos algoritmos de visão destacam-se neste trabalho os métodos baseados em aprendizado de máquina.

Uma necessidade que surge da integração entre câmera e robô é a calibração entre eles, já que ambos possuem referenciais distintos. Por exemplo, o objeto identificado na imagem é descrito em um plano da câmera por coordenadas de pixels (u, v), enquanto a posição do efetuador final do robô é descrita por coordenadas cartesianas (x, y, z) e rotações (α, β, γ) em torno desses eixos. Esse conjunto de variáveis do efetuador final é denominado pose e geralmente utiliza a base de fixação do robô como referência. Nesse contexto, a estimativa da transformação espacial $T^{robô}_{câmera}$ entre os sistemas de coordenada de cada elemento é necessária para possibilitar ações coerentes do robô sobre os objetos nas imagens.

De forma genérica esse processo é denominado calibração extrínseca. No caso de um robô com uma câmera acoplada ao efetuador esse processo é conhecido como *Hand-Eye Calibration* (HORAUD et al., 1995). A calibração *Hand-Eye* estima uma transformação espacial T^e_c entre o sistema de referência da câmera e do efetuador do robô, conforme a Figura 4. Uma forma de calcular essa transformação é através dos seguintes passos descrito em OpenCV (2021):

1. um padrão de calibração estático ("Alvo", Figura 4) é utilizado para estimar a transformação T^c_a entre o Alvo e a Câmera;
2. o efetuador do robô é posicionado em diferentes poses;
3. para cada pose uma transformação T^b_e entre o Efetuador e a Base do robô é salva;
4. para cada pose uma transformação T^c_a entre o Alvo e a Câmera é salva.

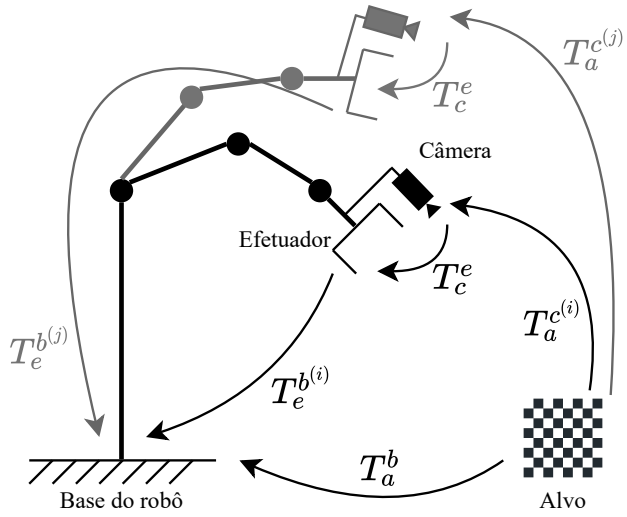
Para duas ou mais poses surge a relação da Equação 2.2, que resulta num problema do tipo $AX = XB$. Resolvendo para X obtém-se a transformação T^e_c.

$$T_e^{b^{(1)}} T_c^e T_a^{c^{(1)}} = T_e^{b^{(2)}} T_c^e T_a^{c^{(2)}} \tag{2.2}$$

$$(T_e^{b^{(2)}})^{-1} T_e^{b^{(1)}} T_c^e = T_c^e T_a^{c^{(2)}} (T_a^{c^{(1)}})^{-1}$$

$$A_i X = X B_i$$

Figura 4: *Diagrama de transformações espaciais para tarefa de calibração, com duas poses de robô genérico.*



Fonte: Adaptada de (OPENCV, 2021).

2.3 Aprendizado de Máquina

Aprendizado de máquina (ML, tradução do inglês, *Machine Learning*) é um subconjunto da área de Inteligência Artificial que estuda e aplica algoritmos para resolver problemas práticos através da construção de um *dataset* (do inglês, conjunto de dados) e treinamento de um modelo estatístico baseado nesse *dataset* (BURKOV, 2020). Em geral, os problemas resolvidos por ML exigem conhecimentos do ambiente físico que são triviais para um especialista humano, mas de difícil programação em um computador. Alguns dos problemas mais comuns resolvidos com ML, apresentados no trabalho de Goodfellow et al. (2016), são:

- Classificação: Dentro de um conjunto k de categorias o algoritmo determina a qual pertence um novo dado de entrada. Um exemplo de aplicação são modelos para reconhecimento de objetos, onde a entrada é uma imagem e a saída é um número inteiro entre 1 e k que representa a lista de categorias treinadas.

- Regressão: O algoritmo infere um valor numérico a partir de um dado de entrada. Diferentemente da classificação a saída é contínua. Uma aplicação de regressão é no setor financeiro, em específico na análise de crédito dos clientes. Um modelo de regressão pode ser treinado sobre a base de dados de um banco para estimar quanto crédito deve ser oferecido aos clientes.

- Detecção de anomalias: Nesse problema o algoritmo deve analisar uma sequência de eventos e indicar padrões anômalos aos previamente observados. Um contexto desse problema são máquinas-ferramenta, onde é analisada a vibração dos componentes mecânicos com objetivo de detectar anomalias que possam levar o equipamento à falha.

Além da diferenciação quanto aos tipos de problemas em que algoritmos de ML são empregados existe uma macro divisão quanto à forma de aprendizado desses algoritmos. Russell et al. (2020) descreve três grandes grupos de aprendizado de máquina. De forma breve pode-se enunciá-los como:

- Aprendizado não-supervisionado: Um modelo aprende padrões nos dados de entrada sem realimentação explícita.
- Aprendizado supervisionado: Um modelo aprende padrões dos dados de entrada recebendo realimentação via exemplos de pares de entrada-saída desejados.
- Aprendizado por reforço: Um modelo aprende por reforços negativos e positivos.

Este trabalho utiliza algoritmos de aprendizado supervisionado como solução ao problema de classificação de objetos em imagens. Portanto, o foco da revisão é limitado a esse escopo.

O problema de classificação de objetos em imagens teve um progresso expressivo na última década. Uma forma de avaliar tal progresso é através de desafios impostos pela comunidade científica, a qual disponibiliza *datasets* de imagens para comparação de desempenho de algoritmos. O desafio *ImageNet Large Scale Visual Recognition Challenge* proposto por Deng et al. (2009) contava com 1.2 milhões de imagens e 1000 classes diferentes e foi marcado pelo desempenho de algoritmos baseados em redes neurais artificiais. Em especial, algoritmos baseados em redes neurais e aprendizado profundo (DL do inglês, *Deep Learning*) superaram o estado da arte ano após ano. Como exemplo destacam-se dois trabalhos e suas exatidões de classificação: algoritmo AlexNet de Krizhevsky et al. (2012) 63,3% e algoritmo YOLO de Redmon, Divvala et al. (2016) 88,0%.

Esse progresso no problema de classificação e no desenvolvimento de algoritmos de aprendizado profundo foi impulsionado pela maior disponibilidade de grandes volumes de dados e, também, por avanços no hardware para treinamento dos modelos.

2.3.1 Redes Neurais e Aprendizado Profundo

Tradicionalmente os algoritmos classificados como ML dependem fortemente de *feature engineering* (do inglês, engenharia de características) (STEVENS et al., 2020). As *features* são representações dos dados construídas para destacar informações desejadas. Entretanto, a construção dessas *features* exige especialistas humanos com conhecimentos específicos do problema alvo. Os métodos de aprendizado profundo surgem nesse contexto com objetivo de aprender as representações (*features*) dos dados sem interferência direta.

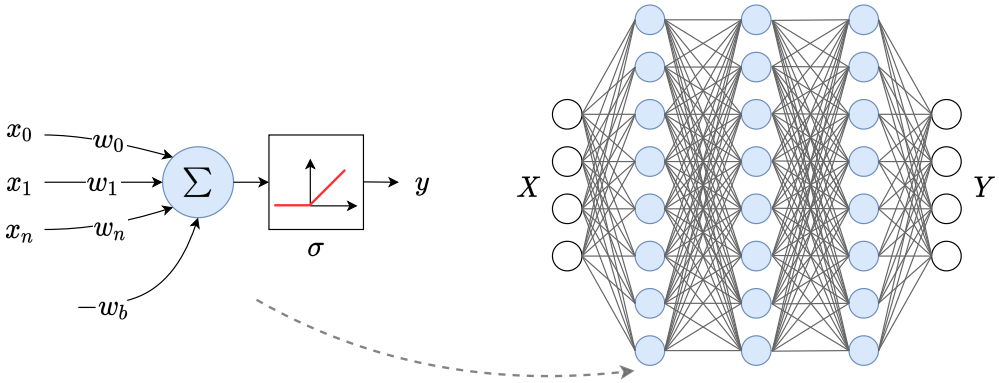
De forma geral, os algoritmos de aprendizado profundo são estruturas de redes neurais artificiais. Tal estrutura é referenciada na literatura como "arquitetura de rede" e é formada por multicamadas de módulos simples. Quase todos esses módulos possuem parâmetros que podem ser alterados durante a etapa de treinamento (LECUN et al., 2015).

Os módulos simples das redes possuem uma unidade básica denominada *Perceptron*. Essa unidade é conhecida como neurônio artificial e sua modelagem - realizada por Rosenblatt (1958) - foi inspirada em observações de neurônios biológicos. Resumidamente, três características são modeladas de forma simplificada no *Perceptron*:

1. Comportamento binário, os neurônios ativam ou não;
2. Existem n sinais de entradas x que são transmitidos até um núcleo com diferentes ganhos w, também chamados de *weights* (do inglês, pesos);
3. A soma dos sinais de entrada deve superar um valor limite w_b para ativar a saída y via uma função de ativação σ.

A modelagem do *Perceptron* toma a forma apresentada na Figura 5, onde também é ilustrada ao lado uma rede neural profunda genérica composta por camadas constituídas de neurônios artificiais.

Figura 5: *Modelo do neurônio artificial Perceptron (esquerda) e uma rede neural profunda com três camadas totalmente conectadas (direita).*



Fonte: Elaborada pelo autor.

Matematicamente, o neurônio artificial é definido pela Equação 2.3, onde σ é uma função de ativação, usualmente do tipo retificador.

$$y = \sigma\left(\sum_{i=0}^{n}(w_i x_i) - w_b\right) \tag{2.3}$$

O método para definir os parâmetros w das redes neurais artificiais é brevemente resumido. Pode-se interpretar essas redes como algoritmos que aproximam funções através do ajuste dos seus parâmetros w. Esse processo de ajuste é conhecido como "treinamento de rede" na literatura. No treinamento de algoritmos supervisionados existe uma função alvo $g(x)$ desconhecida que deseja-se aproximar com a função da rede neural artificial

$$y = f(w, x),$$

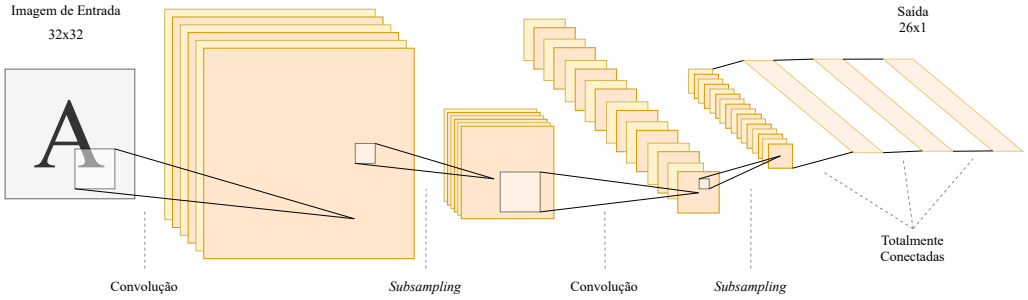
as saídas d da função alvo $g(x)$ são conhecidas e utilizadas para gerar um erro $e = ||d - y||^2$. Esse erro é então aplicado em uma função de custo $P(e)$, cujas derivadas parciais são calculadas em função de cada parâmetro w. Por final, os parâmetros são ajustados proporcionalmente a um passo r para minimizar P. Tal passo é denominado taxa de aprendizado. Esse processo é iterativo e repetido até atingir um valor aceitável para o erro e. Caso o erro desejado não seja atingido recorre-se à alterações dos chamados hiperparâmetros, como número de iterações e taxa de aprendizado r, outras opções são modificar a arquitetura da rede e/ou atualizar os dados utilizados.

2.3.2 Redes Neurais Convolucionais

As redes neurais convolucionais são algoritmos de aprendizado profundo que utilizam filtros de convolução conhecidos como *kernels*. Tais filtros são empregados para extrair características elementares de imagens, como bordas e formas. Essa estrutura foi incorporada em redes neurais artificiais por LeCun et al. (1998) em um trabalho sobre aplicação de redes convolucionais no processamento de imagens, texto e séries temporais. Os autores apresentam a aplicação de *kernels* de convolução como uma forma de contornar o elevado número de parâmetros das estruturas compostas exclusivamente de camadas totalmente conectadas (Figura 5). Por exemplo, se cada pixel de uma imagem for a entrada de um elemento da estrutura da rede neural, uma primeira camada teria $largura \times altura$ (pixels) parâmetros. Já com a aplicação de um *kernel* de dimensões fixas (5×5 pixels, 25 parâmetros por exemplo) e a operação de convolução esse número é bastante reduzido, junto com a quantidade de memória necessária no hardware de treinamento.

A partir dos filtros são construídas arquiteturas de redes como da Figura 6, onde são empregadas duas camadas de convolução para extrair características da imagem de entrada, três camadas totalmente conectadas para mapear as características extraídas em uma classe e duas camadas de amostragem denominadas *subsampling*. Essa última estrutura da CNN cria um mapa de características condensadas, em geral extraindo o valor máximo dos *kernels* e reduzindo as dimensões da imagem de entrada.

Figura 6: *Exemplo de CNN para classificação de imagens de dígitos.*



Fonte: Elaborada pelo autor com base na rede convolucional apresentada por LeCun et al. (1998).

3 MATERIAIS E MÉTODOS

Este capítulo é dedicado à apresentação da arquitetura do sistema proposto e os métodos utilizados para realizá-lo. Inicia-se com uma visão geral do problema de inspeção de peças e suas especificações. Em sequência, a arquitetura da solução é descrita e seus subsistemas são detalhados, com os principais materiais e métodos empregados para desenvolvê-los. Por fim, realiza-se a integração das partes do sistema.

3.1 Problema de inspeção

Uma aplicação típica de sistemas de visão de máquina em manufaturas é a inspeção de peças através da detecção e classificação desses objetos em imagens (THEODORIDIS et al., 2006). O caso de estudo deste trabalho é a simulação de uma rotina de inspeção visual, que é realizada por um robô fixo com uma câmera acoplada à sua flange. O sistema real usado como referência para construção do DT é apresentado na Figura 7. A inspeção é feita em um conjunto de peças mecânicas que estão dispostas sobre uma área de trabalho. O sistema de visão faz a identificação de quais peças estão sobre essa área e, ao localizar uma peça específica, o robô se aproxima e adquire uma nova imagem da peça "alvo".

Figura 7: *Sistema real de referência, robô KUKA iiwa com câmera acoplada ao efetuador e peças dispostas sobre uma área de trabalho.*

Fonte: Cortesia do Instituto SENAI de Inovação Soluções Integradas em Metalmecânica (ISI SIM).

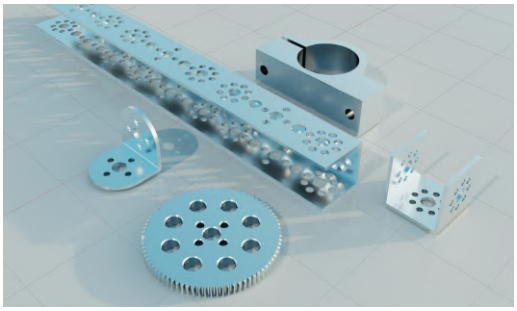
Para caracterizar a tarefa de inspeção foram selecionadas cinco peças mecânicas reais.

Esse conjunto, apresentado na Figura 8a, é formado pelas seguintes peças de alumínio: um suporte *U*, um suporte *L*, uma engrenagem de dentes retos, um mancal de motor cc e um perfil. Também foram utilizados os modelos 3D das peças, que são dispostos na Figura 8b e compõe o ambiente de simulação.

Figura 8: *Conjunto de peças mecânicas.*

(a) *Peças reais e referência de escala.*

(b) *Modelos 3D das peças.*



Fonte: Elaborada pelo autor.

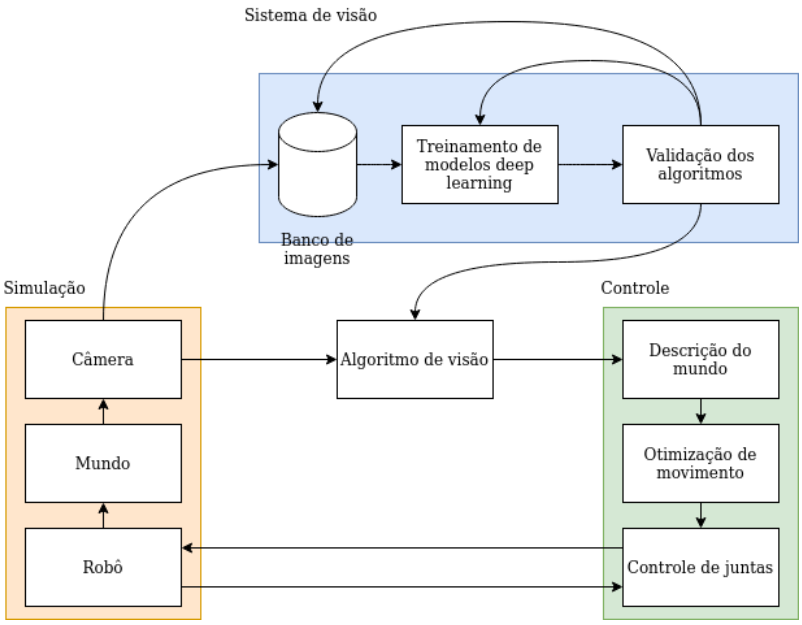
O início da rotina de inspeção é dado quando o controle do sistema recebe uma mensagem com o nome da peça alvo. Em sequência o robô se movimenta até uma posição zero, onde toda área de trabalho é visível pela câmera da flange. Uma imagem é capturada e processada pelo sistema de visão, obtendo uma informação de quais peças estão sobre a superfície de trabalho, assim como a sua posição no plano da imagem. Com essa informação o robô executa movimentos de aproximação nas peças alvo e captura novas imagens, essa operação é repetida conforme a quantidade de peças "alvo" detectadas. Por exemplo, se a peça alvo for uma engrenagem e três engrenagens forem detectadas sobre a área de trabalho, são executadas três aproximações.

3.2 Arquitetura do sistema

A arquitetura deste trabalho é dividida em subsistemas, de forma que o seu desenvolvimento possa ser realizado em módulos, a Figura 9 apresenta um diagrama com essa divisão. Todos os componentes do diagrama são aplicações de software e possuem as seguintes funções:

- Simulação: Simular a cinemática do robô, o ambiente ou "mundo" que o robô está inserido e a câmera;
- Sistema de visão: Adquirir imagens do ambiente, processá-las com algoritmo de ML e retornar as posições das peças. Paralelamente à rotina de inspeção também deve realizar o treinamento e validação dos algoritmos de ML.
- Controle: Define a sequência de operações executadas pelo sistema de visão, lê o estado de juntas do robô e publica novos estados de junta conforme resultado de posição das peças.

Figura 9: *Diagrama da simulação proposta, com fluxos de informação entre módulos.*

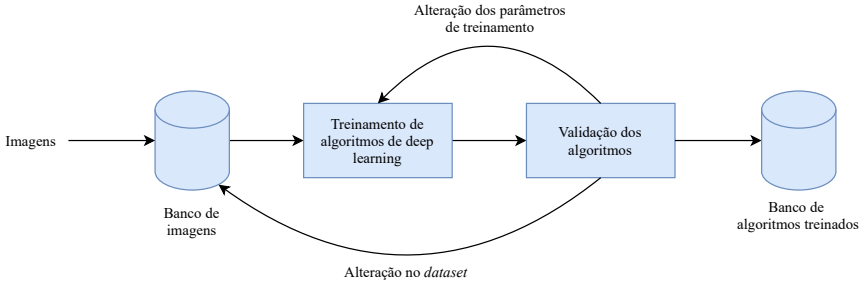


Fonte: Elaborada pelo autor.

3.3 Sistema de visão

O sistema de visão tem por finalidade processar as imagens adquiridas pela câmera para obter descrições do ambiente. Tais descrições são utilizadas como orientação das ações do robô. Para realizar o processamento das imagens foram selecionadas técnicas de ML supervisionado com estrutura de CNN, que possuem bom desempenho em tarefas de classificação de objetos, como visto na Seção 2.3. A estrutura desse sistema é apresentada pelo diagrama da Figura 10 e pode ser resumida em três etapas: aquisição e preparação de uma base de dados (nesse caso imagens), treinamento da CNN selecionada e validação da CNN com base em métricas de desempenho. O processo de treinamento é iterativo, e conforme os resultados obtidos na etapa de validação pode-se alterar os dados da entrada (por exemplo, aumentando a quantidade de amostras) ou os parâmetros de treinamento dos algoritmos. No final desse processo um arquivo contendo os parâmetros internos da CNN é salvo.

Figura 10: *Diagrama do sistema de visão.*



Fonte: Elaborada pelo autor.

3.3.1 Banco de imagens

A construção de um banco de dados, nesse caso um banco de imagens, é o primeiro passo para aplicar técnicas de ML depois da definição do objetivo do trabalho. As imagens do *dataset* são utilizadas nas etapas de treinamento, validação e teste dos algoritmos e sua qualidade é fundamental para obter bons resultados. Como os métodos utilizados são ML supervisionado, além das imagens é necessária a informação alvo que o algoritmo deve ser capaz de inferir. No caso de classificação de objetos cada arquivo de imagem possui um arquivo de rótulo associado (geralmente codificado em texto), que contém informações de posição na forma de caixas de limite (do inglês, *bounding boxes*) e classes dos objetos presentes na imagem. As *bounding boxes* são codificadas em centro, largura e altura do objeto em pixels.

A criação de *datasets* tradicionalmente exige grande esforço manual para rotular cada objeto na imagem. Além disso, o volume de dados necessários para treinamento de redes de DL é bastante alto, atingindo facilmente a ordem de milhões de imagens, como visto na Seção 2.3 com o *dataset ImageNet* constituído por 1,2 milhões de imagens. Essa combinação de volume e esforço manual é um limitante da utilização de algoritmos de DL. Uma forma de contornar esse problema é construir imagens sintéticas de maneira automatizada. Rodrigues (2020) mostra a criação automática de um *dataset* com imagens de peças mecânicas a partir dos modelos CAD 3D das peças. Além disso, o autor executa o treinamento de algoritmos de ML obtendo bons resultados na tarefa de classificação de objetos nas imagens.

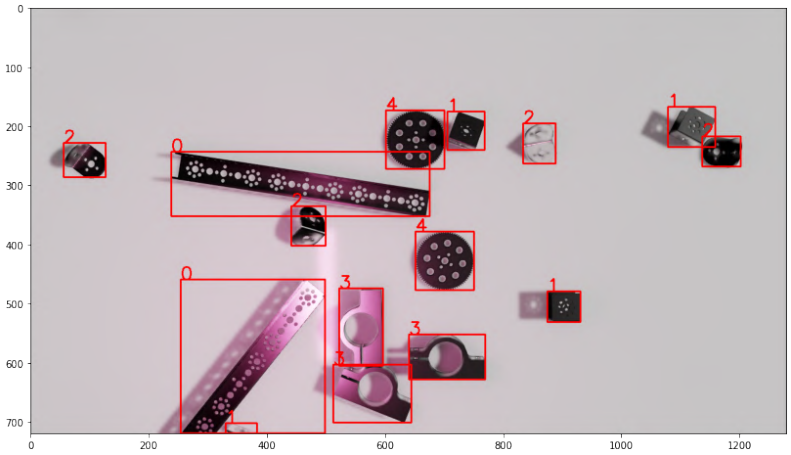
O *dataset* de imagens foi construído de forma automática a partir dos modelos 3D das peças. Isso é realizado dentro da ferramenta de simulação IsaacSim (NVIDIA, 2021), que permite a texturização das peças e aplicação do método de aleatorização de domínio (DR do inglês, *Domain Randomization*). Tobin et al. (2017) apresenta esse método como uma forma de criar variabilidade dos dados simulados. A hipótese testada pelo autor é que se a variabilidade em simulação for significativa, os modelos de ML treinados em simulação serão capazes de generalizar em aplicações com dados reais. Essa variabilidade consiste, no caso do problema de classificação de objetos em imagens, na alteração aleatória de textura, posição, quantidade e iluminação dos objetos simulados. Duas amostras do *dataset* geradas com esse método são apresentadas na Figura 11, onde destacam-se variações de textura, iluminação e quantidade de peças da área de trabalho. Para compor o *dataset* foram geradas aproximadamente 2500 amostras, quantidade considerada suficientes a partir de trabalho anteriores.

Figura 11: *Amostras do dataset aplicando DR.*

Fonte: Elaborada pelo autor.

As informações de rótulo, que codificam a posição e classe dos objetos, também são geradas automaticamente. A Figura 12 exemplifica essas informações plotando-as sobre uma imagem sintética da área de trabalho. A numeração observada na fronteira das *bounding boxes* é a codificação de classe dos objetos.

Figura 12: *Exemplo de amostra do dataset com rótulos bounding boxes e numeração de classes.*



Fonte: Elaborada pelo autor.

3.3.2 Arquitetura da rede neural artificial

A arquitetura de rede neural utilizada neste trabalho foi definida considerando o problema de classificação de objetos. Como introduzido na Seção 2.3.2 as redes convolucionais foram projetadas para tratar com dados do tipo imagem e são bastante estudadas nesse problema específico de classificação. A rede utilizada é uma adaptação da rede convolucional YOLO (do inglês, *You Only Look Once*) originalmente desenvolvida por Redmon, Divvala et al. (2016). A adaptação utilizada é denominada YOLOv5 (JOCHER, 2021).

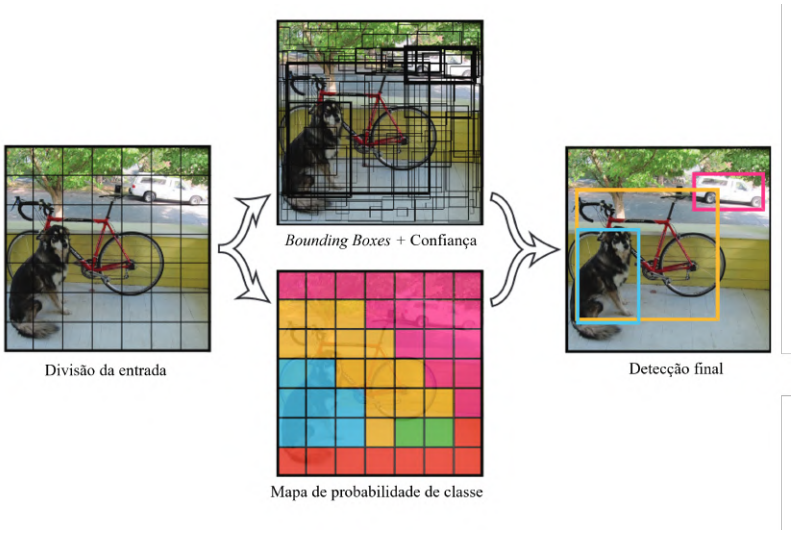
Essa escolha é em função da velocidade de processamento atingida pela YOLO. Redmon, Divvala et al. (2016) destacam que a implementação de algoritmos capazes de detectar rapidamente objetos permite a construção de sistemas robóticos mais responsivos e, também, apresentam resultados de tempo de processamento inferiores a 50 ms. Armstrong (2021) também destaca, no seu trabalho sobre detecção de ervas daninhas em imagens de plantações, a rede YOLO como uma possível alternativa para rápido processamento

de imagens. Quanto ao desempenho de precisão na detecção de objetos as novas versões da YOLO, YOLOv2 (REDMON; FARHADI, Ali, 2017) e YOLOv3 (REDMON; FARHADI, A., 2018) são equiparáveis ao estado da arte e mantém a velocidade de processamento.

Uma vez que o objetivo deste trabalho é desenvolver o DT de um dispositivo robótico com sistema de visão, que atua como sistema de inspeção numa manufatura, o tempo de processamento teve grande peso na escolha.

Resumidamente, o algoritmo selecionado realiza duas tarefas em imagens, a detecção e a classificação dos objetos detectados em uma única arquitetura de rede. A estratégia para executar essas tarefas consiste em fracionar a imagem de entrada com uma grade e, para cada parte da divisão, inferir os limites do objeto na imagem e sua classe. Esse processo é exemplificado na Figura 13, onde a imagem de entrada gera um mapa de *bounding boxes* (do inglês, caixas de limite) que indicam a confiança do algoritmo em existir um objeto na região limitada e um segundo mapa de probabilidade da região representar uma classe do *dataset*. Por final, as *bounding boxes* de menor confiança são eliminadas e as classes predominantes são aplicadas às *bounding boxes* restantes gerando a detecção final.

Figura 13: *Estratégia de detecção e classificação da rede YOLO.*



Fonte: Adaptada de Redmon, Divvala et al. (2016).

3.3.3 Biblioteca de Aprendizado Profundo

As bibliotecas de DL são abstrações de funções matemáticas em uma linguagem de programação, geralmente Python e C++. O seu objetivo é flexibilizar a construção de modelos de aprendizado profundo em pesquisa e aplicações comerciais (STEVENS et al., 2020). Atualmente duas bibliotecas se destacam: PyTorch e TensorFlow, mantidas, respectivamente, pelas empresas Facebook e Google. Neste trabalho é utilizado PyTorch para desenvolvimento das aplicações. Em questão de performance elas são equivalentes e a escolha foi em função da sintaxe do PyTorch.

Uma das vantagens explorada na utilização dessas bibliotecas é a rápida construção de estruturas de redes neurais artificiais através do encadeamento de pequenas funções. O

exemplo da Figura 14 ilustra esse processo, onde uma estrutura de CNN, vista na subseção 2.3.2, é realizada pela classe *MyNeuralNetwork* em linguagem de programação Python. As funções básicas são definidas em *init()* por duas convoluções (*conv1* e *conv2*) e três camadas totalmente conectadas (*fc1*, *fc2* e *fc3*).

Figura 14: *Exemplo de utilização da biblioteca PyTorch em linguagem de programação Python.*

```python
import torch
import torch.nn as nn

class MyNeuralNet():
  def init():
    conv1 = nn.Conv2d(1, 6, 5)
    conv2 = nn.Conv2d(1, 6, 5)
    fc1 = nn.Linear(16 * 5 *5, 120)
    fc2 = nn.Linear(120, 84)
    fc3 = nn.Linear(84, 10)
#...
```

Fonte: Elaborada pelo autor.

3.3.4 Treinamento da rede

A etapa de treinamento da rede YOLO inicia com a divisão do *dataset* em duas partes, uma de treinamento e outra de validação. É comum na literatura uma relação de 80% - 20%, com a maior parte das imagens para o treinamento. A parte de validação é utilizada durante o treinamento para avaliar o desempenho da rede sobre métricas de precisão e ajustar seus parâmetros internos. Adicionalmente, um *dataset* de teste é utilizado após o treinamento para quantificar o desempenho da rede em novas imagens.

Além da divisão do *dataset* é necessário escolher os hiperparâmetros de treinamento. Esses parâmetros não são os internos da rede, como os pesos w vistos na Seção 2.3.1. Os hiperparâmetros principais são taxa de aprendizado (do inglês, *learning rate*) e épocas (do inglês, *epoch*) que definem, respectivamente, o tamanho do passo de minimização e a quantidade de iterações. Cada *epoch* representa uma passagem completa de todas as amostras de treinamento pela rede. Foram utilizados *learning rates* de 0,01 e 50 *epochs* para treinamento.

O principais hiperparâmetros e quantidades de dados utilizados no treinamento são resumidos abaixo:

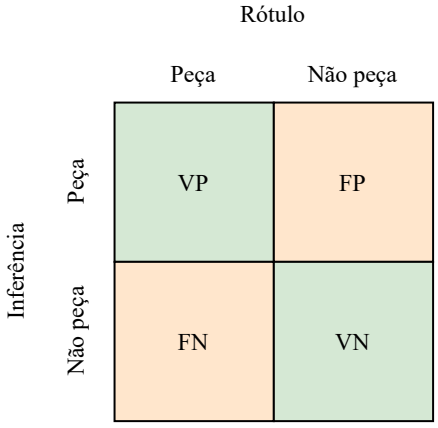
- *Epochs*: 50;
- *Learning rate*: 0,01;
- *Dataset* de treino: 1812 imagens;
- *Dataset* de validação: 773 imagens;
- *Dataset* de teste: 500 imagens.

3.3.5 Validação do treinamento

O desempenho da rede YOLO treinada é avaliado sobre um *dataset* de teste, com 500 imagens não utilizadas durante o treino. Essa avaliação é dividida nas duas tarefas da rede, detecção e classificação de objetos. Para detecção gera-se uma matriz de confusão para quantificar *precision* e *recall*. Para a classificação é construída uma matriz de confusão com tamanho $n \times n$, onde n é o número de classes, peças diferentes, utilizadas no treinamento.

As métricas de *precision* e *recall* representam nesse contexto de detecção de peças, respectivamente, a quantidade de detecções corretas dentro de todas as realizadas e a quantidade de detecções corretas dentro de todos as possíveis. Esse problema gera a matriz binária da Figura 15, onde o algoritmo pode inferir que existe ou não uma peça numa região da imagem. Portanto, o resultado da detecção pode ser uma das quatro opções da matriz, VP Verdadeiro Positivo, FP Falso Positivo, FN Falso Negativo e VN Verdadeiro Negativo. Destaca-se que nesse problema de detecção de peças não existe rotulagem para as regiões sem peças, então, o resultado de VN não será obtido permanecendo nulo na matriz.

Figura 15: *Matriz de confusão para detecção de peças.*



Fonte: Elaborada pelo autor.

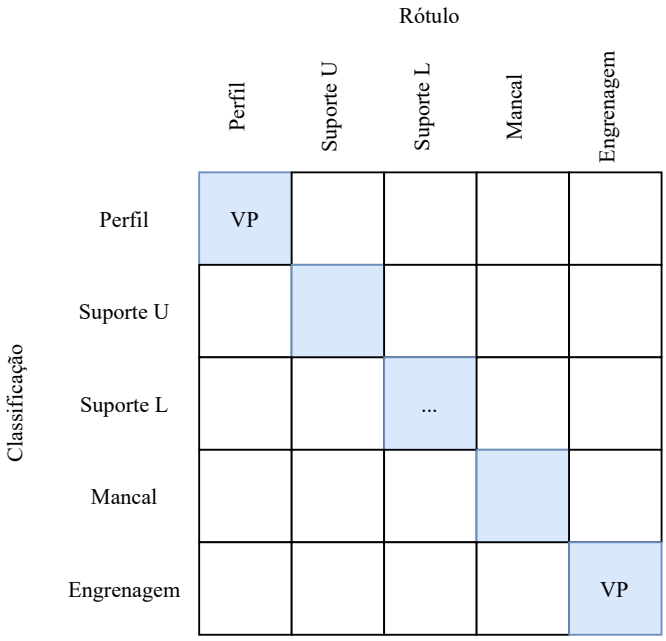
Formalmente, *Precision* e *Recall* são definidas, respectivamente, pelas Equações 3.1 e 3.2.

$$Precision = \frac{VP}{VP + FP} \tag{3.1}$$

$$Recall = \frac{VP}{VP + FN} \tag{3.2}$$

Para a tarefa de classificação é construída uma matriz de confusão de dimensão 5×5, com os cinco tipos de peças descritos na Seção 3.1. O modelo dessa matriz é ilustrado na Figura 16. A diagonal principal representa a quantidade de Verdadeiros Positivos, nas colunas são quantificados os Falsos Negativos e nas linhas os Falsos Positivos.

Figura 16: *Matriz de confusão para classificação.*



Fonte: Elaborada pelo autor.

3.4 Simulação

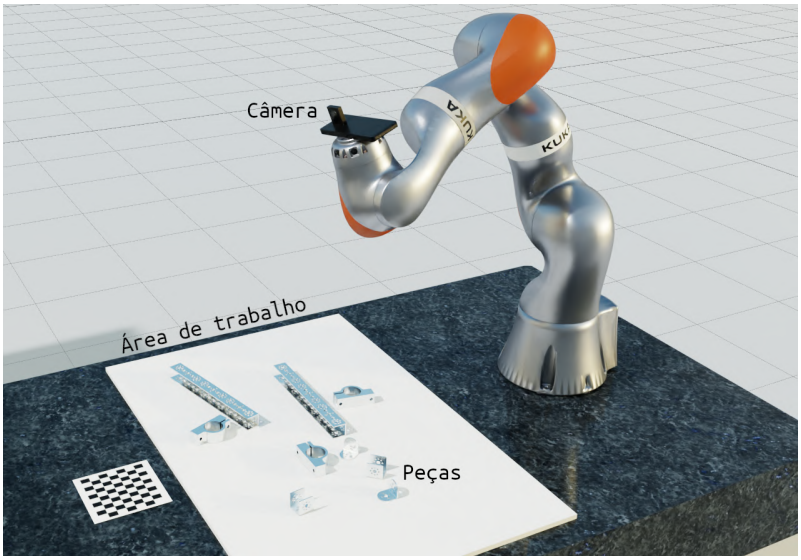
O módulo de simulação tem como objetivo construir um DT do robô e seu ambiente de trabalho e, também, possibilitar integração com o sistema de visão. A simulação foi desenvolvida com o software IsaacSim (NVIDIA, 2021), cuja finalidade é servir como plataforma de desenvolvimento de aplicações de robótica. Além disso, esse software permite simulação de física, como colisões, e visualização fotorealista dos modelos 3D que compõe a simulação, fator importante neste contexto de processamento de imagens.

3.4.1 Modelos 3D

Um dos principais componentes de um DT são representações digitais dos dispositivos reais através de modelos CAD. Dentro do contexto deste trabalho os modelos utilizados são 3D e possuem a função de gerar uma representação gráfica do problema de estudo através da composição de uma cena, ilustrada na Figura 17. Os elementos que compõe tal cena são: robô, área de trabalho e peças para detecção. Também existem componentes sem estrutura 3D na cena como iluminação, câmera e texturas. Todos os componentes da simulação são organizados no formato USD (sigla do inglês, *Universal Scene Description*).

Os modelos 3D das peças são obtidos em repositórios online gratuitos, os arquivos fonte do modelo do robô KUKA iiwa são do repositório ROS Industrial (HOORN, 2021) (ROS do inglês, *Robot Operating System*) e também permitem utilização e modificação livre.

Figura 17: *Cena composta com modelos 3D do robô, peças, câmera e área de trabalho.*



Fonte: Elaborada pelo autor.

3.4.2 Movimento do robô em simulação

A cinemática do robô de estudo KUKA iiwa foi descrita utilizando os parâmetros de DH revisados na Seção 2.2.1. A geometria resulta em parâmetros relativamente simples indicados na Tabela 3.1 abaixo. Como todos os eixos das juntas podem ser alinhados simultaneamente ao eixo $\hat{Z}$ da base, a coluna do parâmetro a_i é nula. Os parâmetros foram atribuídos com o robô totalmente estendido e, adicionalmente, os limites de rotação das juntas foram adicionados à tabela.

Tabela 3.1: *Parâmetros de DH para o robô de estudo KUKA iiwa, mais limites de rotação das juntas.*

i	α_i	d_i	a_i	θ_i	Limites (graus)
1	$-\pi/2$	d_{02}	0	θ_1	$\pm 170°$
2	$\pi/2$	0	0	θ_2	$\pm 120°$
3	$\pi/2$	d_{24}	0	θ_3	$\pm 170°$
4	$-\pi/2$	0	0	θ_4	$\pm 120°$
5	$-\pi/2$	d_{46}	0	θ_5	$\pm 170°$
6	$\pi/2$	0	0	θ_6	$\pm 120°$
7	0	d_{67}	0	θ_7	$\pm 175°$

As distâncias entre os sistemas de coordenadas d_{02}, d_{24}, d_{46} e d_{67} são, respectivamente, 340 mm, 400 mm, 400 mm e 126 mm, conforme site da fabricante.

Dentro do ambiente de simulação o robô é descrito por arquivo URDF (do inglês, *Unified Robot Description Format*), que contém informações de cinemática, dinâmica, modelo 3D e modelo de colisão. A descrição URDF do KUKA iiwa foi implementada com base no repositório ROS Industrial (HOORN, 2021) e modificações foram feitas para a descrição concordar com a realizada através da convenção de DH. Em comparação

à descrição feita via DH haviam sistemas de coordenadas rotacionados em $\hat{Z}$ que foram corrigidos.

Neste trabalho é analisada somente a posição do efetuador do robô simulado, em função da descrição da área de trabalho feita pelo sistema de visão. Não são analisadas velocidades e acelerações do manipulador. A movimentação até a pose indicada pelo resultado do sistema de visão é realizada através da transformação do sistema de coordenadas 2D da imagem, para o sistema 3D da base do robô. Como não existe informação de profundidade com uma câmera monocular foi fixada uma altura z de trabalho para a câmera. Conhecendo também o ângulo de abertura da câmera foi realizada essa transformação entre plano da imagem e plano da área de trabalho.

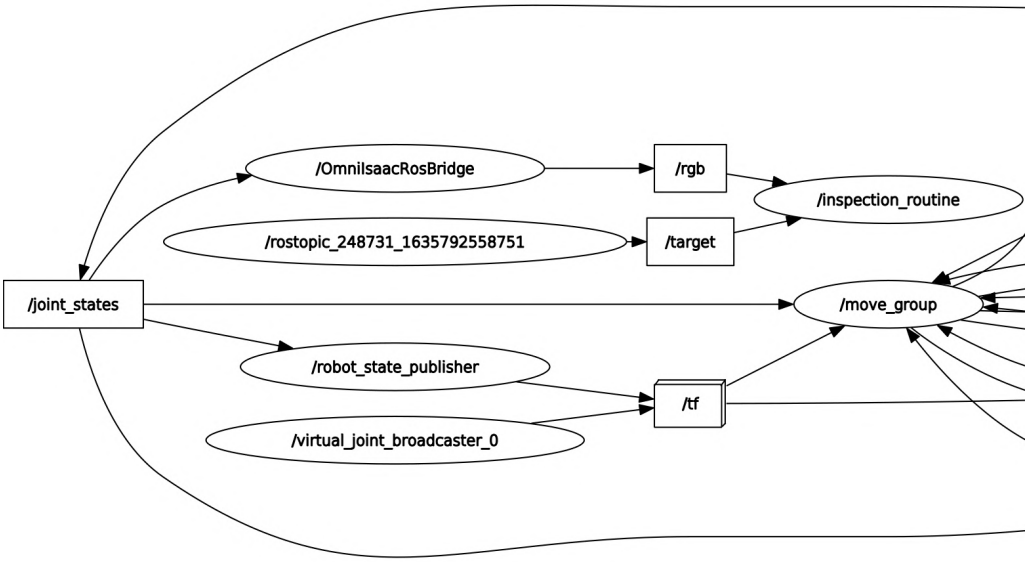
3.5 Controle e Integração visão-robô

A integração entre o sistema de visão e o robô simulado é feita com o software ROS. O ROS é um conjunto de ferramentas e bibliotecas de software livre dedicado ao desenvolvimento de aplicações de robótica. Neste trabalho ele é utilizado para comunicação de dados entre a simulação e o sistema de visão. Essa tarefa é facilitada pelo ROS através de uma padronização das mensagens e estrutura de comunicação *publisher & subscriber* entre nós e tópicos. Um nó, no contexto do ROS, é um programa executável responsável por uma ou mais tarefas. Um tópico pode ser interpretado como um espaço de memória onde os nós podem escrever ou ler mensagens com formatos padronizados.

Um diagrama de computação simplificado com os nós e tópicos do sistema é apresentado na Figura 18, onde os nós são representados por elipses e os tópicos por retângulos. A ferramenta de simulação utilizada, IsaacSim, permite comunicação com ROS através de um nó específico nomeado */OmniIsaacRosBridge*. Um nó */inspection_routine*, externo à simulação, foi construído para controlar a tarefa de inspeção. A comunicação entre esses dois nós é feita pela troca de mensagens nos tópicos. São duas mensagens principais na comunicação, o estado de juntas do robô e a imagem da câmera RGB.

O nó de inspeção é acionado por uma terceira mensagem com o tipo de peça alvo, ou seja, sobre qual tipo de peça deve ser feita uma aproximação e aquisição de imagem em detalhe. Em sequência, esse nó lê as mensagens do tópico */rgb*, que contém imagem da câmera da simulação, e executa uma inferência com a rede YOLO previamente treinada. O resultado dessa inferência retorna uma estimativa das posições (*bounding boxes*) das peças e suas classes.

Figura 18: *Gráfico de computação simplificado com principais nós e tópicos do sistema.*



Fonte: Elaborada pelo autor.

Vale destacar que a verão do ROS utilizada neste trabalho é a Noetic. Todo ambiente de simulação e a rede YOLO são executados em uma máquina Linux com Ubuntu 20.04, processador Intel 10700, 32 Gbytes de memória RAM e placa gráfica NVIDIA RTX 3060.

Neste trabalho são utilizados métodos iterativos do *framework* MoveIt! (COLEMAN et al., 2014) para resolver a cinemática inversa do robô e gerar sua trajetória. O MoveIt! consiste numa ferramenta de software de código aberto, com funcionalidades de planejamento de movimento, manipulação de objetos, cinemática inversa e checagem de colisão.

3.5.1 Calibração *Hand-eye*

O processo de calibração *Hand-eye* foi realizado em simulação posicionando um quadro de marcadores fixos tipo ArUco (GARRIDO-JURADO et al., 2014), ilustrado na Figura 19. Os marcadores são detectados na imagem da câmera simulada por algoritmos da biblioteca OpenCV (2021) e uma pose é estimada para o quadro, ou alvo. O robô é movimentado para 15 poses diferentes e as poses do alvo, câmera e efetuador são salvas compondo, então, o problema $AX = BX$ apresentado na Seção 2.2.2. A solução desse problema é feita numericamente com o método de (TSAI et al., 1989) e a ferramenta MoveIt! (COLEMAN et al., 2014) no software ROS.

Figura 19: *Cena utilizada para calibração Hand-eye com alvo ArUco.*

Fonte: Elaborada pelo autor.

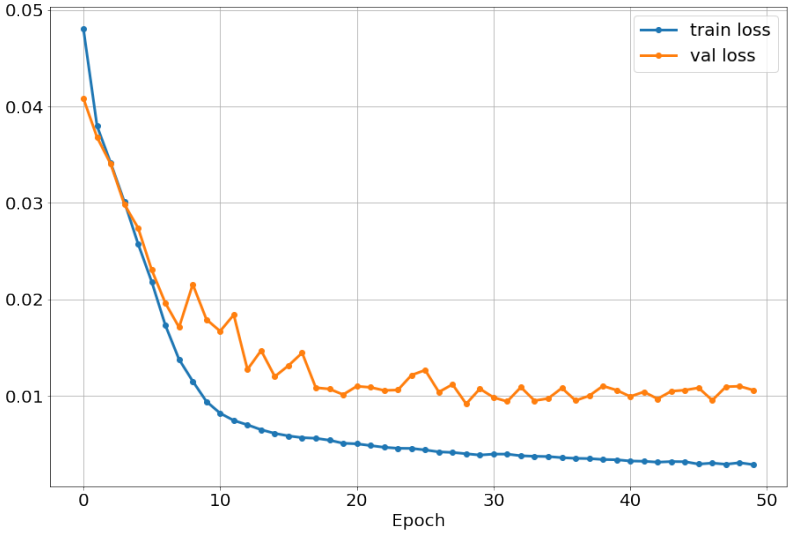
4 RESULTADOS E DISCUSSÕES

Este capítulo apresenta os resultados da rede YOLO nas tarefas de detecção e classificação e, da sua integração no ambiente de simulação. As métricas apresentadas na Seção 3.3.5 são utilizadas para avaliar o desempenho da rede. Uma primeira análise é feita sobre o *dataset* de teste e depois sobre diferentes cenas de simulação. Por final, uma breve análise com imagens reais é apresentada e discutida.

4.1 Detecção e classificação de objetos

Durante a etapa de treinamento também foi analisado o progresso de minimização da função de custo, ou *loss*. Os valores dessa função ao longo das iterações de *epochs* são as somas de erro para cada conjunto de imagens de treino e validação. A redução desse valor, logo do erro, indica melhora de desempenho da rede. Na Figura 20 é apresentado o *loss* de treinamento e validação durante 50 *epochs*. Fica clara uma menor taxa de redução do erro após a vigésima *epoch*, isso também é evidente no gráfico da Figura 21 com as métricas de *precision* e *recall*. Esse comportamento implica que o treinamento poderia ser concluído com pelo menos a metade das *epochs* e na metade do tempo. O tempo total de treinamento para 50 *epochs* foi de 35 minutos com uma placa gráfica NVIDIA RTX 3060.

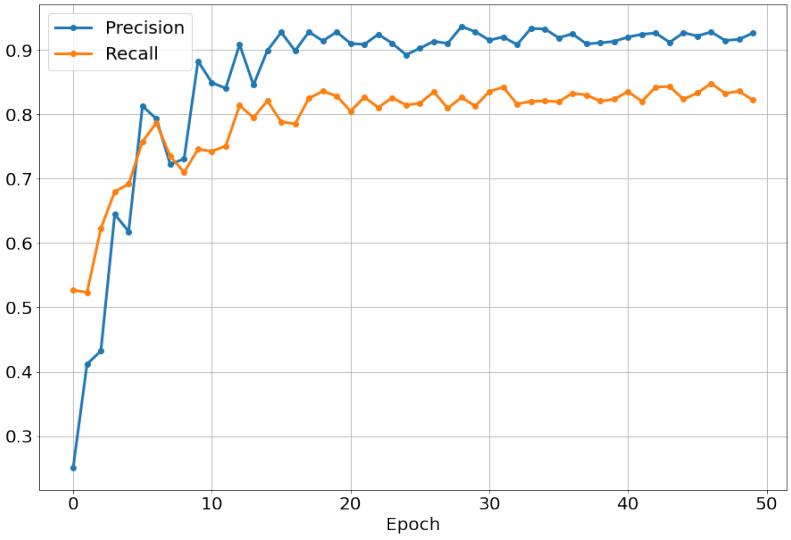
Figura 20: *Minimização das funções de custo de treinamento e validação.*



Fonte: Elaborada pelo autor.

As métricas de *precision* e *recall* ao longo do processo de treinamento são apresentadas na Figura 21. Como observado no gráfico de *loss* o desempenho é estabilizado após 20 *epochs*, a *precision* atinge valores acima de 90% enquanto o *recall* permanece entre 80% e 85%.

Figura 21: *Evolução das métricas de Precision e Recall durante treinamento.*



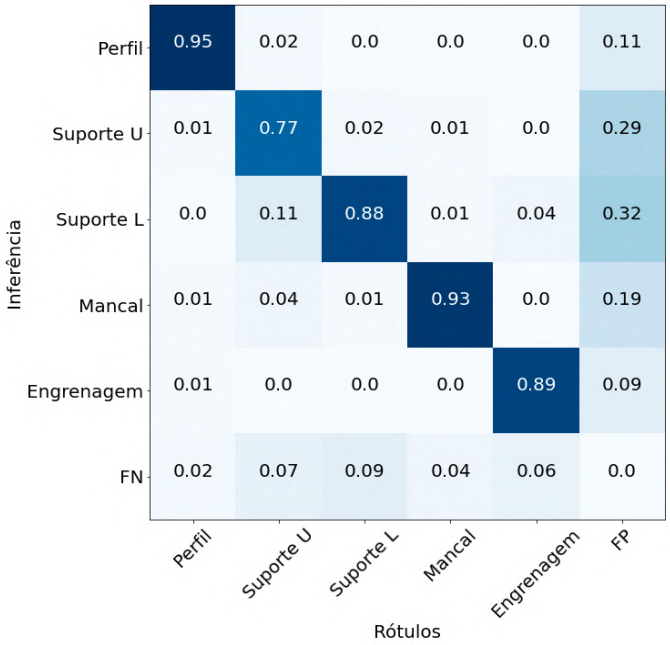
Fonte: Elaborada pelo autor.

As duas tarefas de detecção e classificação foram avaliadas sobre um *dataset* de teste contendo 500 imagens sintéticas, e a matriz de confusão da Figura 22 foi gerada com as cinco classes presentes no *dataset*. Vale destacar que as classes não estão balanceadas no *dataset*, ou seja, existe uma quantidade diferente de amostras para cada peça. Para facilitar a visualização dos resultados na matriz de confusão foram utilizadas taxas no lugar de valores absolutos. Também foi adicionada à matriz uma dimensão extra para representar as detecções Falso Positivas e Falso Negativas. A Tabela 4.1 apresenta as métricas avaliadas por classe de objeto. De modo geral a rede obteve bons desempenhos no *dataset* de teste, com 94,8% de *precision* e 87,7% de *recall*.

Tabela 4.1: *Resumo dos resultados de classificação da rede YOLO.*

Classe	**Imagens**	**Rótulos**	***Precision***	***Recall***
Perfil	500	939	0,966	0,950
Suporte U	500	1396	0,951	0,764
Suporte L	500	1724	0,881	0,852
Mancal	500	1317	0,952	0,920
Engrenagem	500	821	0,991	0,901
Geral	500	6197	0,948	0,877

Figura 22: *Matriz de confusão.*



Fonte: Elaborada pelo autor.

4.2 Análise da integração visão-robô

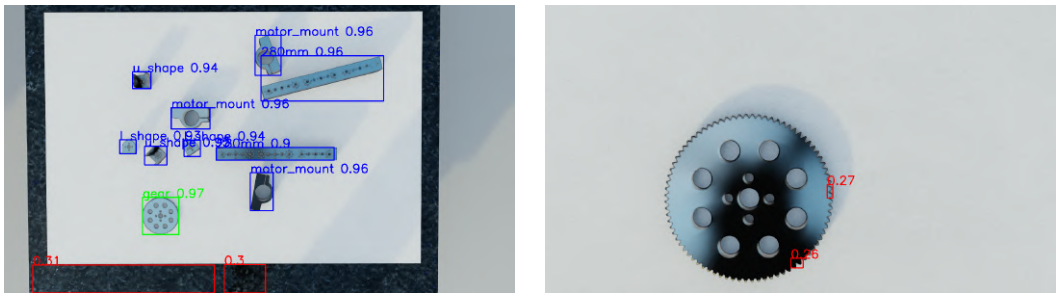
A análise da integração entre o algoritmo de processamento de imagens e o robô KUKA tem como objetivo principal verificar se o sistema de visão é capaz de gerar descrições consistentes para orientar as ações do robô.

O primeiro teste do sistema completo realizando a tarefa de inspeção é descrito em três etapas: detecção de objetos na área de trabalho, aproximação do robô e detecção do objeto em imagem aproximada. Esse cenário de inspeção exige que a rede YOLO seja capaz de reconhecer o mesmo objeto com tamanhos diferentes, em uma imagem distante e outra próxima da área de trabalho.

O resultado obtido no primeiro teste foi uma falha de detecção na imagem próxima, ou seja, a rede não foi capaz de indicar a posição ou classe do objeto na imagem. Esse resultado é ilustrado na Figura 23. A Figura 24a mostra a correta detecção das peças sobre a área de trabalho, com destaque à peça alvo em verde. No entanto, ao aproximar a câmera a peça alvo não é detectada (Figura 24b). Os valores numéricos nas imagens indicam a confiança do resultado, valores inferiores a 0,8 são destacados com bordas vermelhas, superiores em azul e a peça alvo em verde.

Figura 23: *Detecção e classificação da rede YOLO.*

(a) *Correta detecção em imagem distante.* **(b)** *Falha de detecção em imagem próxima.*



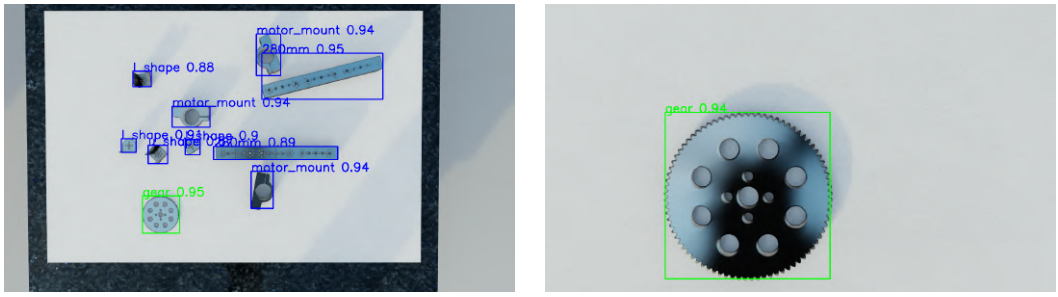
Fonte: Elaborada pelo autor.

Esse comportamento é devido à falta de amostras que representem peças próximas à câmera no *dataset* de treinamento. Para melhorar o desempenho da rede foram geradas 500 novas imagens sintéticas com a câmera próxima à área de trabalho, e uma nova rede YOLO foi treinada. Esse processo de retreinamento e ajuste de dados é comum no ciclo de vida de algoritmos de DL, como indicado na Seção 3.3.

Um segundo teste foi realizado na simulação e obteve-se correta detecção nas diferentes distâncias de câmera, como ilustra a Figura 24. Vale destacar que o tempo necessário para gerar as 500 novas imagens foi de 33 minutos, e o retreinamento da rede usou 22 minutos.

Figura 24: *Detecção e classificação da rede YOLO treinado com amostras adicionais.*

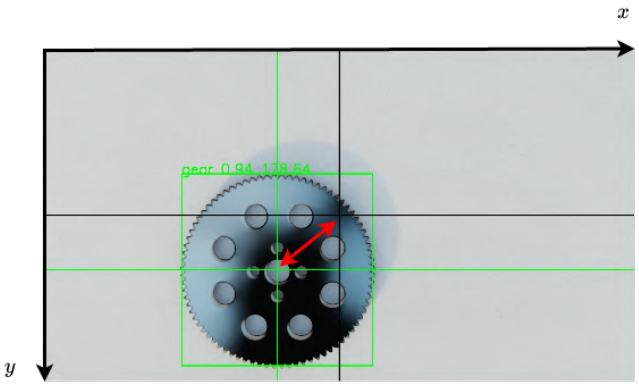
(a) *Correta detecção em imagem distante.* **(b)** *Correta detecção em imagem próxima.*



Fonte: Elaborada pelo autor.

Além da correta detecção de peças é necessário que o robô execute o movimento de aproximação na direção da peça alvo. Para quantificar o desempenho de aproximação foi calculada a distância entre o centro da peça alvo e o centro da imagem, conforme Figura 25. A distância é codificada em pixels. Ao final de cada aproximação é executada uma inferência da rede YOLO sobre a imagem. O tempo de inferência foi da ordem de 58 ms, com processamento em placa gráfica.

Figura 25: *Erro de aproximação do efetuador.*



Fonte: Elaborada pelo autor.

Foram coletadas 92 amostras de imagens aproximadas de dimensão 1280x720 pixels, como da Figura 25, e para cada uma foi calculada a distância entre o centro do objeto e o centro da imagem (em coordenadas x e y). Os resultados dessa avaliação são apresentados na Tabela 4.2. Percebe-se alta dispersão dos dados, o que foi julgado como um baixo desempenho na tarefa de aproximação. Esse problema pode ter mais de uma fonte, como configurações do algoritmo MoveIt!, incorreta detecção e calibração extrínseca da câmera. Todavia, esse erro de posicionamento não prejudicou o sistema na classificação de peças pequenas, um ajuste na altura de operação da câmera basta para capturar toda a peça na imagem. Em uma tarefa com especificações mais rígidas poderia ser implementado um controle da pose do efetuador através da realimentação de novas detecções.

Tabela 4.2: *Distâncias do objeto detectado para o centro da imagem.*

	Distâncias (pixels)		
	Distancia	**x**	**y**
Amostras	92	92	92
Média	319,68	-47,61	-35,68
Desvio Padrão	164,80	314,82	166,71
Mínimo	43,10	-584,00	-347,00
Máximo	656,88	630,00	322,00

4.3 Detecção e classificação em imagens reais

Dada a disponibilidade das peças reais, foi construído um pequeno *dataset* com 20 imagens para testar o desempenho do algoritmo treinado. Os rótulos para cada objeto nas imagens foram feitos manualmente. A matriz de confusão para detecção é apresentada na Figura 26a, e os resultados de *precision* e *recall* são, respectivamente, 79% e 100%. Esse nível de *recall* significa que todas os objetos rotulados foram detectados, porém algumas detecções foram feitas em locais sem peças, reduzindo a métrica de *precision*. A seta na Figura 27 indica um local sem peça com detecção equivocada. Os objetos falsos detectados são detalhes do ambiente não modelados em simulação, o que indica resultados

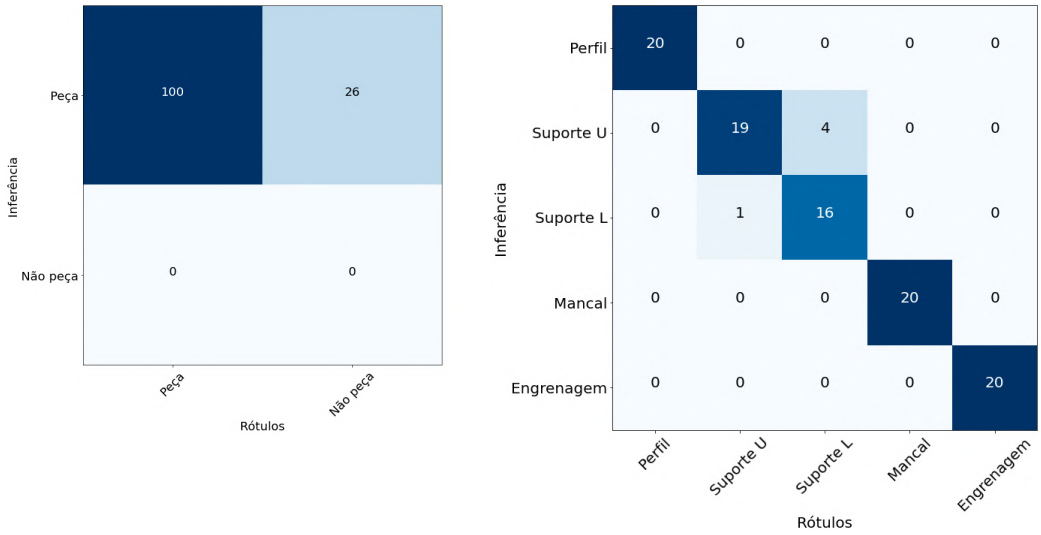
potencialmente melhores em ambientes mais controlados. Também é importante observar que, devido ao pequeno número de imagens no *dataset*, qualquer erro de detecção tem potencial de causar forte alteração nas métricas avaliadas, em especial na *precision*.

Quanto à classificação dos objetos, foi obtida uma média de 95% de *precision* (Figura 26). Esse teste com dados reais ainda possui pouco volume para validar a transferência do treinamento em simulação para o ambiente real, mas indica um caminho de melhoria na caracterização da área de trabalho do robô.

Melhores resultados poderiam ser obtidos caso o algoritmo fosse testado com *datasets* maiores compostos de imagens reais. Para a geração desses *datasets* podem ser usados, em estudos futuros, métodos de *data augmentation*, visando a obtenção de imagens geradas por computados a partir da alteração de parâmetros de imagens reais.

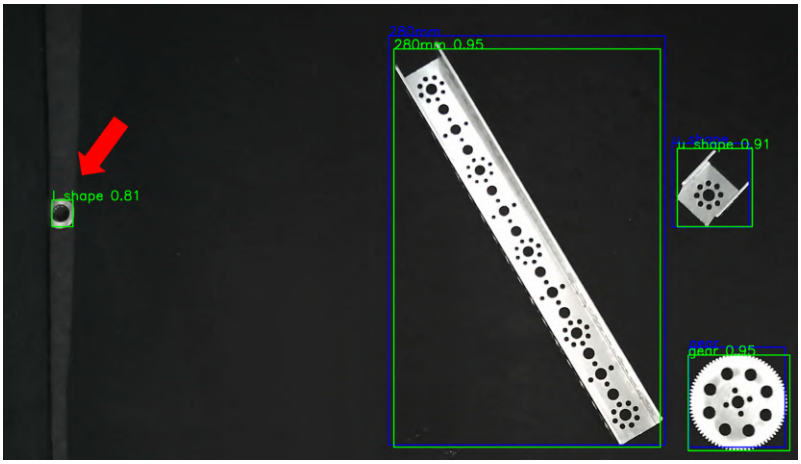
Figura 26: *Matrizes de confusão com resultados do teste em imagens reais.*

(a) *Detecção.* **(b)** *Classificação.*



Fonte: Elaborada pelo autor.

Figura 27: *Resultado de detecção e classificação em imagem real, com indicação de falhas.*



Fonte: Elaborada pelo autor.

5 CONCLUSÕES

Este trabalho propôs a construção de uma versão digital de um sistema robótico em uma tarefa de inspeção de peças. A tarefa de inspeção foi realizada com técnicas de ML e integrada às ações do robô. Para isso, iniciou-se por uma revisão de conceitos e tecnologias, como forma de analisar fundamentos e o estado da arte. A partir dessa revisão foram fundamentadas as escolhas da rede YOLO, como algoritmo de processamento de imagens, e ferramentas de simulação para desenvolver o *digital twin*.

Uma arquitetura de sistema em módulos foi aplicada para facilitar o desenvolvimento e integração das partes do trabalho. O uso do software ROS em conjunto com o simulador IsaacSim facilitou essa divisão, em razão dos dados das diferentes partes do sistema utilizarem um único canal de comunicação com mensagens padronizadas.

O sistema de visão, composto pela rede YOLO e o banco de dados, atingiu bons resultados nas tarefas de detecção e classificação de objetos. As métricas de *precision* e *recall* foram, respectivamente, 94,8% e 87,7%. Além disso, a utilização de dados sintéticos no treinamento proporcionou rapidez de testes e retreinamento frente a baixos desempenhos na etapa de validação. Por exemplo, a falha de detecção de objetos em imagens próximas da área de trabalho foi rapidamente solucionada em ambiente de simulação com geração de novos dados, o que também indica que a simulação é um meio rápido de iterar ciclos de vida de algoritmos de ML, adicionando novos dados e testando novos cenários rapidamente.

A integração entre as partes do sistema proposto foi realizada, os dados da simulação foram processados pelos nós do ROS e resultados reenviados ao robô simulado. Porém, a integração do sistema de visão (câmera mais rede YOLO) com o robô exige um estudo mais detalhado de calibração, visto que foi obtida uma alta variação da distância entre o ponto "alvo" da imagem e o ponto atingido após movimentação. Uma rotina de calibração e novos padrões estáticos podem ser estudados nesse sentido.

A utilização de ferramentas do estado da arte para simulação permitiu a construção e teste de um sistema complexo e se mostraram promissoras para desenvolver a lógica ou treinamento (no contexto de ML) de sistemas de forma totalmente digital, resumindo a implementação física em processos de calibração e atualização de software. A continuação deste trabalho é a integração do sistema digital com o físico. Alguns pontos importantes como calibração, melhor representação do ambiente real nos dados sintéticos e teste em imagens reais (com e sem *data augmentation*) já começaram a ser abordados no decorrer deste estudo, e podem guiar novos desenvolvimentos.

REFERÊNCIAS

AKINOLA, Iretiayo et al. Dynamic Grasping with Reachability and Motion Awareness. *arXiv preprint arXiv:2103.10562*, 2021.

ARMSTRONG, David Rutherford. *Machine Learning For Identification and Classification of Crops and Weeds*. *UFRGS*, Porto Alegre - Brasil, 2021.

BOSCHERT, S.; ROSEN, R. *Digital Twin—The Simulation Aspect*. [S.l.: s.n.], 2016. p. 59–74.

BURKOV, A. *Machine Learning Engineering*. [S.l.: s.n.], 2020.

CAULIFIELD, B. *NVIDIA, BMW Blend Reality, Virtual Worlds to Demonstrate Factory of the Future*. [S.l.: s.n.], 2021. Acesso em outubro de 2021. Disponível em: <<https://blogs.nvidia.com/blog/2021/04/13/nvidia-bmw-factory-future/>>.

COLEMAN, David et al. Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study. *Journal of Software Engineering for Robotics*, v. 5, n. 1, p. 3–16, 2014.

CRAIG, J. J. *Introduction to Robotics Mechanics and Control*. [S.l.: s.n.], 1986.

DENG, Jia et al. ImageNet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition. [S.l.: s.n.], 2009. p. 248–255.

GARRIDO-JURADO, S. et al. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, v. 47, n. 6, p. 2280–2292, 2014.

GLAESSGEN, E. H.; STARGEL, D. S. *The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles*. [S.l.: s.n.], 2012.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.: s.n.], 2016.

GRIEVES, M. *Digital Twin: Manufacturing Excellence through Virtual Factory Replication*. [S.l.: s.n.], 2015.

HOORN, G. Van der. *kuka-experimental*. [S.l.: s.n.], 2021. Acesso em outubro de 2021. Disponível em: <<https://github.com/ros-industrial/kuka_experimental>>.

HORAUD, R.; DORNAIKA, F. *Hand-eye calibration*. *The International Journal of Robotics Research*, v. 14, n. 3, p. 195–210, 1995.

JOCHER, Glenn. *YOLOv5*. [S.l.: s.n.], 2021. Acesso em outubro de 2021. Disponível em: <<https://github.com/ultralytics/yolov5>>.

KOUSI, N. et al. Digital twin for adaptation of robots' behavior in flexible robotic assembly lines. *Procedia Manufacturing*, v. 28, p. 121–126, 2019.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks. In: ADVANCES in Neural Information Processing Systems 25. [S.l.: s.n.], 2012. v. 25, p. 1097–1105.

KRÜGER, J. et al. Innovative control of assembly systems and lines. *CIRP Annals*, v. 66, n. 2, p. 707–730, 2017.

KUKA. *LBR iiwa*. [S.l.: s.n.], 2021. Acesso em outubro de 2021. Disponível em: <<https://www.kuka.com/pt-br/produtos-servicos/sistemas-de-roboticos/robos-industriais/lbr-iiwa>>.

LECUN, Yann; BENGIO, Yoshua. Convolutional networks for images, speech, and time series. In: p. 255–258.

LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep learning. *Nature*, v. 521, n. 7553, p. 436–444, 2015.

NVIDIA. *Isaac Sim Overview*. [S.l.: s.n.], 2021. Acesso em setembro de 2021. Disponível em: <<https://docs.omniverse.nvidia.com/app_isaacsim/app_isaacsim/overview.html>>.

OPENAI. Solving Rubik's Cube with a Robot Hand. *arXiv preprint arXiv:1910.07113*, 2019.

OPENCV. *Camera Calibration and 3D Reconstruction*. [S.l.: s.n.], 2021. Acesso em outubro de 2021. Disponível em: <<https://docs.opencv.org/3.4.15/d9/d0c/group__calib3d.html#gaebfc1c9f7434196a374c382abf43439b>>.

REDMON, Joseph; DIVVALA, Santosh et al. You Only Look Once: Unified, Real-Time Object Detection. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [S.l.: s.n.], 2016. p. 779–788.

REDMON, Joseph; FARHADI, A. YOLOv3: An Incremental Improvement. *arXiv preprint arXiv:1804.02767*, 2018.

REDMON, Joseph; FARHADI, Ali. YOLO9000: Better, Faster, Stronger. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [S.l.: s.n.], 2017. p. 6517–6525.

RODRIGUES, João Vitor. *Classificação de peças mecânicas a partir de visão computacional e aprendizado de máquina utilizando imagens sintéticas*. *UFRGS*, Porto Alegre - Brasil, 2020.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, v. 65, n. 6, p. 386–408, 1958.

RUSSELL, Stuart J.; NORVIG, Peter. *Artificial Intelligence: A Modern Approach*. [S.l.: s.n.], 2020.

SHAFTO, M. et al. *DRAFT Modeling, Simulation, Information Technology Processing Roadmap*. [S.l.]: NASA, 2010.

STEVENS, Eli; ANTIGA, Luca. *Deep Learning with Pytorch*. [S.l.: s.n.], 2020.

TAO, Fei; ZHANG, Meng; NEE, A.Y.C. *Digital Twin Driven Smart Manufacturing*. [S.l.: s.n.], 2019. p. 3–5.

THEODORIDIS, Sergios; KOUTROUMBAS, Konstantinos. *Pattern Recognition, Third Edition*. [S.l.: s.n.], 2006.

TOBIN, Josh et al. Domain randomization for transferring deep neural networks from simulation to the real world. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). [S.l.: s.n.], 2017. p. 23–30.

TSAI, R.Y.; LENZ, R.K. A new technique for fully autonomous and efficient 3D robotics hand/eye calibration. *international conference on robotics and automation*, v. 5, n. 3, p. 345–358, 1989.

VISION, Kinetic. *True Digital Twin*. [S.l.: s.n.], 2020. Acesso em outubro de 2021. Disponível em: <<https://kinetic-vision.com/true-digital-twin/>>.