INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO AMAZONAS
CAMPUS MANAUS-DISTRITO INDUSTRIAL
CURSO DE BACHARELADO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

PAULO RAFAEL RODRIGUES FEITOSA

SIMULAÇÃO DE UM SISTEMA AUTOMATIZADO DE INSPEÇÃO DE RÓTULOS DE BATERIAS UTILIZANDO WEBOTS

MANAUS - AM
2024

PAULO RAFAEL RODRIGUES FEITOSA

SIMULAÇÃO DE UM SISTEMA AUTOMATIZADO DE INSPEÇÃO DE RÓTULOS DE BATERIAS UTILIZANDO WEBOTS

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia do Amazonas, Campus Manaus - Distrito Industrial, Curso de Bacharelado em Engenharia de Controle e Automação, como requisito parcial para obtenção do título de Bacharel em Engenharia de Controle e Automação.

Orientador: Prof. Dr. Alyson de Jesus dos Santos

Manaus-AM
2024

Dados Internacionais de Catalogação na Publicação (CIP)

F311s

Feitosa, Paulo Rafael Rodrigues.

Simulação de um sistema automatizado de inspeção de rótulos de baterias utilizando *webots* / Paulo Rafael Rodrigues Feitosa. — Manaus, 2024.

77f.: il. color.

Monografia (Graduação) — Instituto Federal de Educação, Ciência e Tecnologia do Amazonas, *Campus* Manaus Distrito Industrial, Curso de Engenharia de Controle e Automação, 2024.

Orientador: Prof.º Alyson de Jesus dos Santos, Dr.

1. Webots. 2. Python. 3. Pytesseract. 4. Teachable machine. I. Feitosa, Paulo Rafael Rodrigues. II. Instituto Federal de Educação, Ciência e Tecnologia do Amazonas. III. Título.

CDD 629.89

Elaborada por Oziane Romualdo de Souza (CRB11/ nº 734)

ATA DE DEFESA PÚBLICA DO TRABALHO DE CONCLUSÃO DE CURSO

Aos 05 dias do mês de Agosto, de 2024, às 18h 30minutos, o(a) discente Paulo Rafael Rodrigues Feitosa apresentou o seu Trabalho de Conclusão de Curso para avaliação da Banca Examinadora constituída pelos seguintes integrantes: Prof(a). Dr. Alyson de Jesus dos Santos (docente-orientador), Prof(a). (Gabriel Rebello Guerreiro) e Prof(a). (Daniel Fonseca de Souza). A sessão publica de defesa foi aberta pelo(a) presidente da banca, que apresentou a Banca Examinadora e deu continuidade aos trabalhos, fazendo uma breve referência ao TCC, que tem como título Simulação de um Sistema Automatizado de Inspeção de Rótulos de Baterias Utilizando Webots. Na sequência, o(a) discente teve até 30 minutos para a comunicação oral de seu trabalho. Cada integrante da banca examinadora fez suas arguições após a defesa do mesmo. Ouvidas as explicações do(a) discente, a banca examinadora, reunida em caráter sigiloso, para proceder à avaliação final, deliberou e decidiu pela APROVAÇÃO com média final **Oito, sete (8,7)** do referido trabalho.

Foi dada ciência ao(à) discente que a versão final do trabalho deverá ser entregue até o dia 05/09/2024, com as devidas alterações sugeridas pela banca. Nada mais havendo a tratar, a sessão foi encerrada às 20h20 min, sendo lavrada a presente ata, que, uma vez aprovada, foi assinada por todos os membros da Banca Examinadora e pelo(a) discente.

Prof.(a) Orientador(a)/Presidente: Alyson de Jesus dos Santos

Prof.(a) Avaliador 1: Gabriel Rebello Guerreiro

Prof.(a) Avaliador 2: Daniel Fonseca de Souza

Discente: Paulo Rafael Rodrigues Feitosa

À Deus e minha família.

AGRADECIMENTOS

Gostaria de agradecer primeiramente a Deus pela minha vida e por sua infinita graça e amor. Agradeço por ter me dado a oportunidade de fazer este curso e pela força proporcionada ao longo dessa caminhada.

Agradeço e dedico este trabalho aos meus pais e meus irmãos, pelo suporte e apoio que me deram para atingir este sonho, e por toda a paciência que tiveram comigo durante os momentos de dificuldades, sempre acreditando no meu potencial.

Agradeço também ao meu orientador, Prof. Dr. Alyson, por toda a orientação e colaboração durante este trabalho.

Por fim, agradeço a todos os professores pelo conhecimento transmitido e a todos os colegas que me ajudaram em algum momento.

O temor do Senhor é o princípio do saber

(Provérbios 1:7)

RESUMO:

Este trabalho consiste em uma simulação no Webots de um sistema de inspeção visual para informações gravadas em rótulos de baterias de celular por meio de tampografia. O ambiente possui uma esteira de entrada onde as baterias são inseridas. Uma câmera acoplada a um robô UR5e captura as imagens das baterias e as envia para o algoritmo de inspeção feito em Python. O algoritmo de inspeção é composto de um sistema OCR (Optical Character Recognition) para verificar se as informações textuais estão presentes no rótulo, e um modelo de classificação para verificar se os símbolos foram gravados no rótulo. O robô UR5e então separa as baterias aprovadas e reprovadas.

Palavras-chave: Webots. Python. Pytesseract. Teachable Machine.

ABSTRACT:

This work consists of a simulation in Webots of a visual inspection system for information printed on cell phone battery labels using pad printing. The environment includes an entry conveyor where the batteries are inserted. A camera attached to a UR5e robot captures images of the batteries and sends them to the inspection algorithm developed in Python. The inspection algorithm comprises an OCR (Optical Character Recognition) system to check if the textual information is present on the label and a classification model to verify if the symbols have been printed on the label. The UR5e robot then separates the approved and rejected batteries.

Keywords: Webots. Python. Pytesseract. Teachable Machine.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API Application Programming Interface

CNPJ Cadastro Nacional da Pessoa Jurídica

GIF Graphics Interchange Format

GUI Graphical User Interface

IDE Integrated Development Environment

OCR Optical Character Recognition

OPENCV Open Source Computer Vision Library

PCA Principal Component Analysis

PIM Polo Industrial de Manaus

PNG Portable Network Graphics

RGB Red, Green, Blue

ROS Robot Operating System

SVM Support Vector Machine

TCC Trabalho de Conclusão de Curso

TIFF Tagged Image File Format

VRML Virtual Reality Modeling Language

SUMÁRIO

1 INTRODUÇÃO

A automação industrial tem se tornado uma componente essencial na modernização dos processos produtivos, permitindo aumentos significativos na eficiência, precisão e segurança. O uso de sistemas automatizados possibilita a execução de tarefas repetitivas de maneira rápida e precisa, reduzindo a intervenção humana e minimizando erros (Locks et al., 2018). Com a integração de tecnologias avançadas, a automação industrial não apenas otimiza a produção, mas também melhora a qualidade dos produtos e a segurança no ambiente de trabalho (Sauer et al., 2013).

A visão computacional, uma subárea da inteligência artificial, desempenha um papel crucial na automação moderna, proporcionando às máquinas a capacidade de "enxergar" e interpretar o mundo ao seu redor (Svensson et al., 2021). Sistemas de visão computacional utilizam câmeras e algoritmos para processar e analisar imagens, permitindo a identificação e classificação de objetos com alta precisão (Schwabe & Castellacci, 2020). Essa tecnologia é amplamente aplicada em diversos setores industriais, desde a inspeção de qualidade até o controle de processos e a robótica.

No contexto de inspeção automatizada, a visão computacional se destaca por sua capacidade de realizar verificações detalhadas e consistentes, algo que seria difícil de manter apenas com inspeção humana (Welfare et al., 2019). Através do uso de algoritmos de aprendizado de máquina, os sistemas de visão computacional podem ser treinados para reconhecer padrões específicos e detectar defeitos ou irregularidades em produtos. Esse nível de detalhamento e precisão é essencial para garantir a qualidade em processos de fabricação e montagem.

A simulação de sistemas representa uma ferramenta indispensável no desenvolvimento de projetos de automação. Utilizar ambientes simulados permite testar e validar diferentes configurações e estratégias antes da implementação física, reduzindo significativamente os custos e os riscos associados a erros de design e implementação (Tsao et al., 2020). Ferramentas de simulação como o Webots oferecem um ambiente controlado onde se podem

modelar e experimentar diferentes cenários, facilitando a identificação de problemas e a otimização de soluções. A capacidade de simular a interação entre diversos componentes, como robôs, câmeras e esteiras, acelera o processo de desenvolvimento e assegura maior precisão na fase de implementação (Radi et al., 2010).

O reconhecimento óptico de caracteres (OCR) é uma tecnologia específica dentro da visão computacional, voltada para a identificação e extração de texto de imagens. Em ambientes industriais, o OCR é utilizado para verificar a presença e a legibilidade de informações impressas em produtos, como números de série, datas de validade e códigos de barras. A aplicação de OCR em sistemas automatizados de inspeção permite uma verificação rápida e precisa, garantindo que todos os produtos atendam aos requisitos de qualidade e conformidade antes de serem enviados ao mercado (Khastgir et al., 2018).

1.1 JUSTIFICATIVA

Durante um período de trabalho em uma fábrica de baterias, foi identificado um processo denominado tampografia, utilizado para gravar informações no corpo das células de baterias. A tampografia é uma técnica de impressão que utiliza um tampão de silicone para transferir tinta de uma placa gravada para a superfície do objeto a ser marcado. Esse método é amplamente empregado para imprimir em superfícies irregulares ou texturizadas, garantindo alta precisão e qualidade na gravação de informações como datas de fabricação, números de lote e especificações técnicas. A tampografia é valorizada por seu baixo custo por unidade produzida e pela sua capacidade de realizar impressões rápidas e precisas em diversos tipos de superfícies e materiais, sendo uma escolha comum na indústria de eletroeletrônicos, cosméticos e brinquedos (Oscar Flues, 2023).

Após a gravação das informações, uma inspeção manual é realizada para verificar se as informações foram impressas corretamente. Contudo, o uso de operadores humanos para esta tarefa apresenta várias desvantagens significativas. Primeiramente, a inspeção manual é suscetível a erros humanos,

como fadiga e distração, que podem levar à falha na detecção de defeitos ou informações incorretas. Além disso, o processo manual é lento e pode limitar a capacidade de produção da fábrica, uma vez que a velocidade de inspeção está diretamente ligada à capacidade do operador.

Essas limitações motivaram o desenvolvimento de uma automação para realizar a tarefa de inspeção de forma mais eficiente e precisa. A automação deste processo não apenas elimina a possibilidade de erro humano, mas também aumenta a velocidade e a consistência da inspeção.

Neste projeto, a simulação de um sistema de inspeção no Webots visa criar um ambiente virtual onde a automação pode ser testada e otimizada. Utilizando uma esteira para a movimentação das placas, uma câmera para a captura de imagens e um robô UR5e para a separação de placas aprovadas e reprovadas, o sistema permitirá a validação de algoritmos de OCR e classificação e a avaliação de seu desempenho em um cenário controlado.

1.2 OBJETIVO GERAL

Desenvolver uma simulação no Webots de um sistema automatizado com reconhecimento óptico de caracteres (OCR) e um classificador para a inspeção de placas, verificando a presença e a legibilidade de informações impressas e a presença de símbolos.

1.3 OBJETIVOS ESPECÍFICOS

a) Construir o ambiente virtual no Webots, incluindo a modelagem de uma esteira de entrada;

b) Desenvolver o controlador do robô UR5e responsável por separar as baterias aprovadas e reprovadas;

c) Integrar o Webots com o Pycharm para possibilitar a execução do algoritmo de OCR e classificação em Python;

d) Realizar a aquisição das imagens das baterias através de uma câmera posicionada estrategicamente;

e) Implementar o algoritmo de OCR para a extração precisa dos caracteres das imagens adquiridas.

f) Desenvolver um algoritmo de classificação para detectar a presença de símbolos na bateria.

1.4 ESTRUTURA DO TCC

Este trabalho está estruturado de forma a apresentar de maneira detalhada todas as etapas do desenvolvimento do projeto. No Capítulo 1, será introduzido o contexto do projeto, destacando a importância da automação, visão computacional e OCR no ambiente industrial. O Capítulo 2 apresentará a fundamentação teórica, abordando os conceitos essenciais e tecnologias utilizadas, como o Webots, Python, e algoritmos de visão computacional. No Capítulo 3, será descrita a metodologia aplicada, detalhando os processos de construção do ambiente virtual, desenvolvimento do controlador do robô, integração com Pycharm, e implementação do algoritmo de OCR e classificação. O Capítulo 4 focará na apresentação e discussão dos resultados obtidos, incluindo a análise de desempenho do sistema simulado e os testes realizados. Por fim, o Capítulo 5 trará as considerações finais, sintetizando as conclusões do estudo e propondo direções para trabalhos futuros, além de discutir as possíveis melhorias e aplicações práticas do sistema desenvolvido.

2 REFERENCIAL TEÓRICO

O referencial teórico busca expor os principais conceitos e tecnologias que serão utilizados nesse trabalho, começando pela discussão dos conceitos de visão computacional e imagens, e depois a apresentação dos elementos de software e hardware utilizados no projeto.

2.1 WEBOTS

O Webots é um simulador de código aberto utilizado para modelar, programar e simular robôs em um ambiente virtual tridimensional. Desenvolvido pela Cyberbotics, o Webots oferece uma plataforma robusta e flexível para a pesquisa, ensino e desenvolvimento de robótica, permitindo a criação de mundos simulados complexos que replicam cenários reais de maneira altamente precisa.

Figura 1 – Logotipo Webots

Fonte: CYBERBOTICS (2023)

Uma das principais características do Webots é a sua capacidade de integrar diversos tipos de robôs, sensores e atuadores em um único ambiente simulado. Isso facilita a experimentação e o teste de algoritmos de controle e navegação sem a necessidade de hardware físico. O Webots suporta uma ampla gama de dispositivos, desde simples sensores de distância até complexos braços robóticos, proporcionando um ambiente completo para o desenvolvimento de soluções robóticas.

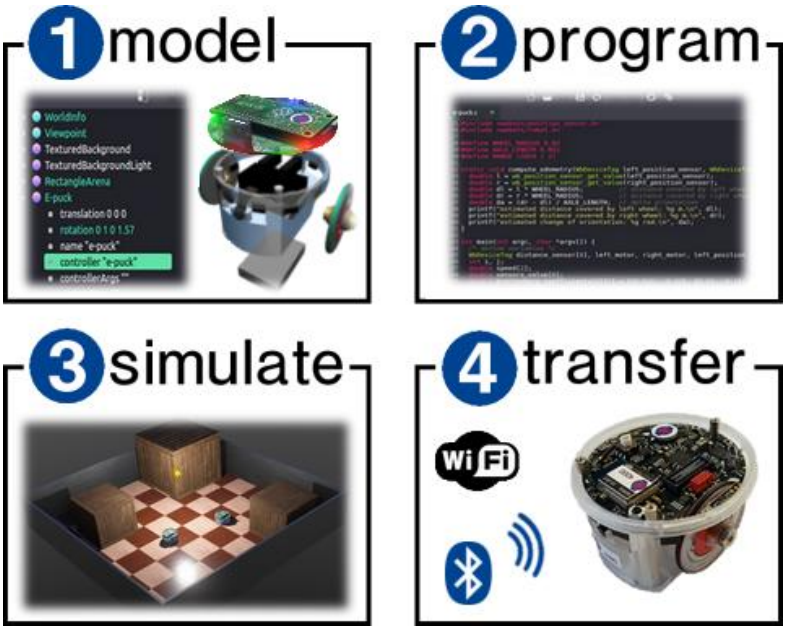
O ambiente de desenvolvimento do Webots é intuitivo e compreende várias ferramentas que auxiliam na criação e manipulação de simulações. A

interface gráfica do usuário (GUI) é bem estruturada, com uma janela de visualização 3D, uma árvore de cena que exibe todos os componentes do mundo simulado, uma paleta de ferramentas para adicionar e modificar objetos, e um painel de propriedades para ajustes detalhados dos parâmetros dos objetos.

Além disso, o Webots oferece suporte a múltiplas linguagens de programação, incluindo C, C++, Java, MATLAB, e Python, permitindo que os desenvolvedores utilizem a linguagem com a qual estão mais familiarizados. O uso de Python, em particular, é amplamente popular devido à sua simplicidade e à extensa biblioteca de módulos que facilita a implementação de algoritmos complexos de maneira eficiente.

Uma vantagem significativa do Webots é sua capacidade de realizar simulações em tempo real, possibilitando a visualização imediata dos resultados das alterações no código e nas configurações dos robôs. Isso é particularmente útil para o desenvolvimento iterativo, onde pequenos ajustes podem ser testados e refinados rapidamente. Além disso, o Webots permite a gravação e reprodução de simulações, o que é útil para a análise detalhada do comportamento do robô e para a apresentação de resultados em contextos acadêmicos e industriais. A Figura 2 mostra as principais etapas de um projeto no Webots.

Figura 2 - Etapas de criação de uma simulação



Fonte: CYBERBOTICS (2023)

2.2 VISÃO COMPUTACIONAL

O problema-alvo da visão computacional é calcular as propriedades do mundo tridimensional a partir de uma ou mais imagens digitais. As propriedades são principalmente geométricas (por exemplo, forma e posição de objetos sólidos) e dinâmicas (por exemplo, velocidades de objetos) (Trucco; Verri, 1998). O termo visão computacional pode ser confundido com processamento de imagens, mas existem diferenças entre os campos, pois o processamento de imagens diz respeito às propriedades da imagem e transformações imagem-imagem, enquanto o principal alvo da visão computacional é o mundo 3-D (Trucco; Verri, 1998).

A maioria dos algoritmos de visão computacional requer algum processamento preliminar de imagem, o que torna a sobreposição entre as duas disciplinas significativa. Exemplos de processamento de imagem incluem aprimoramento (computação de uma imagem de melhor qualidade do que a original), compressão (elaboração de representações compactas para imagens digitais, normalmente para fins de transmissão), restauração (eliminando o efeito de degradações conhecidas) e extração de recursos (localização de elementos de imagem especiais, como contornos ou áreas texturizadas) (Trucco; Verri, 1998).

2.2.1 Imagens

Uma imagem pode ser definida como uma função bidimensional, *f(x, y)*, onde *x* e *y* são coordenadas espaciais (plano), e a amplitude de *f* em qualquer par de coordenadas *(x, y)* é chamada a intensidade da imagem nesse ponto (Russel, 2010). O termo nível de cinza é frequentemente utilizado para se referir à intensidade das imagens monocromáticas. As imagens a cores são formadas por uma combinação de imagens individuais em 2-D. Por exemplo, no sistema de cores RGB, uma imagem colorida consiste em três imagens componentes individuais (vermelho, verde e azul). A Figura 3 mostra a representação do modelo matemático de imagens digitais (Queiroz; Gomes, 2001).

Figura 3 - Modelo de uma imagem digital

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}$$

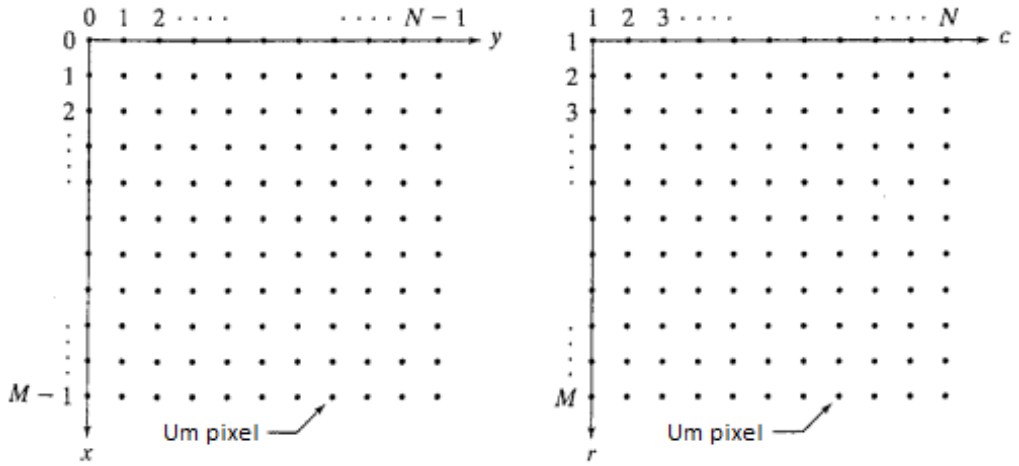
Fonte: GONZALEZ (2004)

Uma imagem pode ser contínua em relação às coordenadas *x* e *y*, e em amplitude. Para que uma imagem possa ser armazenada e/ou processada em um computador, torna-se necessária sua digitalização tanto em nível de coordenadas espaciais quanto de valores de brilho.

A conversão de tal imagem em forma digital requer que as coordenadas, bem como a amplitude, sejam digitalizadas. A digitalização dos valores das coordenadas chama-se amostragem; a digitalização dos valores de amplitude chama-se quantização. Assim, quando *x*, *y*, e os valores de amplitude de f são todas quantidades finitas e discretas, a imagem torna-se digital (Gonzalez; Woods; Eddins, 2004)

O resultado da amostragem e da quantização é uma matriz de números reais, conforme a Figura 4, com M linhas e N colunas, onde cada ponto representa um valor de *f(x, y).* Note-se que *x* varia de *0* a *M-1*, e *y* de *0* a *N-1*, em incrementos de números inteiros.

Figura 4 - Representação de uma imagem digital bidimensional



Fonte: GONZALEZ (2004)

2.3 OPENCV

O OpenCV (Open Source Computer Vision Library) é uma biblioteca de software de visão computacional e aprendizagem de máquina, de código aberto. Ele foi desenvolvido pela Intel em 1999 (Mordvintsev; Abid, 2017). O OpenCV foi concebido para fornecer uma infraestrutura comum para aplicações de visão computacional e para acelerar o uso da aprendizagem nos produtos comerciais.

Figura 5 - Logotipo OpenCV

Fonte: OPENCV (2024)

A biblioteca do OpenCV possui mais de 2500 algoritmos otimizados, que inclui um conjunto abrangente de algoritmos clássicos e de última geração de visão computacional e aprendizado de máquina (OPENCV, 2024). Esses algoritmos podem ser usados para detecção e reconhecimento de objetos, classificação de ações humanas em vídeos, rastrear o movimento de objetos, extrair modelos 3D a partir de imagens, juntar imagens para produzir uma imagem de alta resolução de uma cena inteira, encontrar imagens semelhantes em um banco de dados, calibração de câmeras etc. OpenCV tem mais de 47 mil pessoas da comunidade de usuários e número estimado de downloads superior a 18 milhões. A biblioteca é amplamente utilizada em empresas, grupos de pesquisa e por órgãos governamentais (OPENCV, 2024).

O OpenCV tem interface para C++, Python, Java e MATLAB, e suporta os sistemas operacionais Windows, Linux, Android e MAC OS. A biblioteca foi nativamente escrita em C++, e atualmente é distribuído por meio de uma licença BSD.

2.4 PYTHON

Python é uma linguagem de programação de propósito geral criada em 1991 por Guido Van Rossum, que se tornou muito popular nos últimos anos devido a sua simplicidade e legibilidade do código. Ela permite o programador expressar suas ideias em poucas linhas de código sem reduzir a legibilidade (Mordvintsev; Abid, 2017).

Figura 6 - Logotipo Python

Fonte: PYTHON (2024)

Dentre as aplicações e áreas que podem ser trabalhadas com Python tem-se: Aprendizagem de máquina, ciência de dados, desenvolvimento WEB, jogos, design gráfico, frameworks, prototipagem. Python possui licença de código aberto, e possui interface de comunicação com outras linguagens de programação como C/C++, Java, .NET, C# e Ruby (Beazley; Jones, 2013).

2.5 OCR (Reconhecimento Óptico de Caracteres)

O Reconhecimento Óptico de Caracteres (OCR) é uma tecnologia de processamento de imagens que permite a conversão de diferentes tipos de documentos, como arquivos digitalizados de papel, PDFs ou imagens capturadas por uma câmera digital, em dados editáveis e pesquisáveis. Essa tecnologia tem se tornado essencial em diversos setores, incluindo industrial, educacional, financeiro e governamental, onde a digitalização e automação de processos são críticas para a eficiência operacional (Biondich et al., 2002).

O OCR funciona através da análise de padrões de luz e sombra em uma imagem para identificar caracteres de texto. A tecnologia envolve as seguintes etapas:

a) **pré-processamento da Imagem**: esta etapa visa melhorar a qualidade da imagem para a análise subsequente. Envolve a correção de distorções, remoção de ruído, ajuste de contraste e brilho, e binarização, que é a conversão da imagem em preto e branco para facilitar a detecção dos caracteres (Nguyen et al., 2020);

b) **segmentação:** nesta fase, a imagem é dividida em suas partes constituintes, como linhas, palavras e caracteres individuais. A segmentação é crucial para isolar os caracteres de forma a possibilitar seu reconhecimento preciso (Laique et al., 2020);

c) **detecção e reconhecimento de caracteres:** utilizando algoritmos avançados, como redes neurais e machine learning, o OCR identifica os caracteres segmentados. Cada caractere é comparado com um conjunto de padrões previamente treinados, permitindo que o sistema reconheça e converta os caracteres em texto digital (Hom et al., 2022);

d) **pós-processamento**: após o reconhecimento dos caracteres, o texto resultante é verificado e corrigido, se necessário. Correções ortográficas e contextuais são aplicadas para garantir a precisão do texto convertido (Sarzynski et al., 2017).

No contexto industrial, o OCR é amplamente utilizado para a leitura e verificação de informações em produtos e embalagens, como números de série, datas de validade e códigos de barras. A automação desse processo elimina a necessidade de inspeção manual, reduzindo erros e aumentando a velocidade da produção (Biondich et al., 2002).

O OCR também é integrado em sistemas mais complexos para aplicações específicas, como o reconhecimento de placas de veículos, digitalização de documentos históricos, e sistemas de leitura automática de medidores (Cutter & Manduchi, 2017). A eficiência do OCR é potencializada quando combinada com outras tecnologias de visão computacional e inteligência

artificial, permitindo uma ampla gama de aplicações inovadoras (Laique et al., 2020).

2.6 PYTESSERACT

O Pytesseract é uma biblioteca Python que serve como um *wrapper* para o Tesseract-OCR, um dos motores de OCR mais poderosos e amplamente utilizados. Desenvolvido inicialmente pelo HP Labs e atualmente mantido pela Google, o Tesseract é uma ferramenta de código aberto que oferece reconhecimento de texto altamente preciso para diversas aplicações. A integração do Tesseract com Python através do Pytesseract facilita a implementação de funcionalidades de OCR em scripts e aplicações Python, tornando o processo de extração de texto de imagens muito mais acessível e eficiente.

A utilização do Pytesseract oferece várias vantagens para desenvolvedores que trabalham com processamento de imagem e visão computacional. Primeiramente, a biblioteca permite a conversão direta de imagens em texto, simplificando a integração com outras ferramentas e fluxos de trabalho em Python. Além disso, o Pytesseract suporta uma ampla variedade de formatos de imagem, incluindo JPEG, PNG, BMP, GIF e TIFF, o que o torna uma solução versátil para diferentes tipos de aplicações.

O Pytesseract não apenas permite a extração de texto simples, mas também oferece funcionalidades adicionais, como a extração de dados em formato de *string*, a obtenção de detalhes sobre a posição dos caracteres e a identificação de campos específicos dentro de uma imagem. Essas funcionalidades avançadas são particularmente úteis em aplicações que requerem uma análise mais detalhada dos dados de texto.

2.7 APRENDIZAGEM DE MÁQUINA

A aprendizagem de máquina, ou machine learning, é um ramo da inteligência artificial que se concentra no desenvolvimento de algoritmos e técnicas que permitem aos computadores aprender e tomar decisões com base em dados. Ao invés de serem programados explicitamente para realizar uma tarefa, os sistemas de aprendizagem de máquina são treinados em grandes conjuntos de dados, onde identificam padrões e criam modelos preditivos que podem ser utilizados para tomar decisões ou fazer previsões sobre novos dados (Biondich et al., 2002).

A aprendizagem de máquina pode ser dividida em três principais categorias:

a) **aprendizagem supervisionada**: nesta abordagem, o algoritmo é treinado com um conjunto de dados rotulados, ou seja, cada exemplo de treinamento é composto por uma entrada e a saída desejada. O objetivo é aprender uma função que mapeia as entradas para as saídas corretas. Exemplos comuns de técnicas de aprendizagem supervisionada incluem regressão linear, árvores de decisão, máquinas de vetores de suporte (SVM) e redes neurais. Aplicações típicas incluem classificação de e-mails como spam ou não-spam, reconhecimento de voz e previsão de valores numéricos (Chen et al., 2021).

b) **aprendizagem não supervisionada**: diferente da aprendizagem supervisionada, a aprendizagem não supervisionada trabalha com dados que não possuem rótulos ou saídas desejadas. O objetivo aqui é identificar estruturas ou padrões ocultos nos dados. As técnicas comuns incluem agrupamento (*clustering*) e análise de componentes principais (PCA). Aplicações incluem segmentação de clientes em marketing, detecção de anomalias e compressão de dados (Sample et al., 2004).

c) **aprendizagem por reforço**: esta abordagem envolve treinar um agente para tomar ações em um ambiente de forma a maximizar

alguma noção de recompensa cumulativa. O agente aprende a partir de suas próprias ações e das recompensas ou penalidades resultantes. Técnicas populares de aprendizagem por reforço incluem Q-learning e políticas de rede neural profunda (Deep Q-Networks). Aplicações incluem jogos de computador, controle robótico e sistemas de recomendação (Koert et al., 2020).

A aprendizagem de máquina tem sido fundamental para avanços em várias áreas, incluindo visão computacional, processamento de linguagem natural e sistemas autônomos. No contexto deste trabalho, técnicas de aprendizagem de máquina serão aplicadas para detectar a presença do logotipo da Suframa e do símbolo de lixeira nas baterias

2.8 TEACHABLE MACHINE

Teachable Machine é uma ferramenta desenvolvida pelo Google que permite a criação e o treinamento de modelos de aprendizado de máquina de maneira intuitiva e acessível. Projetada para ser usada por pessoas sem experiência prévia em programação ou ciência de dados, a Teachable Machine facilita o processo de treinar classificadores personalizados através de uma interface gráfica amigável (Kalshetty & Rakshit, 2021).

O principal objetivo do Teachable Machine é democratizar o acesso às tecnologias de aprendizado de máquina, permitindo que qualquer pessoa possa criar modelos para tarefas específicas como classificação de imagens, reconhecimento de poses ou sons. O processo de treinamento envolve três etapas simples: coleta de dados, treinamento do modelo e exportação para uso em projetos (Dwivedi et al., 2021).

a) **Coleta de Dados**: Nesta etapa, o usuário coleta exemplos das classes que deseja classificar. Por exemplo, no caso deste trabalho, imagens de baterias contendo o logotipo da Suframa e o símbolo de lixeira são capturadas. A ferramenta permite a coleta de dados diretamente através da webcam, upload de arquivos ou gravação de áudio (Forchhammer et al., 2022).

b) **Treinamento do Modelo**: Uma vez que os dados são coletados, o Teachable Machine usa esses exemplos para treinar um modelo de aprendizado de máquina. A ferramenta utiliza redes neurais profundas para aprender a distinguir entre as diferentes classes baseadas nas características dos dados fornecidos. Durante o treinamento, a ferramenta ajusta os pesos dos neurônios na rede neural para minimizar o erro de classificação (Dwivedi et al., 2023).

c) **Exportação e Implementação**: Após o treinamento, o modelo pode ser exportado para diferentes formatos, como TensorFlow.js, TensorFlow Lite ou arquivos de Keras, permitindo sua integração em diversos tipos de aplicações. No contexto deste projeto, o modelo treinado pode ser incorporado ao sistema de inspeção automatizada para detectar a presença do logotipo da Suframa e do símbolo de lixeira nas baterias (Forchhammer et al., 2022).

O uso do Teachable Machine no projeto oferece várias vantagens. Primeiramente, ele simplifica o processo de criação de um classificador personalizado sem a necessidade de conhecimentos avançados em programação ou machine learning. Além disso, a interface intuitiva acelera o desenvolvimento, permitindo a rápida iteração e ajuste do modelo com base no desempenho observado. Finalmente, a capacidade de exportar o modelo treinado em formatos compatíveis com diversas plataformas facilita sua integração em sistemas reais.

3 MÉTODOS E MATERIAIS

O objetivo deste capítulo é detalhar os procedimentos e ferramentas utilizados no desenvolvimento do projeto de simulação de um sistema OCR com Webots. Descreve-se o processo de instalação do software Webots no sistema operacional Windows, fornecendo uma visão geral da interface do programa e apresentando um guia passo a passo para a criação do mundo simulado, incluindo a configuração de uma esteira transportadora, o posicionamento da câmera e a integração de um robô UR5e. Este capítulo também explica como os componentes foram programados e integrados para realizar a inspeção de placas, oferecendo uma compreensão completa dos materiais e métodos aplicados no projeto.

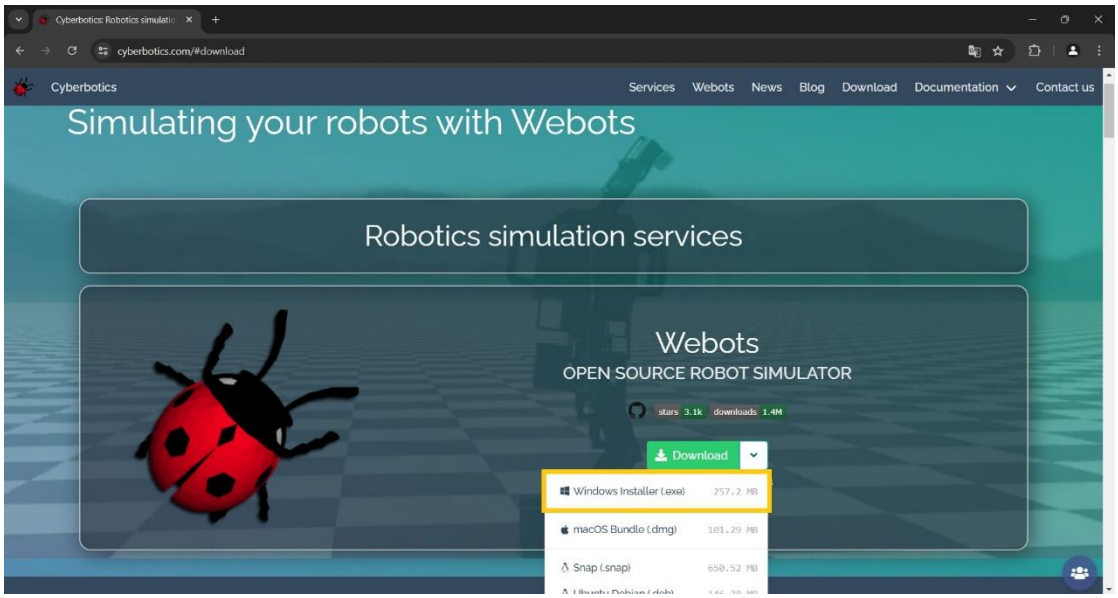
3.1 WEBOTS

3.1.1 Instalação do Webots

O Webots é um simulador de código aberto utilizado para modelar, programar e simular robôs em um ambiente virtual. Ele pode ser instalado em diversos sistemas operacionais, incluindo Windows, macOS e Linux, neste TCC ele foi instalado no sistema operacional Windows seguindo os passos descritos abaixo:

a) *download* do *Webots*: O software foi baixado diretamente do site oficial (Cyberbotics.com) e foi escolhida a versão compatível com o sistema operacional Windows. Na Figura 7, está destacado em amarelo o botão para download da versão para Windows do Webots;

Figura 7 - Tela de download do *Webots*



Fonte: O Autor (2024)

b) instalação: após o download, o arquivo de instalação foi executado e as instruções do assistente de instalação foram seguidas. Os termos de licença foram aceitos, o diretório de instalação desejado foi selecionado e o processo foi concluído.

3.1.2 Integração com o Pycharm

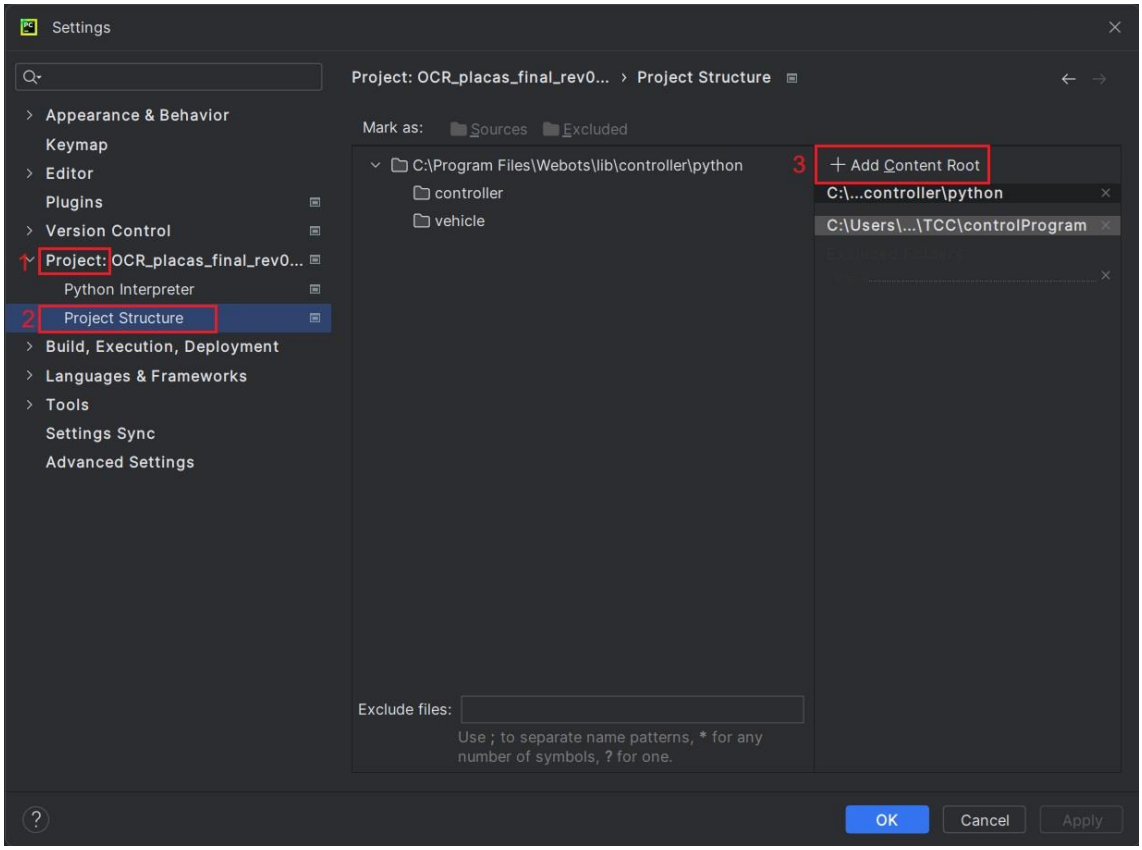
Usar um Ambiente de Desenvolvimento Integrado (IDE) é conveniente, pois ele geralmente possui ferramentas avançadas de edição de código-fonte, um depurador embutido, ferramentas de detecção de erros e controle de versão.

O Webots funciona com qualquer IDE para criar, construir e depurar controladores de robôs. Trata-se simplesmente de configurar a IDE corretamente para usar as regras de construção dos controladores do Webots. Para este TCC, foi utilizado o PyCharm, que é um ambiente de desenvolvimento integrado (IDE) multiplataforma, especificamente para a linguagem Python. Ele fornece análise de código, um depurador gráfico, ferramenta de teste unitário e integração com sistemas de controle de versão (VCSes). Para configurar um projeto no PyCharm para editar um controlador Python e executá-lo, é necessário adicionar a API do Webots para Python, e isso foi feito por meio dos seguintes passos:

a) abriu-se a janela de configurações do projeto por meio do atalho “CTRL + ALT + S”;

b) na janela *Settings*, selecionou-se a guia *Project* / *Project Structure* e, em seguida, o botão *Add Content Root* foi usado para adicionar uma nova pasta ao caminho. A Figura 8 mostra a sequência seguida para a realização dessa etapa;

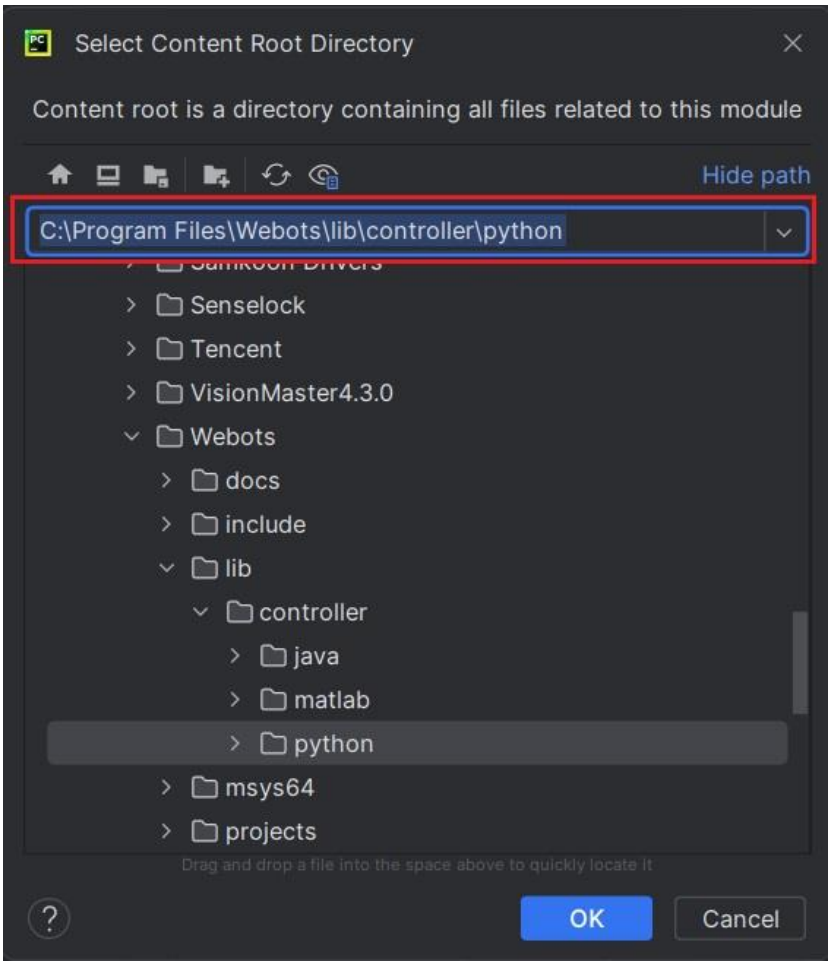
Figura 8 - Passos para a adição da API



Fonte: O Autor (2024)

c) a Figura 9 mostra a janela *Select Content Root Directory.* No campo destacado em vermelho foi necessário inserir o caminho para a pasta “/lib/controller/python”, que está dentro do diretório onde o *Webots* foi instalado no computador.

Figura 9 - Janela Select Content Root Directory

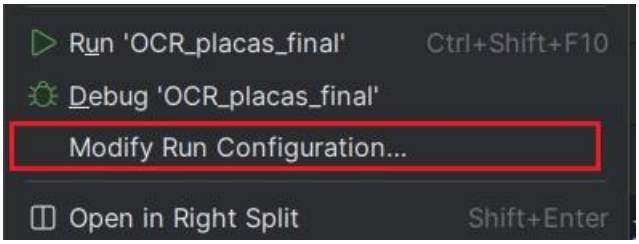


Fonte: O Autor (2024)

Depois de adicionar a API do Webots para Python o próximo passo foi criar a configuração de execução. Para isso seguiu-se as seguintes etapas:

a) Pressionou-se o botão direito do mouse sobre o nome do script em Python do controlador, na árvore do projeto no PyCharm, e então selecionou-se a opção Modify Run Configuration, que está destacada em vermelho na Figura 10;

Figura 10 - Opção Modify Run Configuration

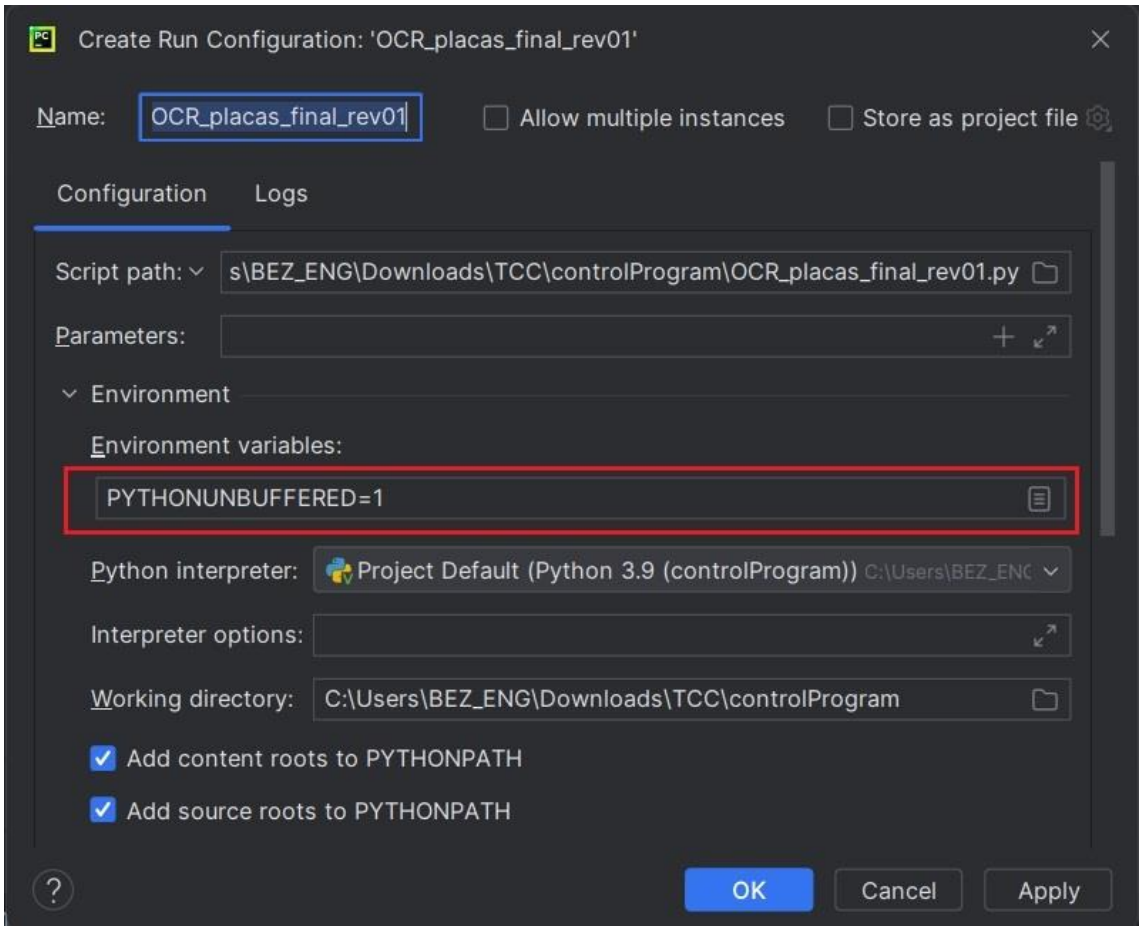


Fonte: O Autor (2024)

b) a Figura 11 mostra a janela aberta resultante do passo anterior. No retângulo destacado em vermelho, inseriram-se os seguintes caminhos:

- “PYTHONUNBUFFERED=1;”;
- “Path=C:\Program Files\Webots\lib\controller\\;”;
- “C:\Program Files\Webots\msys64\mingw64\bin\\;”;
- “C:\Program Files\Webots\msys64\mingw64\bin\cpp”.

Figura 11 - Adição de variáveis de ambiente



Fonte: O Autor (2024)

3.1.3 Construção do mundo

Um Mundo é um arquivo que contém informações como onde os objetos estão, como eles se parecem, como interagem entre si, qual é a cor do céu e as definições de gravidade, atrito, massas dos objetos. Ele define o estado inicial de uma simulação. Os diferentes objetos são chamados de Nós e são

organizados hierarquicamente em uma estrutura denominada Árvore de Cena. Portanto, um nó pode conter sub-nós. Um mundo é armazenado em um arquivo com a extensão ‘.wbt’. O formato do arquivo é derivado da linguagem VRML97 e é legível por humanos. Os arquivos do mundo devem ser armazenados diretamente em um diretório chamado worlds na pasta do projeto.

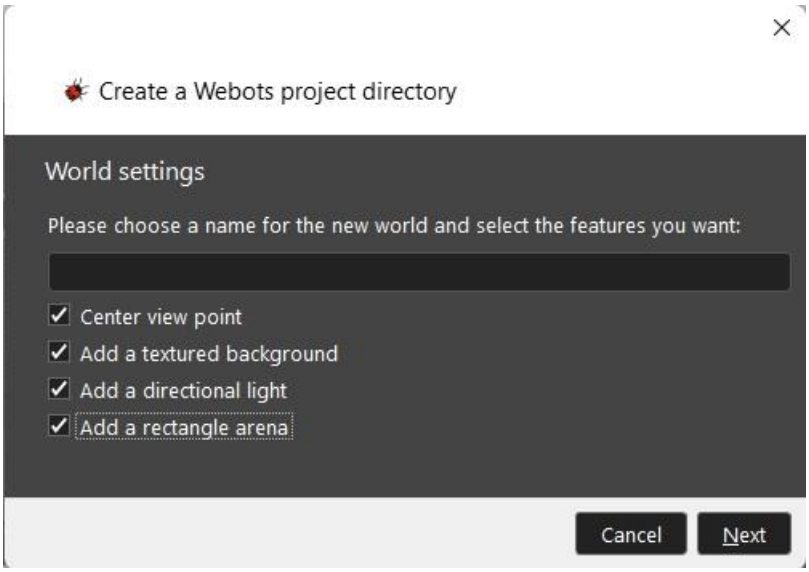
Para criar um novo projeto selecionou-se o botão “*File / New / New Project Directory*” e seguiu-se os seguintes passos:

a) nome do projeto: “TCC”;

b) nome do arquivo mundo: “inspecionaOCR.wbt”;

c) marcou se todos os campos exibidos na Figura 12 incluído o “*Add a rectangle arena*” que não é selecionado originalmente;

Figura 12 - Janela Create a Webots project directory

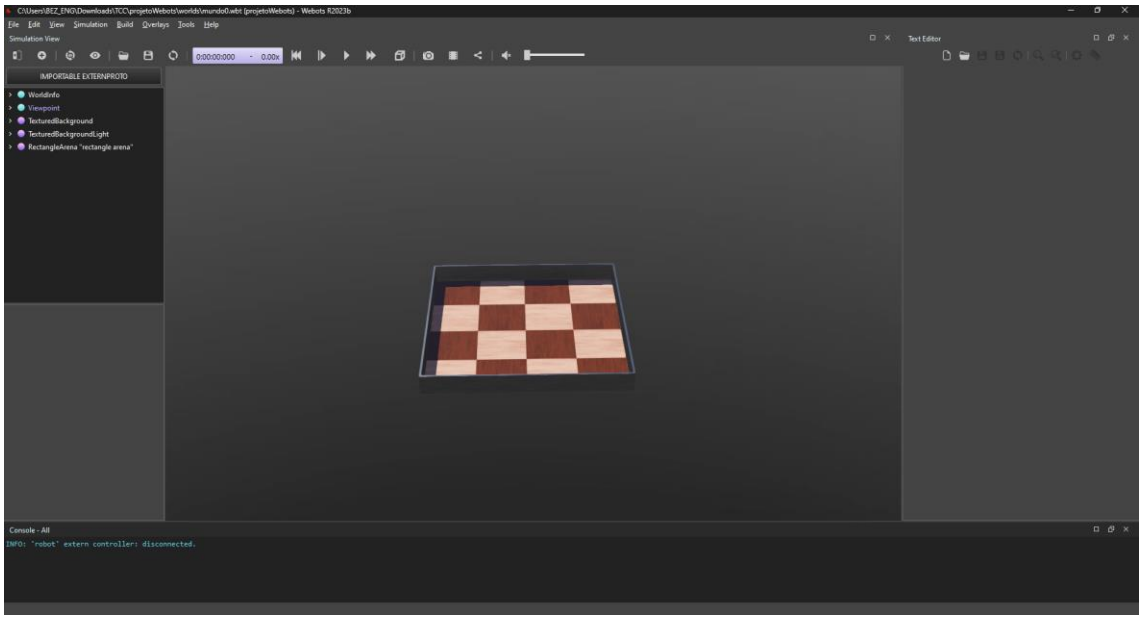


Fonte: O Autor (2024)

d) Webots exibe uma lista de diretórios e arquivos que foram criados. Eles correspondem à hierarquia padrão de arquivos de um projeto Webots. Ao Clicar no botão Concluir a janela de criação de projetos é fechada.

Ao criar o projeto a visualização 3D exibe uma arena quadrada com um piso quadriculado conforme a Figura 13. É possível mover o ponto de vista na visualização 3D usando o mouse: botão esquerdo, botão direito e a roda.

Figura 13 - Tela principal do mundo



Fonte: O Autor (2024)

A árvore de cena pode ser visualizada em duas sub-janelas da janela principal: a visualização 3D (no centro da janela principal) é a representação 3D da árvore de cena e a visualização da árvore de cena (à esquerda) é a representação hierárquica da árvore de cena. A visualização da árvore de cena é onde os nós e campos podem ser modificados. Inicialmente ela listou os seguintes nós (Figura 13):

a) *WorldInfo*: contém parâmetros globais da simulação;

b) *Viewpoint*: define os parâmetros da câmera do ponto de vista principal;

c) *TexturedBackground*: define o fundo da cena (é possível ver montanhas distantes se girar um pouco o ponto de vista);

d) *TexturedBackgroundLight*: define a luz associada ao fundo acima;

e) *RectangleArena*: define o único objeto visível na cena criada.

Cada nó tem algumas propriedades personalizáveis chamadas *Fields,* que são atributos que definem as propriedades e comportamentos dos nós no mundo simulado. Cada nó, que pode representar elementos como robôs, sensores, atuadores, objetos e ambientes, possui um conjunto de *Fields* que

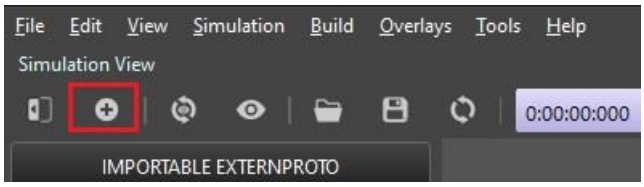
especificam suas características, como posição, rotação, cor, textura, entre outros. Os *Fields* podem ser de diferentes tipos, incluindo valores numéricos, vetores, *strings* e listas, permitindo uma grande flexibilidade na configuração e personalização dos elementos da simulação. Através da modificação dos *Fields*, os usuários podem ajustar dinamicamente os parâmetros dos nós durante a simulação, facilitando o desenvolvimento e a otimização de cenários robóticos complexos.

A primeira modificação feita na cena foi excluir o nó *RectangleArena* da árvore de cena, pois ele define um piso simples fixado no ambiente estático, sem propriedades físicas e cercado por paredes. Outros pisos pré-construídos estão disponíveis na biblioteca de objetos do *Webots*. Para excluir o nó *RectangleArena*, selecionou-se ele na árvore de cena e pressionou-se a tecla del do teclado.

Para adicionar um novo piso, seguiram-se os seguintes passos:

a) selecionou-se o último nó da árvore de cena, que, no caso, era o *TexturedBackgroundLight*, e pressionou-se o botão *Add*, que está destacado em vermelho na Figura 14;

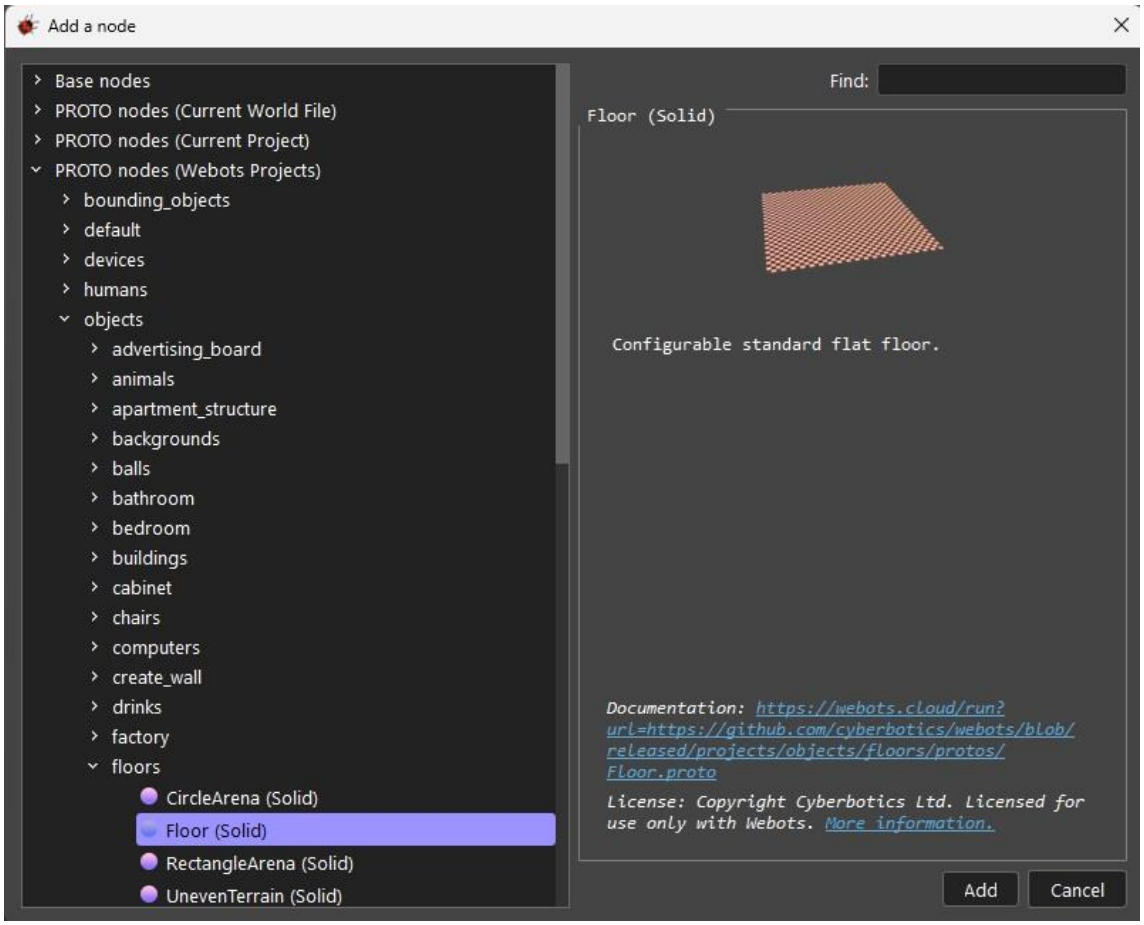
Figura 14 - Botão Add



Fonte: O Autor (2024)

b) na janela aberta, escolheu-se *PROTO nodes (Webots Projects) / objects / floors / Floor (Solid),* conforme mostra a Figura 15, e depois pressionou-se o botão *Add* no canto inferior direito da janela *Add a node* (Figura 15);

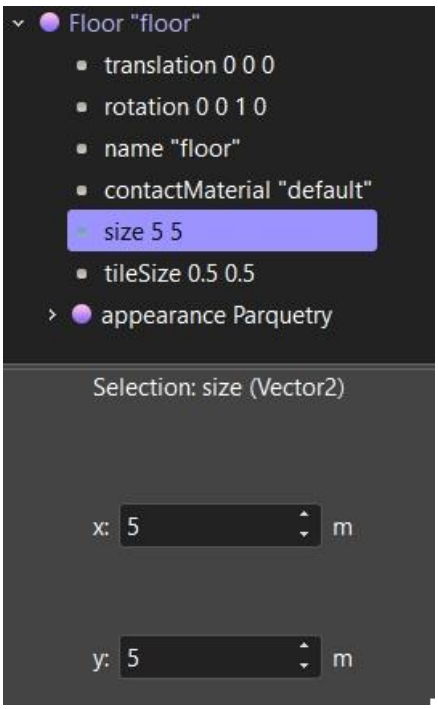
Figura 15 - Janela Add a node



Fonte: O Autor (2024)

c) o novo piso foi adicionado à cena, e os próximos passos foram alterar o seu tamanho e aparência. Para alterar o tamanho, deu-se um duplo clique com o botão esquerdo do mouse no nó *Floor* na árvore de cena para expandir os campos, e então selecionou-se o campo *size* e alterou-se o seu valor para (5, 5), conforme a Figura 16;

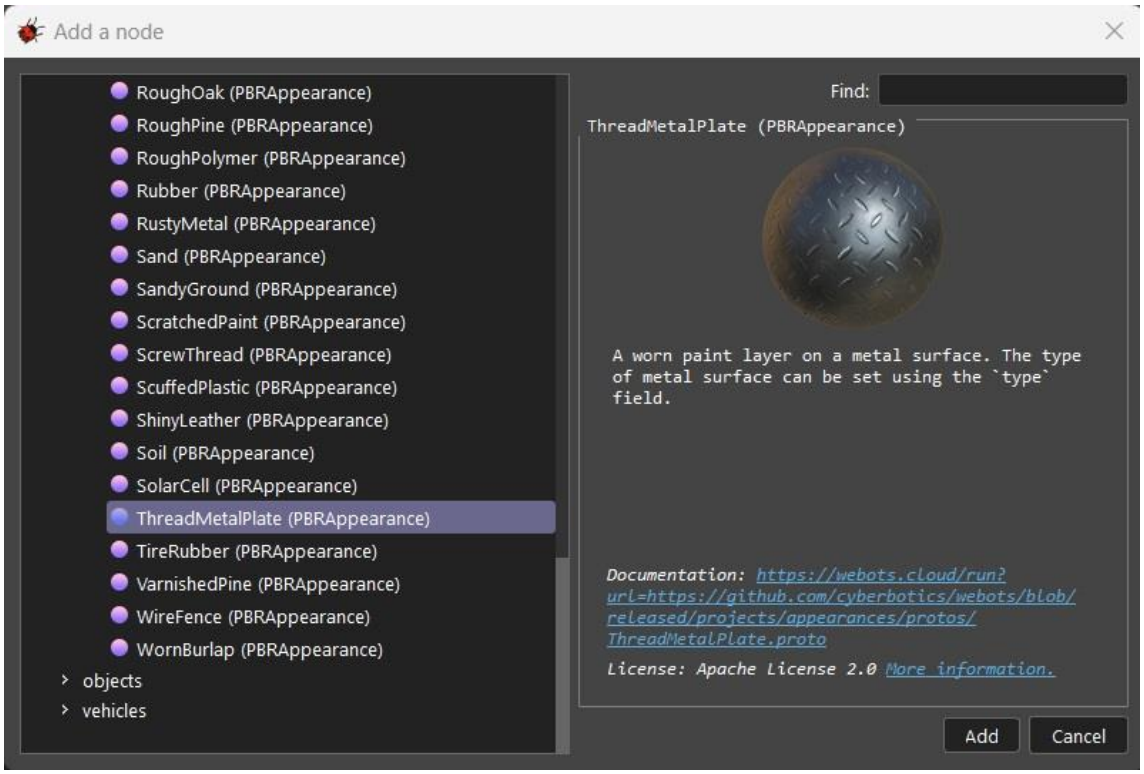
Figura 16 - Edição do tamanho do piso



Fonte: O Autor (2024)

d) para alterar a aparência, o primeiro passo foi excluir a aparência original. Para isso, selecionou-se o campo *appearance* e pressionou-se a tecla *del* do teclado. Depois, pressionou-se o botão *Add* para inserir o novo nó, e escolheu-se *PROTO nodes (Webots Projects)* / *appearances* / *ThreadMetalPlate*, conforme mostra a Figura 17. O resultado da configuração do piso pode ser visto na Figura 18.

Figura 17 - seleção da aparência do piso



Fonte: O Autor (2024)

Figura 18 - aparência final do piso

Fonte: O Autor (2024)

Os passos mostrados para configurar o piso foram aplicados para configurar os demais nós da simulação. Os nós também podem ser modificados por meio da edição do arquivo ".wbt", que é armazenado na pasta worlds do projeto.

3.1.4 Instanciação de um PROTO

No *Webots*, um PROTO é um arquivo que define um protótipo de nó, permitindo a criação de componentes personalizados reutilizáveis na simulação. Esses arquivos são utilizados para encapsular a complexidade de um objeto, robô ou ambiente, fornecendo uma interface simplificada e parametrizável para sua instanciação e manipulação. Um PROTO pode incluir definições de geometria, materiais, sensores, atuadores e scripts de controle, além de permitir a definição de parâmetros ajustáveis que facilitam a configuração e a reutilização do componente em diferentes cenários. Isso torna o desenvolvimento e a manutenção de simulações mais eficientes, promovendo a modularidade e a consistência no design de ambientes robóticos complexos. A Figura 19 mostra o modelo de instância de um PROTO, e essas instâncias podem ser adicionadas ao arquivo ".wbt" da simulação.

Figura 19 - Modelo de instância de um PROTO

```
protoName {
        fieldName1  value1
        fieldName2  value2
        fieldName3  value3
        ...
        ...
        ...
        fieldNameN   valueN
}
```

Fonte: O Autor (2024)

3.1.5 Configuração do nó *TexturedBackground*

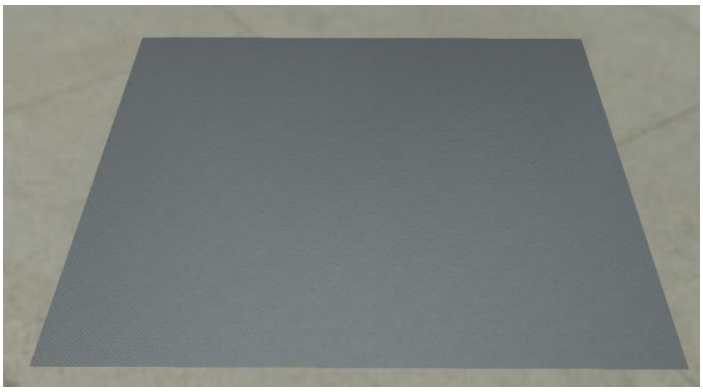
Nesse nó são definidas a aparência de fundo da cena, e para esse projeto foi definida uma aparência de fábrica. A Figura 20 mostra o código de instância desse nó, e a Figura 21 mostra o resultado da configuração desse nó no mundo criado.

Figura 20 - Instância do nó *TexturedBackground*

```
TexturedBackground {
  texture "factory"
}
```

Fonte: O Autor (2024)

Figura 21 - resultado após o ajuste em TexturedBackground



Fonte: O Autor (2024)

3.1.6 Adição das paredes

A Figura 22 mostra a instância das quatro paredes que foram definidas na simulação, e a Figura 23 mostra o resultado da adição das paredes ao mundo.

Figura 22 - Instância das paredes do mundo

```
DEF WALL1 Pose {
  translation 0 2.5 1
  rotation 1 0 0 1.5708
  children [
    DEF wall_long Shape {
      appearance Roughcast {
      }
      geometry Plane {
        size 5 2
      }
    }
  ]
}
DEF WALL2 Pose {
  translation 0 -2.5 1
  rotation 1 0 0 -1.5708
  children [
    USE wall_long
  ]
}
DEF WALL3 Pose {
  translation 2.5 0 1
  rotation -0.5773509358554485 0.5773489358556708 0.5773509358554485 -2.094395307179586
  children [
    DEF wall_short Shape {
      appearance Roughcast {
      }
      geometry Plane {
        size 5 2
      }
    }
  ]
}
DEF WALL4 Pose {
  translation -2.5 0 1
  rotation 0.5773506025225371 0.5773496025232256 0.5773506025225371 2.0944
  children [
    USE wall_short
  ]
}
```

Fonte: O Autor (2024)

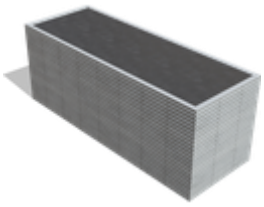
Figura 23 - resultado da adição das paredes

Fonte: O Autor (2024)

3.1.7 Adição da esteira de entrada

A Figura 24 mostra a imagem da esteira utilizada para a entrada das baterias a serem inspecionadas, e a Figura 25 mostra a instância da esteira. Os únicos campos ajustados foram o tamanho e a velocidade.

Figura 24 - Esteira de entrada



Fonte: WEBOTS (2024)

Figura 25 - instancia da esteira de entrada

```
ConveyorBelt {
  size 1.4 0.4 0.6
  speed -0.2
}
```

Fonte: O Autor (2024)

3.1.8 Adição das esteiras de produtos aprovados e reprovados

Para as esteiras de produtos aprovados e reprovados, foi utilizado o mesmo modelo mostrado na Figura 24. A Figura 26 mostra a instância da esteira de aprovados, e a Figura 27 mostra a instância da esteira de reprovados. Os campos modificados foram a posição, orientação, nome, tamanho e velocidade.

Figura 26 - instancia da esteira de aprovados

```
DEF esteiraOK ConveyorBelt {
  translation -1.33 0.85 0
  rotation 0 0 1 1.57
  name "esteiraAprovado"
  size 1 0.4 0.6
  speed 0
}
```

Fonte: O Autor (2024)

Figura 27 - instância da esteira de reprovados

```
DEF esteiraNG ConveyorBelt {
  translation -1.33 -0.85 0
  rotation 0 0 1 1.57
  name "esteiraNG"
  size 1 0.4 0.6
  speed 0
}
```

Fonte: O Autor (2024)

3.1.9 Adição da base do robô colaborativo

A Figura 28 mostra a imagem da base utilizada para apoiar o robô colaborativo responsável por segregar as baterias. Essa base foi modelada a partir da adaptação de um nó mesa, e a Figura 29 mostra a instância da base do robô. Os campos ajustados foram a posição, nome, tamanho geral e dos pés, e a aparência.

Figura 28 - Base de apoio do robô colaborativo

Fonte: O Autor (2024)

Figura 29 - instância da base de apoio do robô

```
Table {
  translation -1.2 -0.03 0
  name "table(1)"
  size 0.4 0.4 0.6
  feetSize 0.05 0.05
  trayAppearance BrushedAluminium {
    colorOverride 0.333 0.341 0.325
  }
  legAppearance BrushedAluminium {
    colorOverride 0.333 0.341 0.325
  }
}
```

Fonte: O Autor (2024)

3.1.10 Adição do robô colaborativo

No desenvolvimento deste projeto, foi utilizado o robô colaborativo UR5e da Universal Robots, uma solução avançada e versátil projetada para trabalhar em estreita colaboração com humanos em ambientes industriais. O UR5e é um braço robótico de seis graus de liberdade, conhecido por sua precisão, repetibilidade e flexibilidade, tornando-o ideal para uma ampla gama de aplicações, incluindo montagem, soldagem, pintura e manipulação de materiais. Equipado com sensores de torque em cada junta, o UR5e pode detectar e responder a forças externas, garantindo uma operação segura ao lado de operadores humanos. No contexto deste trabalho, o UR5e foi empregado para a tarefa de separar placas aprovadas e reprovadas na linha de inspeção automatizada, demonstrando sua capacidade de integração eficiente com sistemas de visão computacional e OCR para garantir a qualidade e a consistência dos produtos manufaturados. A Figura 30 mostra o modelo 3D do robô UR5e, e a Figura 31 mostra o código da instância. Os campos ajustados foram posição, controlador, supervisor e ferramentas.

Figura 30 - Robô colaborativo UR5e

Fonte: WEBOTS (2024)

Figura 31 - instância do robô colaborativo UR5e

```
DEF roboUR5e UR5e {
  translation -1.2 -0.13 0.6
  controller "<extern>"
  supervisor TRUE
  toolSlot [
    DEF VACUM Pose {
      rotation 0 1 0 3.14
    }
    suporteCam_UR5e {
      translation -0.05 -0.052 0
      rotation 1 0 0 -1.57
    }
    Camera {
      translation -0.095 0.077 0
      rotation 0 0 1 1.57
      width 1600
      height 1600
    }
    Display {
      width 1600
      height 1600
    }
  ]
}
```

Fonte: O Autor (2024)

3.1.11 Adição do robô supervisor

No Webots, o controlador Supervisor é uma funcionalidade que permite a supervisão e o controle de alto nível do ambiente de simulação. Diferente dos controladores normais, que são responsáveis por controlar o comportamento de robôs individuais, o controlador Supervisor tem capacidades ampliadas para monitorar e manipular vários aspectos da simulação globalmente. Ele pode acessar e modificar propriedades de qualquer nó na árvore de cena, como adicionar, remover ou mover objetos, além de ajustar parâmetros dinâmicos durante a execução da simulação.

Uma das principais vantagens do controlador Supervisor é a sua capacidade de interagir com o ambiente de maneira abrangente. Por exemplo, ele pode ser usado para reiniciar a simulação, alterar condições ambientais como iluminação e terreno, e até mesmo monitorar o desempenho e o estado de

múltiplos robôs simultaneamente. Isso é especialmente útil para simulações complexas onde a coordenação entre múltiplos agentes ou a adaptação a mudanças no cenário é necessária.

Além disso, o controlador Supervisor pode se comunicar com controladores de robôs individuais, permitindo uma arquitetura de controle hierárquica. Isso possibilita a implementação de estratégias avançadas de controle e coordenação, onde o Supervisor toma decisões de alto nível e delega tarefas específicas aos robôs. O Supervisor também pode acessar informações detalhadas de sensores e atuar conforme necessário para simular eventos complexos ou emergências. O desenvolvimento de um controlador Supervisor é realizado utilizando linguagens de programação como Python, C, C++, Java ou MATLAB, aproveitando as bibliotecas e APIs fornecidas pelo Webots.

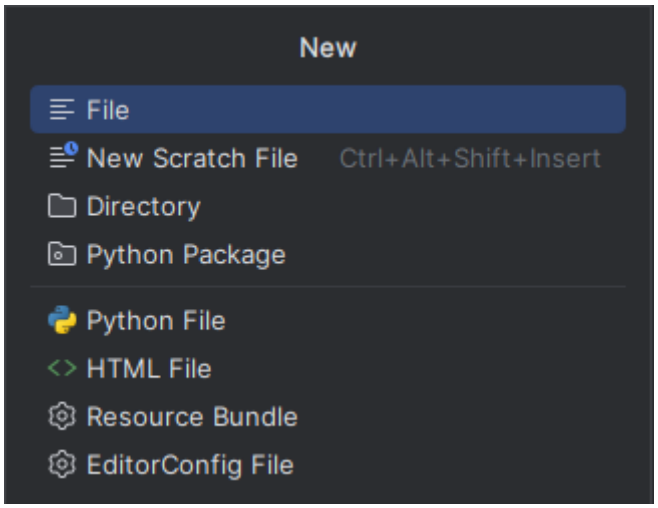
Para um transformar um robô em supervisor basta definir o seu campo *supervisor* como *TRUE.* Para esse projeto o robô supervisor será responsável por adicionar e excluir as baterias da simulação, capturar as imagens da câmera, processar o sinal dos sensores de detecção das células e controlar a velocidade das esteiras de produtos aprovados e reprovados. A figura mostra a instância do robô supervisor utilizado nesse projeto.

3.2 SCRIPT PARA INSPEÇÃO DAS PLACAS

O script de inspeção das placas define a função responsável por inspecionar as imagens para verificar se os textos estão presentes nelas. Essa função recebe como parâmetros a imagem capturada pela câmera e uma lista de palavras a serem procuradas na imagem. A função retorna a imagem com o desenho de um retângulo ao redor das palavras encontradas e o resultado da inspeção no canto superior esquerdo, além de uma variável booleana que indica se a placa foi aprovada ou reprovada.

O primeiro passo é criar o arquivo do programa, e para isso na tela do Pycharm deve-se pressionar o atalho *Alt* + *Insert* para abrir a janela New (Figura 32), e então selecionar a opção ‘*Python File’.*

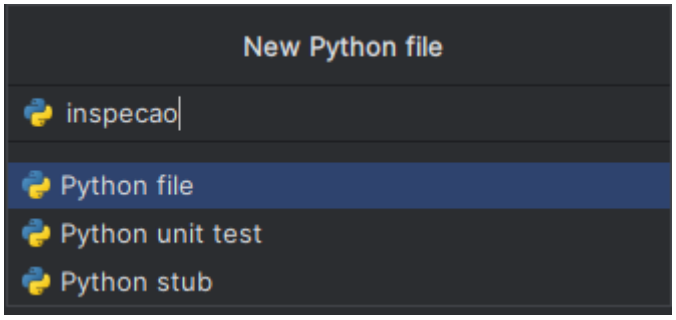
Figura 32 - Janela New



Fonte: O Autor (2024)

Ao clicar em '*Python File'* é exibido a janela *'New Python File*' mostrada na Figura 33, onde deve-se inserir o nome do arquivo, que nesta etapa foi chamado de "inspecao'. Ao pressionar *Enter* o arquivo é criado.

Figura 33 - Janela New Python File



Fonte: O Autor (2024)

3.2.1 Bibliotecas

A Figura 34 mostra as bibliotecas importadas para o script inspecao.py

Figura 34 - Bibliotecas importadas

```
1  import cv2
2  import pytesseract
3  from pytesseract import Output
4  import numpy as np
```

Fonte: O Autor (2024)

3.2.1.1 CV2

OpenCV (Open Source Computer Vision Library) é uma biblioteca de software de código aberto destinada a aplicações de visão computacional e aprendizado de máquina. *cv2* é a interface Python do OpenCV. A biblioteca é altamente eficiente e contém mais de 2.500 algoritmos otimizados para diversas tarefas, como detecção e reconhecimento de rostos, identificação de objetos, classificação de ações humanas em vídeos, rastreamento de movimentos, extração de modelos 3D, produção de mapas de profundidade estereoscópicos, realidade aumentada, processamento de imagem digital, e muito mais. Para instalar o OpenCV deve-se executar no terminal o comando mostrado na Figura 35.

Figura 35 - Comando de instalação do OpenCV

```
pip install opencv-python
```

Fonte: O Autor (2024)

3.2.1.2 Pytesseract

Pytesseract é uma biblioteca Python que serve como um invólucro para o *Tesseract OCR Engine*, um dos mecanismos de OCR mais precisos disponíveis. Ele permite que desenvolvedores extraiam texto de imagens de forma programática. Pytesseract pode processar uma variedade de formatos de imagem e converter o texto contido nelas em *strings* editáveis, permitindo a análise e manipulação de dados textuais provenientes de imagens. É amplamente utilizado em aplicações que requerem a digitalização de documentos, extração de texto de imagens e reconhecimento de caracteres em várias linguagens (Pytesseract, 2024).

Para utilizar o *Pytesseract*, é necessário que o *Tesseract*-OCR esteja instalado no sistema. O processo de instalação envolve o download e a configuração do *Tesseract*, seguido da instalação da biblioteca *Pytesseract* via pip. O comando para instalar o *Pytesseract* é exibido na Figura 36.

Figura 36 - Comando de instalação do Pytesseract

```
pip install pytesseract
```

Fonte: O Autor (2024)

3.2.1.3 Pytesseract.output

Pytesseract.Output é uma classe dentro da biblioteca pytesseract que facilita a especificação do formato de saída dos resultados do OCR. Ela permite que os usuários escolham entre diferentes tipos de estrutura de dados para os resultados do OCR, como dicionário, *string* ou *bytes*. O formato mais utilizado é *Output.DICT*, que retorna um dicionário detalhado contendo informações como o texto reconhecido, as coordenadas da caixa delimitadora de cada palavra, e a confiança de reconhecimento para cada item. Essa granularidade de dados é especialmente útil para análises detalhadas e manipulações avançadas do texto extraído de imagens.

3.2.1.4 Numpy

A biblioteca NumPy é uma poderosa ferramenta de computação científica em Python que fornece suporte para arrays e matrizes de grandes dimensões, junto com uma coleção de funções matemáticas para operar sobre esses arrays de maneira eficiente (Mordvintsev; ABID, 2017). NumPy permite a execução de operações de álgebra linear, transformadas de Fourier, geração de números aleatórios e muitas outras operações matemáticas de forma rápida e eficiente. Sua capacidade de manipular dados numéricos torna-a essencial para tarefas de análise de dados, aprendizado de máquina e processamento de imagens, proporcionando uma base sólida para bibliotecas mais avançadas como SciPy, TensorFlow e OpenCV.

3.2.2 Função verifica_imagem()

Após a importação das bibliotecas foi definida a função verifica_imagem (mostrada na Figura 37), que aceita três parâmetros:

a) imagem: a imagem que será processada;

b) strings_de_busca: uma lista de strings que serão verificadas quanto à presença na imagem.

c) model: variável com os pesos do modelo de classificação treinado.

Figura 37 - Cabeçalho da função verifica_imagem

```
def verifica_imagem(imagem, strings_de_busca, model):
```

Fonte: O Autor (2024)

Os próximos passos foram inicializar a variável strings_encontradas e realizar o reconhecimento na imagem com a função image_to_data. Depois, a função Classificador foi chamada para realizar a predição da imagem com o modelo de classificação treinado. A Figura 38 mostra o código para realizar esses dois passos.

Figura 38 - inicialização e OCR na imagem

```
strings_encontradas = set()
resultados = pytesseract.image_to_data(imagem, output_type=Output.DICT, lang='por')
res_classificador = Classificador(imagem, model)
```

Fonte: O Autor (2024)

A função image_to_data foi configurada para realizar o reconhecimento no idioma português e para retornar o resultado em formato de dicionário. Os campos presentes no dicionário de saída são os seguintes:

a. *block_num*: Número do bloco atual. Quando o *tesseract* faz o OCR, ele divide a imagem em várias regiões, o que pode variar de acordo com os parâmetros do modo de segmentação de página e também outros critérios próprios do algoritmo. Cada bloco é uma região;

b. conf: confiança da predição (de 0 a 100. -1 significa que não foi reconhecido texto);

c. *height*: altura do bloco de texto detectada (ou seja, da caixa delimitadora);

d. *left*: coordenada x onde inicia a caixa delimitadora;

e. *level*: corresponde à categoria do bloco detectado. são 5 valores possíveis: página, bloco, parágrafo, linha e palavra.;

f. *line_num*: número da linha do que foi detectado (inicia com 0);

g. *page_num*: o índice da página onde o item foi detectado;

h. *text*: o resultado do reconhecimento;

i. *top*: coordenada y onde a caixa delimitadora começa;

j. *width*: largura do bloco de texto atual detectado;

k. word_*num*: número da palavra (índice) dentro do bloco atual.

O trecho de código mostrado na Figura 39 contém o loop que fará o processamento dos textos detectados. Um texto só será considerado válido se ele tiver uma confiança maior que 16 e for diferente de um espaço em branco. Caso o texto faça parte da lista string_de_busca ele será adicionado à variável strings_encontradas e será desenhado um retângulo em sua volta.

Figura 39 - Loop de processamento dos textos

```
for i in range(0, len(resultados['text'])):
    if int(resultados['conf'][i]) > 16 and resultados['text'][i] != '':
        texto = resultados['text'][i].strip()
        if texto in strings_de_busca:
            strings_encontradas.add(texto)
            (x, y, w, h) = (resultados['left'][i], resultados['top'][i],
                            resultados['width'][i], resultados['height'][i])
            cv2.rectangle(imagem, (x, y), (x + w, y + h), (255, 0, 0), 2)
```

Fonte: O Autor (2024)

No trecho de código mostrado na Figura 40, são desenhados retângulos ao redor dos objetos encontrados pelo classificador. Em seguida, o conteúdo de strings_encontradas é comparado com strings_de_busca. Caso sejam iguais,

isso significa que a bateria foi aprovada pelo OCR. Se a imagem for aprovada tanto pelo classificador quanto pelo OCR, a variável resultado_inspecao recebe o valor True e a mensagem ‘OK’ é escrita na imagem. Caso contrário, a bateria será reprovada, e a mensagem ‘NG’ será escrita na imagem.

Figura 40 - Verificação do resultado final da inspeção

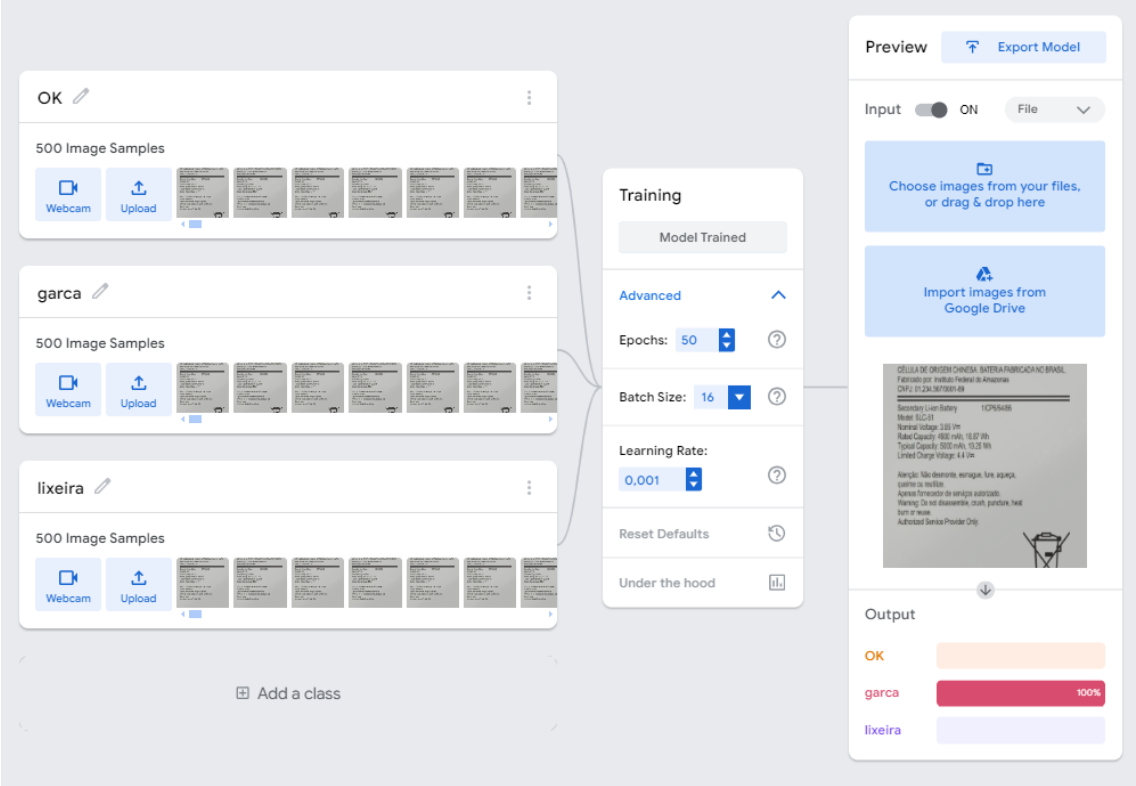
```
l = [(719, 1019, 882, 1180), (893, 942, 1052, 1180)]
for i in range(2):
    if (i + 1) != res_classificador:
        cv2.rectangle(imagem, l[i][0:2], l[i][2:4], (255, 0, 0), 2)
resultado_inspecao_ocr = False
# Verifica se todas as strings foram encontradas
if all(string in strings_encontradas for string in strings_de_busca):
    resultado_inspecao_ocr = True

resultado_inspecao = resultado_inspecao_ocr and (res_classificador == 0)
if resultado_inspecao:
    cv2.putText(imagem, 'OK', (25, 195), cv2.FONT_HERSHEY_SIMPLEX, 8,
                (0, 255, 0), 3, cv2.LINE_AA)
else:
    cv2.putText(imagem, 'NG', (25, 195), cv2.FONT_HERSHEY_SIMPLEX, 8,
                (0, 0, 255), 3, cv2.LINE_AA)

imagem = cv2.cvtColor(imagem, cv2.COLOR_BGR2BGRA)

return imagem, resultado_inspecao
```

Fonte: O Autor (2024)

3.3 TREINAMENTO DO MODELO DE CLASSIFICAÇÃO

Foi treinado um modelo de classificação para detectar a presença do selo do PIM e o símbolo da lixeira, e para essa tarefa foi utilizado a ferramenta Teachable Machine. A Figura 41 mostra uma visão geral da plataforma. As imagens de treinamento foram divididas em três classes com 500 imagens cada. A primeira classe foi das imagens aprovadas, a segunda classe foi das imagens sem o selo do PIM e a terceira classe foi das imagens sem o símbolo da lixeira.

Figura 41 - Teachable Machine - Visão Geral



Fonte: O Autor (2024)

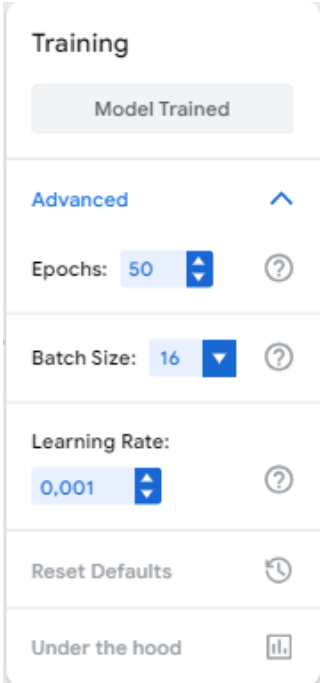
Para o treinamento do modelo de classificação foram configurados parâmetros específicos de treinamento, conforme mostrado Figura 42. Os parâmetros definidos foram:

a) **Epochs (50):** Este parâmetro indica o número de vezes que o algoritmo de treinamento irá passar por todo o conjunto de dados de treinamento. Neste caso, o modelo foi treinado por 50 épocas, o que significa que ele analisou todas as 1500 imagens (500 por classe) 50 vezes para ajustar os pesos dos neurônios e melhorar a precisão da classificação.

b) **Batch Size (16):** Este parâmetro define o número de amostras de treinamento que o modelo processa antes de atualizar os parâmetros internos. Com um *batch size* de 16, o modelo ajusta seus pesos após processar cada conjunto de 16 imagens. Esse ajuste em pequenos lotes pode ajudar a encontrar um ponto de equilíbrio entre a estabilidade e a eficiência do treinamento.

c) **Learning Rate (0.001):** A taxa de aprendizado controla o tamanho dos passos que o algoritmo dá na direção dos mínimos da função de custo. Um valor de 0.001 foi escolhido, indicando passos relativamente pequenos, o que pode ajudar a garantir que o modelo converja de maneira estável para uma solução ótima, evitando grandes oscilações que poderiam ocorrer com taxas de aprendizado maiores.

Figura 42 - parâmetros de treinamento



Fonte: O Autor (2024)

3.4 FUNÇÃO DE CLASSIFICAÇÃO

O trecho de código mostrado na Figura 43 é responsável por realizar as predições usando um modelo de classificação treinado no *Teachable Machine* para verificar a presença do selo do PIM e do símbolo da lixeira nas imagens das baterias. A função Classificador recebe uma imagem (img) e o modelo (*model*) como parâmetros. Primeiramente, extrai a região de interesse da imagem, delimitada pelas coordenadas [320:1200, 470:1100]. Em seguida, a imagem é redimensionada para 224x224 *pixels*, que é a dimensão esperada pela entrada

do modelo. A imagem redimensionada é convertida em um *array NumPy* e remodelada para se adequar ao formato de entrada do modelo (1, 224, 224, 3). A imagem é normalizada, ajustando seus valores para a faixa entre -1 e 1, facilitando o processamento pelo modelo. O modelo então realiza a predição sobre a imagem normalizada, e o índice da classe com a maior probabilidade é retornado pela função, indicando a presença ou ausência dos selos específicos.

Figura 43 - função classificador

```python
def Classificador(img, model):
    # Extrai a regiao de interesse da imagem
    image = img[320:1200, 470:1100]
    # Redimensiona a imagem bruta em pixels (224 de altura e 224 de largura)
    image = cv2.resize(image, dsize: (224, 224), interpolation=cv2.INTER_AREA)
    # Transforme a imagem em um array numpy e remodele-a de
    # acordo com o formato de entrada do modelo.
    image = np.asarray(image, dtype=np.float32).reshape(1, 224, 224, 3)

    # Normalize a matriz de imagens
    image = (image / 127.5) - 1
    # Prevê com o modelo
    prediction = model.predict(image)
    index = np.argmax(prediction)

    return index
```

Fonte: O Autor (2024)

3.5 CRIAÇÃO DO CONTROLADOR DO ROBÔ

Um controlador é um programa que define o comportamento de um robô. Os controladores no Webots podem ser escritos nas seguintes linguagens de programação: C, C++, Java, Python, MATLAB e ROS. Controladores escritos em C, C++ e Java necessitam ser compilados antes de poderem ser utilizados como controladores de robôs. Nesse projeto, o controlador será responsável por realizar o OCR na imagem das baterias e, então, segregá-las com base no resultado.

3.5.1 Bibliotecas

A Figura 44 mostra as bibliotecas importadas para o programa do controlador.

Figura 44 - Bibliotecas do controlador

```
from controller import Supervisor,  Display
from manipula_rotacao import *
import random
from inspecao import *
import io
from keras.models import load_model
```

Fonte: O Autor (2024)

3.5.1.1 Supervisor

Esta biblioteca é utilizada para criar controladores supervisores, que têm capacidades ampliadas em comparação aos controladores normais. Com a biblioteca Supervisor, é possível acessar, monitorar e modificar praticamente qualquer aspecto do ambiente de simulação, proporcionando um nível elevado de controle e flexibilidade.

3.5.1.2 Display

Esta biblioteca contém funções para configuração e exibição de uma imagem por meio do nó Display.

3.5.1.3 Manipula_rotacao

Esta biblioteca contém funções para manipulação dos valores de rotação de um nó, como, por exemplo, a conversão de uma matriz de rotação para a representação eixo-ângulo, conversão de eixo-ângulo para quaternion e vice-versa, e multiplicação de quaternions.

3.5.1.4 Random

A biblioteca Random do Python fornece funções para geração de números pseudoaleatórios, permitindo criar amostras aleatórias, embaralhar sequências, e selecionar itens de uma lista. Suas funcionalidades são amplamente utilizadas em simulações, testes estatísticos e algoritmos que requerem elementos aleatórios, proporcionando flexibilidade e facilidade de uso na programação.

3.5.1.5 inspecao

Esta biblioteca contém o script para a inspeção das placas e a função para fazer a predição com o classificador treinado.

3.5.1.6 Io

A biblioteca io do Python fornece ferramentas para manipulação de fluxos de entrada e saída, oferecendo suporte para leitura e escrita de arquivos em diversos formatos. Ela facilita o gerenciamento de dados em memória, permitindo operações eficientes com arquivos binários e texto, além de suporte para manipulação de *streams* em aplicativos complexos.

3.5.1.7 load_model

A função load_model do módulo keras.models é utilizada para carregar um modelo previamente treinado e salvo em disco. O modelo treinado com o Teachable Machine foi salvo com a extensão .h5. Ao usar load_model, é possível restaurar completamente a arquitetura do modelo, seus pesos e a configuração de treinamento, permitindo realizar predições ou continuar o treinamento a partir do ponto onde foi interrompido.

3.5.2 Leitura dos termos de busca e carregamento do modelo

As strings a serem buscadas na imagem são informadas em um arquivo com a extensão “.txt” chamado “minha_lista2”, onde cada linha desse arquivo contém uma string. No início do programa do controlador, esse arquivo é lido e o seu conteúdo é guardado em uma lista que será repassada como argumento para a função verifica_imagem. Depois, o modelo treinado com o Teachable Machine é carregado para a variável model. A Figura 45 mostra o trecho para a leitura do arquivo de texto e carregamento do modelo.

Figura 45 - Leitura do arquivo de texto e carregamento do modelo

```
with io.open("minha_lista2.txt", 'r', encoding='utf-8') as arquivo:
    # Lê todo o conteúdo do arquivo
    content = arquivo.read()

# Divide o conteúdo em palavras usando o método split
lista_recuperada = content.split()

# Carrega o modelo de classifcacao
model = load_model("keras_model.h5", compile=False)
```

Fonte: O Autor (2024)

3.5.3 Configuração dos nós

O trecho de código exibido na Figura 46 mostra a configuração do robô colaborativo, da câmera, e do display. Além disso é feito a importação das ferramentas do robô e a configuração das esteiras de produtos aprovados e reprovados.

Figura 46 - Configuração dos nós

```
# Inicialização do robô
robot = Supervisor()
timestep = int(robot.getBasicTimeStep())

root_node = robot.getRoot()
children_field = root_node.getField('children')

# Configuração da câmera
camera = robot.getDevice("camera")
camera.enable(timestep)

display = robot.getDevice('display')
# obtém o nó do robô
robot_node = robot.getFromDef('roboUR5e')
# obtém o nó Pose que está no handSlot
pose_node = robot_node.getField('toolSlot').getMFNode(0)

# Cria uma instância da esteira OK
conveyorOK = robot.getFromDef('esteiraOK')
# Cria uma instância da esteira NG
conveyorNG = robot.getFromDef('esteiraNG')
```

Fonte: O Autor (2024)

A Figura 47 mostra a criação de uma lista com o nome das juntas do robô UR5e para a criação das instâncias dos motores das juntas do robô por meio da função ‘getDevice’. A Figura 47 mostra também o código da função ‘move’ utilizada para mover o robô para uma determinada posição.

Figura 47 - Configuração das juntas e função move

```
# Lista com os nomes das juntas do robô
joint_names = [
    'shoulder_pan_joint',
    'shoulder_lift_joint',
    'elbow_joint',
    'wrist_1_joint',
    'wrist_2_joint',
    'wrist_3_joint'
]
# Cria instâncias dos motores das juntas do robô
joints = [robot.getDevice(name) for name in joint_names]
# Função para mover o robô
def move(angulosJuntas):
    for i, joint in enumerate(joints):
        joint.setPosition(angulosJuntas[i])
```

Fonte: O Autor (2024)

3.5.4 Função geraPlaca

O trecho de código apresentado na Figura 48 define a função *geraPlaca*, que tem como objetivo adicionar uma nova placa (bateria) ao ambiente de simulação. Inicialmente, a função seleciona aleatoriamente o status da imagem da placa (OK ou com erro) usando a função *random.choices*, com uma probabilidade de 60% para status OK e 40% para status com erro. Se a placa tiver status OK, é criada uma string *strNode* que define um nó com a imagem correspondente. Caso contrário, é selecionada aleatoriamente uma imagem de erro entre todas possíveis. O nó da placa é então importado para a simulação usando o método *importMFNodeFromString* do campo *children_field*. A função retorna o nó da placa recém-criada e seu status. Em seguida, o código principal inicializa a primeira placa de teste e configura as velocidades iniciais das esteiras OK e NG para zero. O robô é então movido para a posição inicial com a função *move*.

Figura 48 - Função geraPlaca

```
# Função para adicionar uma bateria na simulação
def geraPlaca(children_field, robot):
    statusImg = random.choices([0, 1], weights=[0.50, 0.50], k=1)[0]
    if (statusImg == 0):
        strNode = (f'DEF bat bateriaA057 {{translation 0.3 0.0991165 0.65'
                   f' url ["{statusImg}.jpg"]}}')
    else:
        numErrorImg = random.randint(1, 3)
        strNode = (f'DEF bat bateriaA057 {{translation 0.3 0.0991165 0.65'
                   f' url ["{numErrorImg}.jpg"]}}')

    children_field.importMFNodeFromString(-1, strNode)
    placa_node = robot.getFromDef('bat')
    return placa_node, statusImg

placaTeste, resultado = geraPlaca(children_field, robot)
done_cap_image = False
i = 0
# Configura as velocidades iniciais das esteiras OK e NG
conveyorOK.getField('speed').setSFFloat(0.0)
conveyorNG.getField('speed').setSFFloat(0.0)
#Move o robô para a posição inicial
move([0.00, -1.04, 1.57, -2.09, -1.57, 0.00])
```

Fonte: O Autor (2024)

3.5.5 Loop principal

O trecho de código apresentado na Figura 49 descreve o loop principal do controlador do Webots, onde a simulação é atualizada em cada passo de tempo. Dentro do loop, a posição da placa de teste é continuamente monitorada. Quando a placa alcança uma posição específica, determinada por coordenadas próximas a (-0.549, 0.09, 0.604), e a imagem ainda não foi capturada (*done_cap_image* é falso), a câmera captura a imagem da placa, que é convertida para um *array NumPy*. A imagem é transformada de BGRA para BGR, descartando o canal alfa, e depois convertida para escala de cinza. Em seguida, a imagem em escala de cinza é processada pelo algoritmo de inspeção (função verifica_imagem), que verifica a presença e a legibilidade das informações na placa e também a presença do selo do PIM e do símbolo da lixeira. O resultado é convertido para bytes, e uma nova imagem é criada e exibida no display do

Webots. Após a exibição, a imagem é liberada da memória, e a variável done_cap_image é definida como verdadeira para evitar capturas repetidas na mesma posição.

Figura 49 - Trecho de captura da imagem

```
# Loop principal
while robot.step(timestep) != -1:
    posicaoPlacaTeste = placaTeste.getPosition()
    if (np.isclose(posicaoPlacaTeste[0], -0.549, atol=1e-3) and
            np.isclose(posicaoPlacaTeste[1], 0.09, atol=1e-2) and
            np.isclose(posicaoPlacaTeste[2], 0.604, atol=1e-2) and not done_cap_image):
        image = camera.getImage()
        # Converte a imagem para um array NumPy
        np_image = np.frombuffer(image, dtype=np.uint8).reshape((camera.getHeight(), camera.getWidth(), 4))
        # Converte a imagem para BGR (descarta o canal alfa)
        bgr_image = cv2.cvtColor(np_image, cv2.COLOR_BGRA2BGR)
        # Converte para escala de cinza
        gray_image, resultado = verifica_imagem(bgr_image, lista_recuperada, model)
        # Converta a imagem para bytes
        image_bytes = gray_image.tobytes()
        # Crie a imagem a partir dos bytes
        display_image = display.imageNew(image_bytes, Display.BGRA, camera.getWidth(), camera.getHeight())
        # Exiba a imagem no Display
        display.imagePaste(display_image, 0, 0, False)
        # Libere a imagem
        display.imageDelete(display_image)
        done_cap_image = True
```

Fonte: O Autor (2024)

O trecho de código apresentado na Figura 50 contém duas seções principais dentro do loop principal do controlador. Na primeira seção, verifica-se se a placa de teste atingiu uma posição específica no final da esteira, determinada pelas coordenadas próximas a (-1.332 ou -1.357, -1.3 ou 1.3, 0.602). Se a placa está em uma dessas posições, as velocidades das esteiras OK e NG são definidas para 0.0, parando o movimento. A placa de teste atual é então removida da simulação e uma nova placa é gerada usando a função geraPlaca.

Figura 50 - código de remoção da placa

```
if ((np.isclose(posicaoPlacaTeste[0], -1.332, atol=1e-3) or np.isclose(posicaoPlacaTeste[0], -1.357, atol=1e-3))
        and (np.isclose(posicaoPlacaTeste[1], -1.3, atol=1e-2) or np.isclose(posicaoPlacaTeste[1], 1.3, atol=1e-2))
        and np.isclose(posicaoPlacaTeste[2], 0.602, atol=1e-2)):
    conveyorOK.getField('speed').setSFFloat(0.0)
    conveyorNG.getField('speed').setSFFloat(0.0)
    placaTeste.remove()
    placaTeste, resultado = geraPlaca(children_field, robot)

if(done_cap_image):
    if(i == 0):
        move([0.14, -0.64, 1.54, -2.46, -1.57, 0.00])
    if(i == 32):
        move([0.14, -1.04, 1.57, -2.09, -1.57, 0.00])
```

Fonte: O Autor (2024)

Na segunda seção do código da Figura 50, se a captura de imagem (*done_cap_image*) foi concluída, o código controla o movimento do robô para diferentes posições em uma sequência de tempo. Inicialmente, quando i é 0, o robô é movido para uma posição específica [0.14, -0.64, 1.54, -2.46, -1.57, 0.00]. Após 32 passos de tempo, o robô é movido para uma nova posição [0.14, -1.04, 1.57, -2.09, -1.57, 0.00].

Figura 51 - Código de sucção da bateria

```
if(i > 32 and i <= 192):
    position = pose_node.getPosition()
    placaTranslation = placaTeste.getField('translation')
    placaTranslation.setSFVec3f([position[0], position[1], position[2]-0.008])
    # Converter a matriz de rotação para eixo-ângulo
    initial_axis_angle = rotation_matrix_to_axis_angle(pose_node.getOrientation())
    # Converter a rotação inicial para quaternion
    initial_quaternion = axis_angle_to_quaternion(initial_axis_angle)
    # Criar o quaternion que representa a rotação de 180 graus em relação ao eixo Y
    rotation_180_y = np.array([np.cos((12 * np.pi / 12) / 2), 0, np.sin((12 * np.pi / 12) / 2), 0])
    # Multiplicar o quaternion inicial pelo quaternion de rotação de 180 graus
    quaternion_after_180_y = multiply_quaternions(rotation_180_y, initial_quaternion)
    # Criar o quaternion que representa a rotação de -90 graus em relação ao eixo X
    rotation_minus_90_x = np.array([np.cos(-np.pi / 4), np.sin(-np.pi / 4), 0, 0])
    # Multiplicar o quaternion resultante pelo quaternion de rotação de -90 graus
    final_quaternion = multiply_quaternions(rotation_minus_90_x, quaternion_after_180_y)
    # Converter o quaternion final de volta para eixo-ângulo
    final_axis_angle = quaternion_to_axis_angle(final_quaternion)
    placaTeste.getField('rotation').setSFRotation([-final_axis_angle[0], -final_axis_angle[2],
                                                   -final_axis_angle[1], final_axis_angle[3]])
    placaTeste.resetPhysics()
```

Fonte: O Autor (2024)

O trecho de código apresentado na Figura 51 é responsável por ajustar a posição e a rotação da placa de teste durante a sucção da bateria. Quando o índice i está entre 32 e 192, a posição atual do nó de pose (*pose_node*) é obtida e impressa para depuração. A posição da placa de teste é então atualizada para corresponder à posição do nó de pose, com um pequeno ajuste no eixo *z* para simular a colocação da placa.

Em seguida, o código da Figura 51 converte a matriz de rotação do nó de pose para o formato eixo-ângulo e, então, para *quaternion*. Utiliza-se um *quaternion* para representar uma rotação de 180 graus em relação ao eixo Y e outro para uma rotação de -90 graus em relação ao eixo X. Esses *quaternions* são multiplicados para obter a rotação final da placa. Finalmente, o *quaternion*

resultante é convertido de volta para o formato eixo-ângulo, e a rotação da placa de teste é atualizada no ambiente de simulação. A física da placa é então redefinida para aplicar as novas transformações.

O trecho de código apresentado na Figura 52 é responsável pelos movimentos do robô para segregar as baterias nas esteiras OK e NG, de acordo com os resultados da inspeção. Primeiro, ao atingir o índice i igual a 64, o robô é movido para uma posição específica. Em seguida, com base no resultado da inspeção (resultado), o robô executa diferentes movimentos. Se resultado for igual a 1, o robô é movido para posições específicas aos índices 96, 128 e 160, direcionando a bateria para a esteira OK. Caso contrário, para resultado diferente de 1, o robô é movido para outras posições nos mesmos índices, direcionando a bateria para a esteira NG. Quando o índice i atinge 224, o robô retorna à sua posição inicial, e a variável *done_cap_image* é redefinida para *False*, permitindo a captura de uma nova imagem na próxima iteração. Adicionalmente, se i for maior que 200, as velocidades das esteiras OK e NG são ajustadas para movimentar as baterias para suas respectivas direções. O índice i é incrementado em cada iteração do loop para controlar o tempo e a sequência das ações do robô.

Figura 52 - Código de segregação das baterias

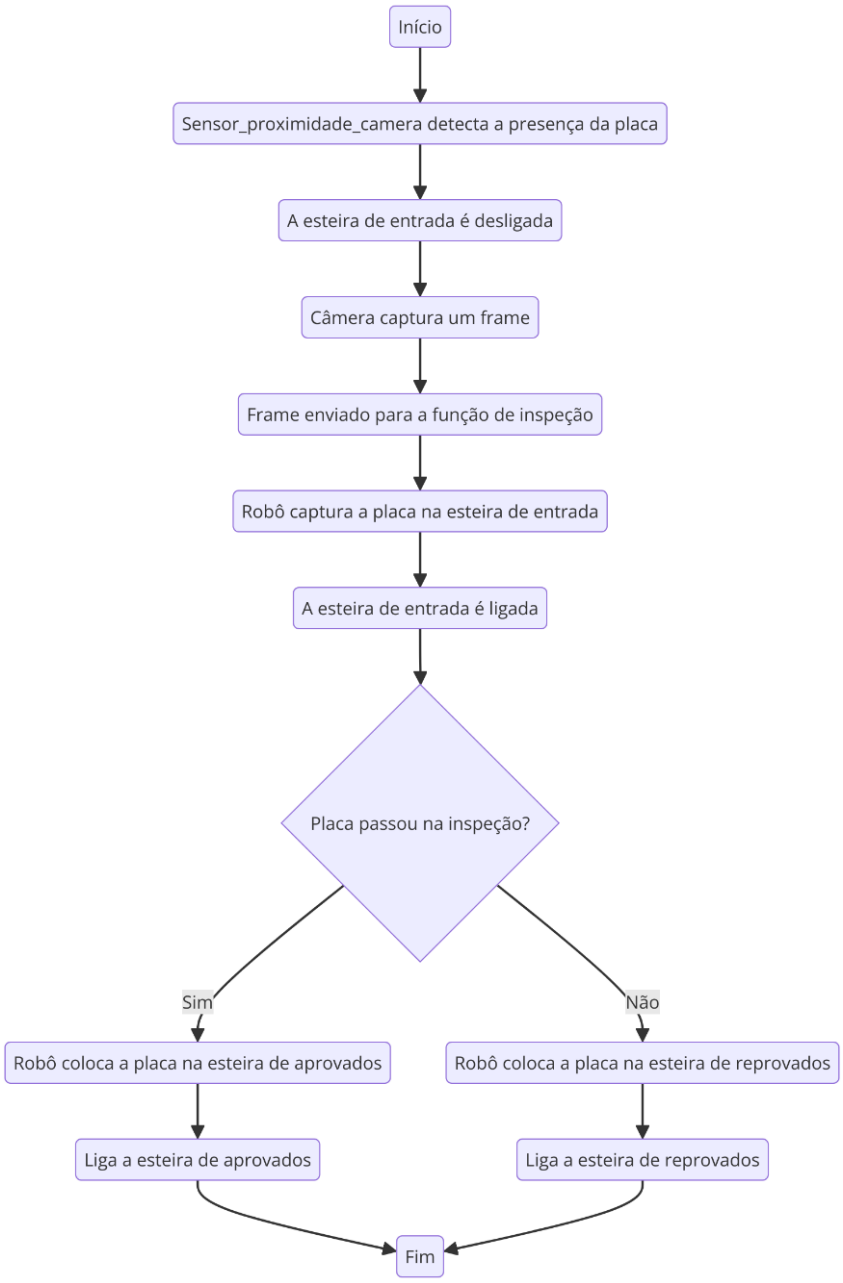
```
if (i == 64):
    move([0.14, -1.04, 1.17, -3.1, -1.57, 0.00])

if(resultado):
    if (i == 96):
        move([1.57, -1.04, 1.17, -3.1, -1.57, 0.00])
    if (i == 128):
        move([1.57,  -1.04, 1.57, -2.09, -1.57, 0.00])
    if (i == 160):
        move([1.57, -0.66, 1.57, -2.48, -1.57, 0.00])
else:
    if (i == 96):
        move([-2.20, -1.04, 1.17, -3.1, -1.57, 0.00])
    if (i == 128):
        move([-2.20, -1.04, 1.57, -2.11, -1.57, 0.00])
    if (i == 160):
        move([-2.20, -0.90, 2.25, -2.91, -1.57,  -0.63])

if (i == 224):
    move([0.00, -1.04, 1.57, -2.09, -1.57, 0.00])
    done_cap_image = False
    i = -1
if(i > 200):
    conveyorOK.getField('speed').setSFFloat(0.4)
    conveyorNG.getField('speed').setSFFloat(-0.4)
i += 1
```

Fonte: O Autor (2024)

4 RESULTADOS

O funcionamento do sistema de automação proposto neste TCC está mostrado na Figura 53. O sistema de automação detecta a presença de uma placa através do sensor Sensor_proximidade_camera, que então desliga a esteira e aciona a câmera para capturar um frame. Este frame é enviado para a função de inspeção. O robô então captura a placa, e, se ela for aprovada, o robô a coloca na esteira de aprovados e a liga; se reprovada, o robô a coloca na esteira de reprovados e a liga.

Figura 53 - diagrama de funcionamento da automação

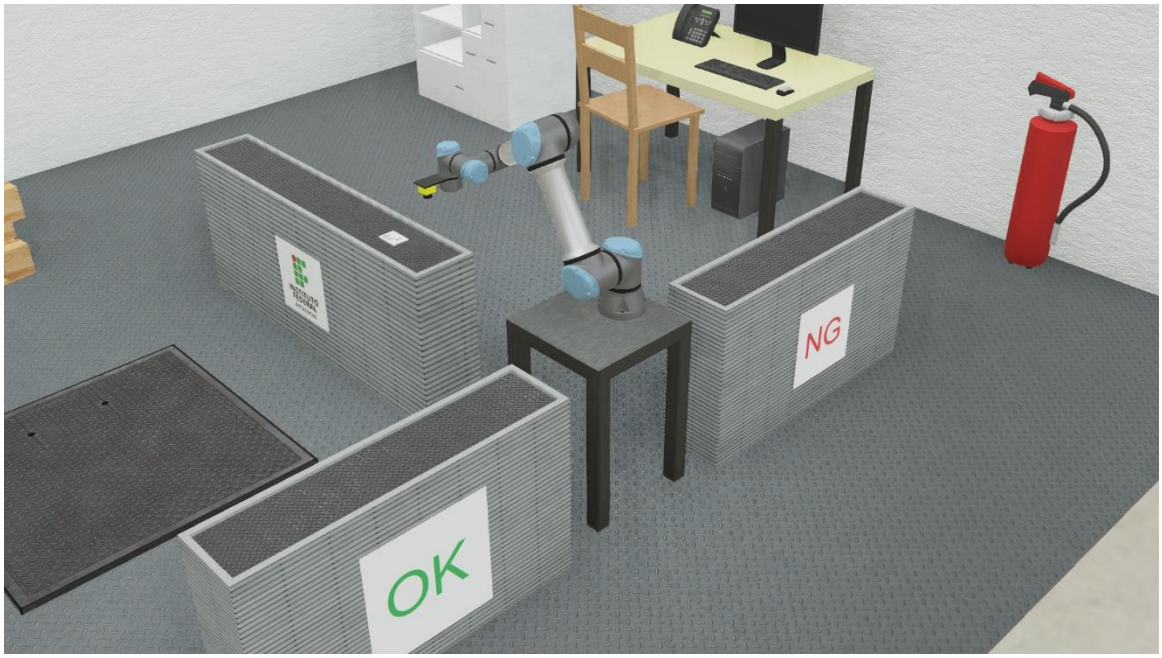


Fonte: O Autor (2024)

4.1 WEBOTS

A Figura 54 mostra a vista isométrica do mundo construído no Webots para a simulação.

Figura 54 - Mundo virtual no Webots



Fonte: O Autor (2024)

4.2 CLASSIFICADOR

A seguir, serão mostradas as métricas de desempenho geradas durante o treinamento do classificador.

4.2.1 Precisão por classe

O Teachable Machine divide as imagens em dois conjuntos:

a) Amostras de treinamento: (85% das amostras) são usadas para treinar o modelo a classificar corretamente novas amostras nas classes criadas.

b) Amostras de teste: (15% das amostras) nunca são usadas para treinar o modelo. Portanto, após o modelo ter sido treinado nas amostras de treinamento, elas são usadas para verificar o desempenho do modelo em dados novos e nunca antes vistos.

A precisão por classe é calculada usando as amostras de teste. A Figura 55 mostra a precisão por classe do modelo treinado.

Figura 55 - Precisão por classe

Accuracy per class

CLASS	ACCURACY	# SAMPLES
OK	1.00	75
garca	1.00	75
lixeira	1.00	75

Fonte: O Autor (2024)

4.2.2 Matriz de confusão

Uma matriz de confusão é uma ferramenta utilizada na avaliação do desempenho de um modelo de classificação. Ela é uma tabela que permite visualizar e comparar as previsões feitas pelo modelo em relação aos valores reais conhecidos. A matriz de confusão é composta por quatro elementos principais:

a) *True Positives* (TP): O número de casos em que o modelo previu corretamente a classe positiva.

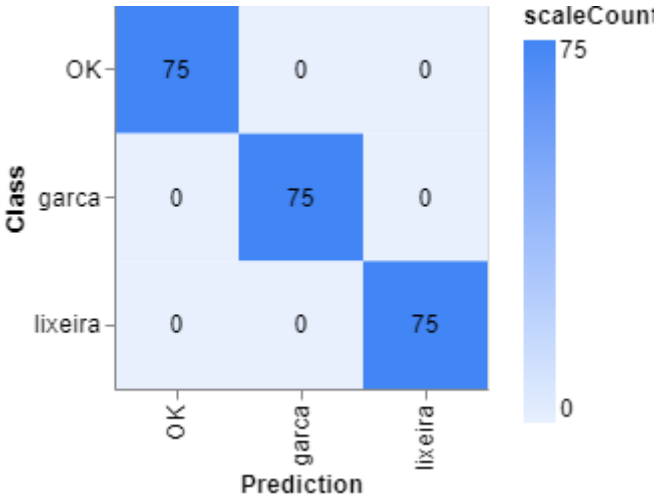
b) *True Negatives* (TN): O número de casos em que o modelo previu corretamente a classe negativa.

c) *False Positives* (FP): O número de casos em que o modelo previu erroneamente a classe positiva.

d) *False Negatives* (FN): O número de casos em que o modelo previu erroneamente a classe negativa.

A Figura 56 mostra a matriz de confusão do modelo treinado. O eixo y (Classe) representa a classe verdadeira das imagens e o eixo x (Predição) representa a classe na qual o modelo classificou as imagens.

Figura 56 - Matriz de confusão do classificador

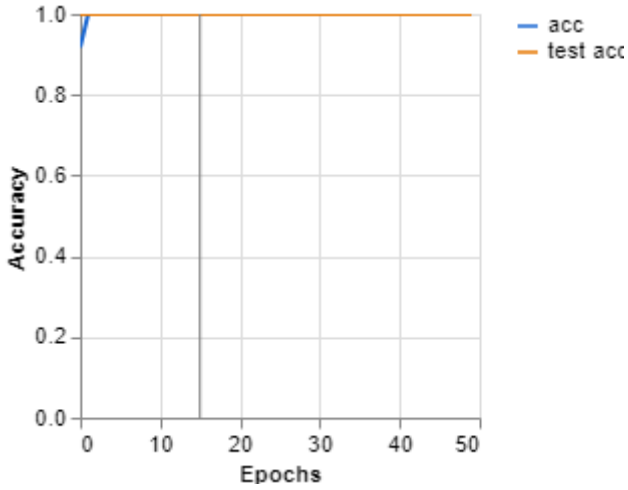


Fonte: O Autor (2024)

4.2.3 Precisão

Precisão é a porcentagem de classificações que um modelo acerta durante o treinamento. A Figura 57 mostra a precisão por época de treinamento do classificador.

Figura 57 - Precisão por época

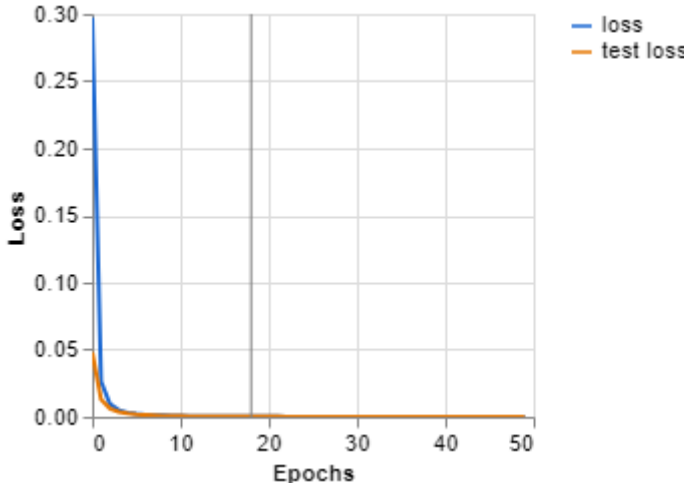


Fonte: O Autor (2024)

4.2.4 Perda

A perda (loss) é uma métrica fundamental em aprendizado de máquina que quantifica a diferença entre as previsões do modelo e os valores reais. Ela serve como uma medida do erro que o modelo está cometendo durante o treinamento e a validação. O objetivo do treinamento é minimizar essa perda, ajustando os pesos do modelo para melhorar a precisão das previsões. Na Figura 58, o gráfico de perda mostra a evolução da perda ao longo das épocas de treinamento. A curva azul representa a perda no conjunto de treinamento, enquanto a curva laranja representa a perda no conjunto de teste. A redução da perda ao longo das épocas indica que o modelo está aprendendo e ajustando-se melhor aos dados. Uma perda baixa sugere que o modelo está fazendo previsões mais precisas.

Figura 58 - Perda



Fonte: O Autor (2024)

4.3 INSPEÇÃO DE UMA BATERIA APROVADA

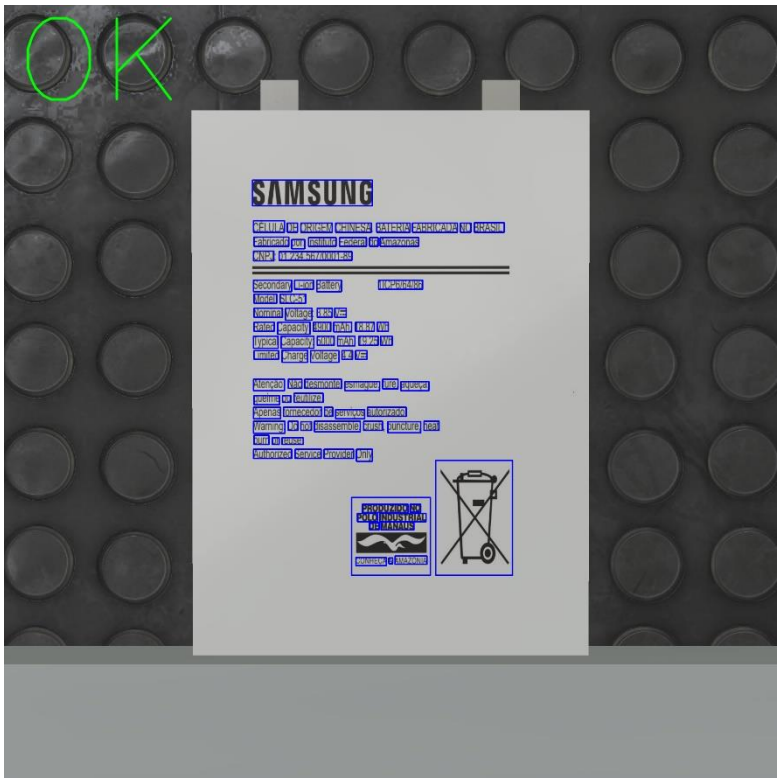
A Figura 59 mostra a modelagem da bateria feita no Webots. Por questão de confidencialidade, o nome do fabricante e o CNPJ foram alterados no rótulo da bateria.

Figura 59 - Modelagem da bateria

Fonte: O Autor (2024)

A Figura 60 mostra o resultado da inspeção para uma bateria aprovada. Foram desenhados retângulos ao redor das letras identificadas e ao redor do desenho da lixeira e do selo do PIM.

Figura 60 - Resultado da inspeção para bateria aprovada

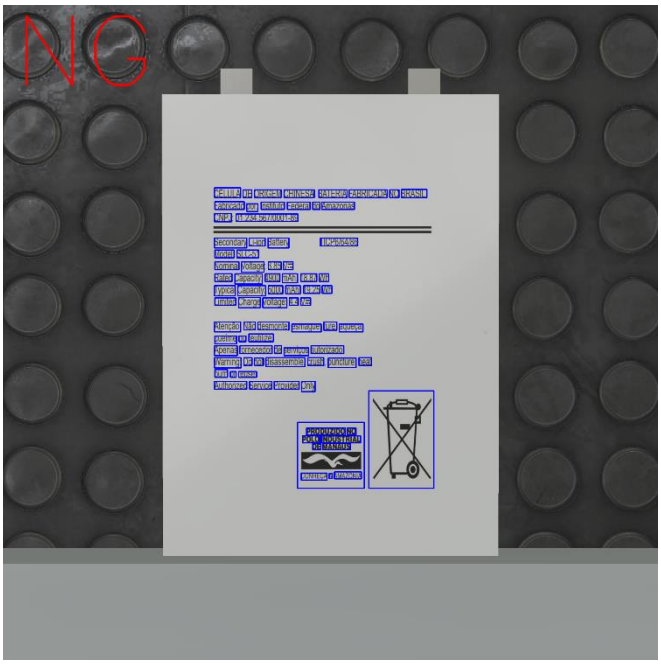


Fonte: O Autor (2024)

4.4 INSPEÇÃO OCR

A Figura 61 mostra o resultado da inspeção para uma bateria reprovada. Esse é um exemplo de um produto reprovado pela inspeção do OCR, pois na imagem é possível ver que a bateria está sem o nome da marca do fabricante de celular.

Figura 61 - Resultado da inspeção para bateria reprovada pelo OCR

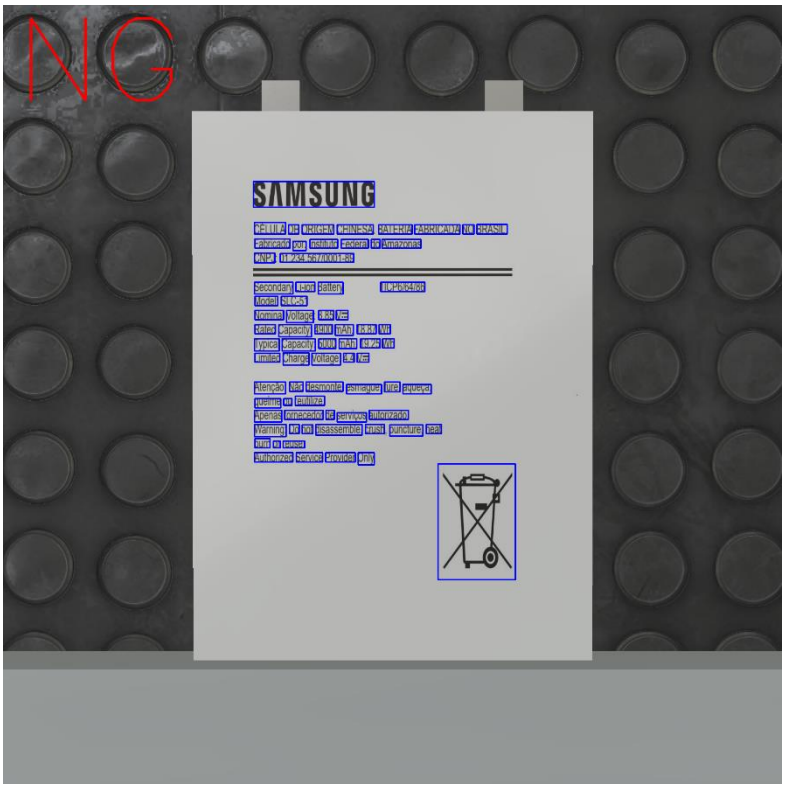


Fonte: O Autor (2024)

4.5 INSPEÇÃO CLASSIFICADOR

A Figura 62 mostra outro exemplo de bateria reprovada na inspeção. Nesse caso, o produto foi reprovado pelo classificador pois, na imagem, é possível verificar que a bateria está sem o selo do PIM.

Figura 62 - Resultado da inspeção para bateria reprovada pelo classificador



Fonte: O Autor (2024)

5 CONSIDERAÇÕES FINAIS

O objetivo deste trabalho foi validar, por meio de uma simulação no Webots, um sistema de inspeção das informações gravadas em baterias. Utilizando tecnologias avançadas de visão computacional e aprendizado de máquina, o sistema foi projetado para verificar a presença e a precisão das informações escritas nas baterias, empregando OCR implementado com o *Pytesseract*, além de um classificador treinado no *Teachable Machine* para detectar a presença de símbolos específicos, como o selo do PIM e o símbolo de lixeira.

A simulação realizada no Webots demonstrou a eficácia do sistema proposto, evidenciando a capacidade do OCR em identificar e verificar corretamente as informações textuais gravadas nas baterias. O classificador, treinado no *Teachable Machine*, mostrou-se robusto na detecção dos símbolos predefinidos, contribuindo para a precisão e confiabilidade do sistema de inspeção. Esses resultados confirmam a viabilidade de utilizar ferramentas acessíveis e de fácil integração, como o *Pytesseract* e o *Teachable Machine*, em aplicações industriais complexas.

O uso de uma simulação no Webots permitiu a avaliação detalhada do desempenho do sistema em um ambiente controlado, proporcionando insights valiosos sobre a implementação e ajustes necessários antes de sua aplicação em um ambiente real. A flexibilidade e a precisão oferecidas pela plataforma de simulação foram fundamentais para testar e validar as diferentes etapas do processo de inspeção, desde a captura de imagens até a classificação e verificação das informações.

Como sugestão para trabalhos futuros, propõe-se a melhoria do algoritmo de inspeção para não apenas verificar a presença dos itens, mas também avaliar a qualidade da gravação realizada pela tampografia. Isso incluiria a análise de parâmetros como clareza, uniformidade e integridade das impressões, garantindo um controle de qualidade mais rigoroso e abrangente. Além disso, é recomendável testar a solução desenvolvida em um ambiente real de produção, avaliando sua performance e adaptabilidade às condições

operacionais práticas. Esses avanços poderão ampliar ainda mais a aplicabilidade e a eficiência do sistema de inspeção, contribuindo para a inovação e a melhoria contínua dos processos industriais.

Em conclusão, este trabalho demonstrou a viabilidade e a eficácia de um sistema automatizado de inspeção de baterias utilizando tecnologias de OCR e aprendizado de máquina. A validação por meio de simulação no Webots foi um passo crucial para garantir a robustez e a precisão do sistema, abrindo caminho para futuras implementações e aprimoramentos que poderão beneficiar significativamente o setor industrial.

REFERÊNCIAS

BEAZLEY, D., JONES, B. K. **Python cookbook**. Sebastopol, CA. O'Reilly Media, 2013, 3° Edição. ISBN: 978-1-449-34037-7.

BIONDICH, Paul G. et al. **A modern optical character recognition system in a real world clinical setting: some accuracy and feasibility observations.** In: Proceedings of the AMIA Symposium. American Medical Informatics Association, 2002. p. 56.

CHEN, De-Sheng et al. A novel application of unsupervised machine learning and supervised machine learning-derived radiomics in anterior cruciate ligament rupture. **Risk Management and Healthcare Policy**, p. 2657-2664, 2021.

CUTTER, Michael; MANDUCHI, Roberto. Improving the accessibility of mobile OCR apps via interactive modalities. **ACM Transactions on Accessible Computing (TACCESS)**, v. 10, n. 4, p. 1-27, 2017.

CYBERBOTICS. **Webots User Guide**. 2023. Disponível em: <https://cyberbotics.com/doc/guide/index>. Acesso em: 24 jun. 2024.

DWIVEDI, Utkarsh et al. Examining the Values Reflected by Children during AI Problem Formulation. **arXiv preprint arXiv:2309.15839**, 2023.

DWIVEDI, Utkarsh et al. Exploring machine teaching with children. In: **2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)**. IEEE, 2021. p. 1-11.

FORCHHAMMER, Stephan et al. Development of an image analysis-based prognosis score using Google's teachable machine in melanoma. **Cancers**, v. 14, n. 9, p. 2243, 2022.

GONZALEZ, R. C., WOODS, R.E., EDDINS, S. L. **Digital image processing using Matlab**. New Jersey, Pearson Prentice Hall, 2004. ISBN 0-13-008519-7.

HOM, Julie et al. **Facilitating clinical research through automation: Combining optical character recognition with natural language processing**. **Clinical Trials**, v. 19, n. 5, p. 504-511, 2022.

KALSHETTY, Ashwini; RAKSHIT, Sutapa. Use case of no code machine learning tools for medical image classification. 2021.

KHASTGIR, Siddartha et al. Calibrating trust through knowledge: Introducing the concept of informed safety for automation in vehicles. **Transportation research part C: emerging technologies**, v. 96, p. 290-303, 2018.

KOERT, Dorothea et al. Multi-channel interactive reinforcement learning for sequential tasks. **Frontiers in Robotics and AI**, v. 7, p. 97, 2020.

LAIQUE, Sobia Nasir et al. Application of optical character recognition with natural language processing for large-scale quality metric data extraction in colonoscopy reports. **Gastrointestinal endoscopy**, v. 93, n. 3, p. 750-757, 2021.

LOCKS, Francisco et al. Biomechanical exposure of industrial workers–Influence of automation process. **International Journal of Industrial Ergonomics**, v. 67, p. 41-52, 2018.

MORDVINTSEV, A., ABID, K. **OpenCV-Python Tutorials Documentation**, Release 1, 2017.

NGUYEN, Anthony et al. **Generating high-quality data abstractions from scanned clinical records: text-mining-assisted extraction of endometrial carcinoma pathology features as proof of principle.** BMJ open, v. 10, n. 6, p. e037740, 2020.

OPENCV. **About**. 2024. Disponível em: < https://opencv.org/about/>. Acesso em: 24 jun. 2024.

Oscar Flues. **O que é tampografia.** 2016. Disponível em: https://oscarflues.com.br/#o-que-e-tampografiaa17c-ab62. Acesso em 19 Jun. 2024.

Pytesseract Documentation 9.2.0. 2024. Disponível em:< https://pytesseract.readthedocs.io/en/latest/. Acesso em 06 Jun 2024.

PYTESSERACT. **PYTESSERACT** 9.2.0. 2024. Disponível em:< https://github.com/madmaze/pytesseract. Acesso em 06 Jun 2024.

PYTHON. **HOME**. 2024. Disponível em: < https://www.python.org/>. Acesso em: 24 jun. 2024.

QUEIROZ, J. E. R., GOMES, H. **Introdução ao Processamento Digital de Imagens**. Revista RITA, Vol. 3. Número 1, 2001.

RADI, Marwan et al. Telepresence in industrial applications: implementation issues for assembly tasks. **Presence: Teleoperators and Virtual Environments**, v. 19, n. 5, p. 415-429, 2010.

RUSSEL, S. NORVIG, P. **Artificial intelligence: a modern approach**. New Jersey, Pearson Prentice Hall, 2010, 3° Edição. ISBN-13: 978-0-13-604259-4.

SAMPLE, Pamela A. et al. Using unsupervised learning with variational bayesian mixture of factor analysis to identify patterns of glaucomatous visual field defects. **Investigative ophthalmology & visual science**, v. 45, n. 8, p. 2596-2605, 2004.

SARZYNSKI, Erin et al. Beta testing a novel smartphone application to improve medication adherence. **Telemedicine and e-Health**, v. 23, n. 4, p. 339-348, 2017.

SAUER, Juergen; NICKEL, Peter; WASTELL, David. Designing automation for complex work environments under different levels of stress. **Applied ergonomics**, v. 44, n. 1, p. 119-127, 2013.

SCHWABE, Henrik; CASTELLACCI, Fulvio. Automation, workers' skills and job satisfaction. **Plos one**, v. 15, n. 11, p. e0242929, 2020.

SVENSSON, Åsa et al. Automation, teamwork, and the feared loss of safety: Air traffic controllers' experiences and expectations on current and future ATM systems. In: **Proceedings of the 32nd European Conference on Cognitive Ergonomics**. 2021. p. 1-8.

TRUCCO, E. VERRI, A. **Introductory Techniques for 3-D Computer Vision**. New Jersey, Pearson Prentice Hall, 1998. ISBN-10 0132611082.

TSAO, Liuxing et al. Modelling performance during repetitive precision tasks using wearable sensors: A data-driven approach. **Ergonomics**, v. 63, n. 7, p. 831-849, 2020.

WEBOTS. **Cloud**. 2024. Disponível em: < https://webots.cloud/>. Acesso em: 26 jun. 2024.

WEBOTS. **Cloud**. 2024. Disponível em: < https://webots.cloud/>. Acesso em: 26 jun. 2024.

WELFARE, Katherine S. et al. Consider the human work experience when integrating robotics in the workplace. In: **2019 14th ACM/IEEE international conference on human-robot interaction (HRI)**. IEEE, 2019. p. 75-84.

APÊNDICE A – Código VRML do mundo criado no Webots

```
#VRML_SIM R2023b utf8

EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/objects/backgrounds/protos/TexturedBackground.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/objects/factory/conveyors/protos/ConveyorBelt.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/appearances/protos/BrushedAluminium.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/robots/universal_robots/protos/UR5e.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/objects/geometries/protos/Rectangle.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/objects/tables/protos/Table.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/objects/factory/manhole/protos/SquareManhole.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/objects/computers/protos/Monitor.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/objects/computers/protos/Keyboard.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/objects/computers/protos/ComputerMouse.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/objects/computers/protos/DesktopComputer.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/objects/telephone/protos/OfficeTelephone.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/objects/chairs/protos/WoodenChair.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/objects/cabinet/protos/Cabinet.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/objects/factory/fire_extinguisher/protos/FireExtinguisher.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/objects/factory/pallet/protos/WoodenPalletStack.proto"
```

```
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/obj
ects/floors/protos/Floor.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/app
earances/protos/ThreadMetalPlate.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/app
earances/protos/Roughcast.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/obj
ects/factory/containers/protos/CardboardBox.proto"
EXTERNPROTO
"https://raw.githubusercontent.com/cyberbotics/webots/R2023b/projects/obj
ects/factory/tools/protos/PlatformCart.proto"
IMPORTABLE EXTERNPROTO "../protos/bateriaA057.proto"
EXTERNPROTO "../protos/suporteCam_UR5e.proto"
EXTERNPROTO "../protos/placaOKNG.proto"

WorldInfo {
  basicTimeStep 16
}
Viewpoint {
  fieldOfView 0.9
  orientation -0.21733453481485632 -0.6528578397996306 0.7256323731676873
5.451333049694524
  position -5.628496223789173 3.001627245493209 4.290984997936865
  exposure 1.7
}
TexturedBackground {
  texture "factory"
}
Floor {
  size 5 5
  appearance ThreadMetalPlate {
  }
}
Monitor {
  translation -0.270032 -2.03973 0.67
  rotation 0 0 -1 -1.5699953071795862
}
ConveyorBelt {
  translation 0.107918 0.0991165 0
  size 1.4 0.2 0.6
  borderThickness 0.02
  speed -0.3
}
DEF roboUR5e UR5e {
  translation -1.2 -0.13 0.6
  controller "<extern>"
  supervisor TRUE
  toolSlot [
    DEF VACUM Pose {
      rotation 0 1 0 3.14
    }
```

```
    suporteCam_UR5e {
      translation -0.05 -0.052 0
      rotation 1 0 0 -1.57
    }
    Camera {
      translation -0.095 0.077 0
      rotation 0 0 1 1.57
      width 1600
      height 1600
    }
    Display {
      width 1600
      height 1600
    }
  ]
}
DEF esteiraOK ConveyorBelt {
  translation -1.33 0.85 0
  rotation 0 0 1 1.57
  name "esteiraAprovado"
  size 1 0.2 0.6
  borderThickness 0.02
  speed 0
}
DEF esteiraNG ConveyorBelt {
  translation -1.33 -0.85 0
  rotation 0 0 1 1.57
  name "esteiraNG"
  size 1 0.2 0.6
  borderThickness 0.02
  speed 0
}
Solid {
  translation -1.47 0.85 0.3
  children [
    Shape {
      appearance PBRAppearance {
        baseColorMap ImageTexture {
          url [
            "../protos/textures/OK.png"
          ]
        }
        metalness 0
      }
      geometry Rectangle {
        size 0.4 0.4
      }
    }
  ]
  name "solid(1)"
}
placaOKNG {
  translation -1.47 -0.85 0.3
  name "placaNG"
  url [
```

```
    "../protos/textures/NG.png"
  ]
}
Table {
  translation 0 -2 0
  size 1.2 0.7 0.66
  feetSize 0.05 0.05
  trayAppearance PBRAppearance {
    baseColor 0.926 0.912 0.698
    roughness 0.8
    metalness 0
  }
  legAppearance BrushedAluminium {
    colorOverride 0.333 0.341 0.325
  }
}
Table {
  translation -1.2 -0.03 0
  name "table(1)"
  size 0.4 0.4 0.6
  feetSize 0.05 0.05
  trayAppearance BrushedAluminium {
    colorOverride 0.333 0.341 0.325
  }
  legAppearance BrushedAluminium {
    colorOverride 0.333 0.341 0.325
  }
}
SquareManhole {
  translation 0 1.1 -0.03
}
Keyboard {
  translation -0.239872 -1.83977 0.67
  rotation 0 0 1 1.57
}
ComputerMouse {
  translation -0.510143999999999 -1.8204900000000004 0.6599803800001015
  rotation      -1.8380656700942722e-06      1.8366025516815409e-06      -
0.9999999999966243 1.5700000000033754
}
DesktopComputer {
  translation -0.239976 -1.96976 0
  rotation 0 0 1 1.57
}
OfficeTelephone {
  translation 0.22000158279800666 -2.000002079908695 0.6634731356355306
  rotation 0.0004957395234327665 -6.72441055365746e-05 0.9999998748602698
1.5700005413911062
}
WoodenChair {
  translation 0.138001 -1.61842 0
  rotation 0 0 1 -1.7017003061004252
}
Cabinet {
  translation 1.61895 -2.24149 0
```

```
  rotation 0 0 1 1.57
}
FireExtinguisher {
  translation -1.44013 -2.38222 -0.00235426
  rotation          -0.0017524193079187739          -0.0013110594822245737
0.9999976050719337 -1.5707953071795862
}
WoodenPalletStack {
  translation 1.82751 1.38377 0
  rotation 0 0 -1 -1.7016996938995748
  palletNumber 4
  lateralMisalignment 0.2
  longitudinalMisalignment 0.01
}
DEF WALL1 Pose {
  translation 0 2.5 1
  rotation 1 0 0 1.5708
  children [
    DEF wall_long Shape {
      appearance Roughcast {
      }
      geometry Plane {
        size 5 2
      }
    }
  ]
}
DEF WALL2 Pose {
  translation 0 -2.5 1
  rotation 1 0 0 -1.5708
  children [
    USE wall_long
  ]
}
DEF WALL3 Pose {
  translation 2.5 0 1
  rotation  -0.5773509358554485  0.5773489358556708  0.5773509358554485  -
2.094395307179586
  children [
    DEF wall_short Shape {
      appearance Roughcast {
      }
      geometry Plane {
        size 5 2
      }
    }
  ]
}
DEF WALL4 Pose {
  translation -2.5 0 1
  rotation 0.5773506025225371 0.5773496025232256 0.5773506025225371 2.0944
  children [
    USE wall_short
  ]
}
```

```
CardboardBox {
  translation -0.87 2.09 0.3
  size 0.4 0.4 0.4
}
CardboardBox {
  translation -0.5 2.11 0.3
  name "cardboard box(1)"
  size 0.3 0.3 0.4
}
PlatformCart {
  translation -0.72002 2.1 -0.00320522
  rotation  0.03318982705221955  0.9994488358663943  0.0006781338754610068
0.0001636039987924022
}
```

APÊNDICE B – Algoritmo de inspeção

```
import cv2
import pytesseract
from pytesseract import Output
import numpy as np
# Especifique o caminho para o executável do Tesseract
pytesseract.pytesseract.tesseract_cmd =
r'C:\Users\prafa\AppData\Local\Programs\Tesseract-
OCR\tesseract.exe'

def Classificador(img, model):
    # Extrai a regiao de interesse da imagem
    image = img[320:1200, 470:1100]
    # Redimensiona a imagem bruta em pixels (224 de altura e 224
de largura)
    image = cv2.resize(image, (224, 224),
interpolation=cv2.INTER_AREA)
    # Transforme a imagem em um array numpy e remodele-a de
    # acordo com o formato de entrada do modelo.
    image = np.asarray(image, dtype=np.float32).reshape(1, 224,
224, 3)

    # Normalize a matriz de imagens
    image = (image / 127.5) - 1
    # Prevê com o modelo
    prediction = model.predict(image)
    index = np.argmax(prediction)

    return index
# Função principal que processa a imagem e verifica as strings
def verifica_imagem(imagem, strings_de_busca, model):
    strings_encontradas = set()
    resultados = pytesseract.image_to_data(imagem,
output_type=Output.DICT, lang='por')
    res_classificador = Classificador(imagem, model)
    for i in range(0, len(resultados['text'])):
        if int(resultados['conf'][i]) > 16 and
resultados['text'][i] != '':
            texto = resultados['text'][i].strip()
            if texto in strings_de_busca:
                strings_encontradas.add(texto)
                (x, y, w, h) = (resultados['left'][i],
resultados['top'][i],
                                resultados['width'][i],
resultados['height'][i])
                cv2.rectangle(imagem, (x, y), (x + w, y + h),
(255, 0, 0), 2)
```

```
    l = [(719, 1019, 882, 1180), (893, 942, 1052, 1180)]
    for i in range(2):
        if (i + 1) != res_classificador:
            cv2.rectangle(imagem, l[i][0:2], l[i][2:4], (255, 0,
0), 2)
    resultado_inspecao_ocr = False
    # Verifica se todas as strings foram encontradas
    if all(string in strings_encontradas for string in
strings_de_busca):
        resultado_inspecao_ocr = True

    resultado_inspecao = resultado_inspecao_ocr and
(res_classificador == 0)
    if resultado_inspecao:
        cv2.putText(imagem, 'OK', (25, 195),
cv2.FONT_HERSHEY_SIMPLEX, 8,
                    (0, 255, 0), 3, cv2.LINE_AA)
    else:
        cv2.putText(imagem, 'NG', (25, 195),
cv2.FONT_HERSHEY_SIMPLEX, 8,
                    (0, 0, 255), 3, cv2.LINE_AA)

    imagem = cv2.cvtColor(imagem, cv2.COLOR_BGR2BGRA)

    return imagem, resultado_inspecao
```

APÊNDICE C – Algoritmo de controle do robô UR5e

```
from controller import Supervisor,  Display
from manipula_rotacao import *
import random
from inspecao_bateria_rev01_traduzido import *
import io
from keras.models import load_model

with io.open("minha_lista2.txt", 'r', encoding='utf-8') as
arquivo:
    # Lê todo o conteúdo do arquivo
    content = arquivo.read()

# Divide o conteúdo em palavras usando o método split
lista_recuperada = content.split()

# Carrega o modelo de classifcacao
model = load_model("keras_model.h5", compile=False)

# Inicialização do robô
robot = Supervisor()
timestep = int(robot.getBasicTimeStep())

root_node = robot.getRoot()
children_field = root_node.getField('children')

# Configuração da câmera
camera = robot.getDevice("camera")
camera.enable(timestep)

display = robot.getDevice('display')
# obtém o nó do robô
robot_node = robot.getFromDef('roboUR5e')
# obtém o nó Pose que está no handSlot
pose_node = robot_node.getField('toolSlot').getMFNode(0)

# Cria uma instância da esteira OK
conveyorOK = robot.getFromDef('esteiraOK')
# Cria uma instância da esteira NG
conveyorNG = robot.getFromDef('esteiraNG')

# Lista com os nomes das juntas do robô
joint_names = [
    'shoulder_pan_joint',
    'shoulder_lift_joint',
    'elbow_joint',
    'wrist_1_joint',
    'wrist_2_joint',
    'wrist_3_joint'
]
# Cria instâncias dos motores das juntas do robô
joints = [robot.getDevice(name) for name in joint_names]
```

```
# Função para mover o robô
def move(angulosJuntas):
    for i, joint in enumerate(joints):
        joint.setPosition(angulosJuntas[i])

# Função para adicionar uma bateria na simulação
def geraPlaca(children_field, robot):
    statusImg = random.choices([0, 1], weights=[0.50, 0.50],
k=1)[0]
    if (statusImg == 0):
        strNode = (f'DEF bat bateriaA057 {{translation 0.3
0.0991165 0.65'
                   f' url ["{statusImg}.jpg"]}}')
    else:
        numErrorImg = random.randint(1, 3)
        strNode = (f'DEF bat bateriaA057 {{translation 0.3
0.0991165 0.65'
                   f' url ["{numErrorImg}.jpg"]}}')

    children_field.importMFNodeFromString(-1, strNode)
    placa_node = robot.getFromDef('bat')
    return placa_node

placaTeste = geraPlaca(children_field, robot)
done_cap_image = False
i = 0
# Configura as velocidades iniciais das esteiras OK e NG
conveyorOK.getField('speed').setSFFloat(0.0)
conveyorNG.getField('speed').setSFFloat(0.0)
#Move o robô para a posição inicial
move([0.00, -1.04, 1.57, -2.09, -1.57, 0.00])
k = 547
# Loop principal
while robot.step(timestep) != -1:
    posicaoPlacaTeste = placaTeste.getPosition()
    if (np.isclose(posicaoPlacaTeste[0], -0.549, atol=1e-3) and
np.isclose(posicaoPlacaTeste[1], 0.09, atol=1e-2)
            and np.isclose(posicaoPlacaTeste[2], 0.604, atol=1e-
2) and not done_cap_image):
        image = camera.getImage()
        # Converte a imagem para um array NumPy
        np_image = np.frombuffer(image,
dtype=np.uint8).reshape((camera.getHeight(), camera.getWidth(),
4))
        # Converte a imagem para BGR (descarta o canal alfa)
        bgr_image = cv2.cvtColor(np_image, cv2.COLOR_BGRA2BGR)
        # Converte para escala de cinza
        gray_image, resultado = verifica_imagem(bgr_image,
lista_recuperada, model)
        # Converta a imagem para bytes
        image_bytes = gray_image.tobytes()
        # Crie a imagem a partir dos bytes
        display_image = display.imageNew(image_bytes,
Display.BGRA, camera.getWidth(), camera.getHeight())
        # Exiba a imagem no Display
```

```
        display.imagePaste(display_image, 0, 0, False)
        # Libere a imagem
        display.imageDelete(display_image)
        done_cap_image = True

    if ((np.isclose(posicaoPlacaTeste[0], -1.332, atol=1e-3) or
np.isclose(posicaoPlacaTeste[0], -1.357, atol=1e-3)) and
(np.isclose(posicaoPlacaTeste[1], -1.3, atol=1e-2)
    or np.isclose(posicaoPlacaTeste[1], 1.3, atol=1e-2)) and
np.isclose(posicaoPlacaTeste[2], 0.602, atol=1e-2)):
        conveyorOK.getField('speed').setSFFloat(0.0)
        conveyorNG.getField('speed').setSFFloat(0.0)
        placaTeste.remove()
        placaTeste = geraPlaca(children_field, robot)

    if(done_cap_image):
        if(i == 0):
            move([0.14, -0.64, 1.54, -2.46, -1.57, 0.00])
        if(i == 32):
            move([0.14, -1.04, 1.57, -2.09, -1.57, 0.00])
        if(i > 32 and i <= 192):
            position = pose_node.getPosition()
            placaTranslation =
placaTeste.getField('translation')
            placaTranslation.setSFVec3f([position[0],
position[1], position[2]-0.008])
            # Converter a matriz de rotação para eixo-ângulo
            initial_axis_angle =
rotation_matrix_to_axis_angle(pose_node.getOrientation())
            # Converter a rotação inicial para quaternion
            initial_quaternion =
axis_angle_to_quaternion(initial_axis_angle)
            # Criar o quaternion que representa a rotação de 180
graus em relação ao eixo Y
            rotation_180_y = np.array([np.cos((12 * np.pi / 12)
/ 2), 0, np.sin((12 * np.pi / 12) / 2), 0])
            # Multiplicar o quaternion inicial pelo quaternion
de rotação de 180 graus
            quaternion_after_180_y =
multiply_quaternions(rotation_180_y, initial_quaternion)
            # Criar o quaternion que representa a rotação de -90
graus em relação ao eixo X
            rotation_minus_90_x = np.array([np.cos(-np.pi / 4),
np.sin(-np.pi / 4), 0, 0])
            # Multiplicar o quaternion resultante pelo
quaternion de rotação de -90 graus
            final_quaternion =
multiply_quaternions(rotation_minus_90_x,
quaternion_after_180_y)
            # Converter o quaternion final de volta para eixo-
ângulo
            final_axis_angle =
quaternion_to_axis_angle(final_quaternion)
            placaTeste.getField('rotation').setSFRotation([-
final_axis_angle[0], -final_axis_angle[2],
```

```
                                                                -
final_axis_angle[1], final_axis_angle[3]])
            placaTeste.resetPhysics()

        if (i == 64):
            move([0.14, -1.04, 1.17, -3.1, -1.57, 0.00])

        if(resultado):
            if (i == 96):
                move([1.57, -1.04, 1.17, -3.1, -1.57, 0.00])
            if (i == 128):
                move([1.57,  -1.04, 1.57, -2.09, -1.57, 0.00])
            if (i == 160):
                move([1.57, -0.66, 1.57, -2.48, -1.57, 0.00])
        else:
            if (i == 96):
                move([-2.20, -1.04, 1.17, -3.1, -1.57, 0.00])
            if (i == 128):
                move([-2.20, -1.04, 1.57, -2.11, -1.57, 0.00])
            if (i == 160):
                move([-2.20, -0.90, 2.25, -2.91, -1.57,  -0.63])

        if (i == 224):
            move([0.00, -1.04, 1.57, -2.09, -1.57, 0.00])
            done_cap_image = False
            i = -1
        if(i > 200):
            conveyorOK.getField('speed').setSFFloat(0.4)
            conveyorNG.getField('speed').setSFFloat(-0.4)
        i += 1
```