Instituto Politécnico
de Castelo Branco
Escola Superior
de Tecnologia

Sistema de reconhecimento de ações para sistemas robóticos

Relatorio de Projeto

Aluno: Daniel Prata Dias

Orientadores

Paulo Jorge Sequeira Gonçalves

Trabalho de Projeto apresentado à Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco para cumprimento dos requisitos necessários à obtenção do grau de Licenciado em Engenharia e Gestão Industrial, realizada sob a orientação científica do Doutor Paulo Jorge Sequeira Gonçalves, do Instituto Politécnico de Castelo Branco.

Setembro de 2024

Composição do júri

Presidente do júri

Doutor Paulo Jorge Sequeira Gonçalves

Professor Coordenador, Escola Superior de Tecnologia de Castelo Branco

Vogais

Samuel Sousa Santos

Licenciado em Engenharia Industrial, APTIV Castelo Branco

Doutor Luís Miguel Pedroso de Moura Correia

Professor Adjunto, Escola Superior de Tecnologia de Castelo Branco

Agradecimentos

Ao professor Paulo Jorge Sequeira Gonçalves, pela sua orientação, apoio e auxílio prestados ao longo do desenvolvimento do presente projeto. A todos os que, de uma maneira ou de outra, apoiaram durante o desenvolvimento do presente projeto

Resumo

Este projeto visa o desenvolvimento e estudo de sistemas inteligentes de reconhecimento de objetos e ações para aplicação em sistemas robóticos em ambientes industriais. O trabalho envolve a análise de sistemas de reconhecimento utilizados em postos de trabalho na indústria, a identificação de um estudo de caso numa empresa da zona industrial de Castelo Branco, e a implementação de um sistema de reconhecimento de objetos e ações em um ambiente relevante para possível aplicação industrial.

Palavras chave

Aprendizagem profunda;

Etiquetagem;

Deteção de objetos;

Sequências de Operações;

Processamento de imagem

Abstract

This project aims to develop and study intelligent systems for object recognition and actions for use in robotic systems in industrial environments. The work involves analyzing recognition systems used in industrial workplaces, identifying a case study in a company from the industrial area of Castelo Branco, and implementing a system for object and action recognition in a relevant environment for potential industrial application.

Keywords

Labelling;

Code;

Detection;

Images training;

Operation Sequence.

Índice

Índice de figuras

Lista de tabelas

Lista de abreviaturas, siglas e acrónimos

IDE – Integrated Development Environment

IA – Inteligência Artificial

IoT – Internet of Things

YOLO – You Only Look Once

mAP – mean Average Precision

1. Introdução

Nos últimos anos, o campo da robótica tem testemunhado avanços significativos, impulsionados pelo desenvolvimento de tecnologias de reconhecimento de ações. Os sistemas de reconhecimento em robótica envolvem a utilização de sensores, algoritmos de processamento de dados e inteligência artificial para perceber e interpretar o ambiente ao redor do robô. Esses sistemas permitem que robôs identifiquem objetos e reconheçam padrões. Paralelamente, os sistemas de ações automatizadas possibilitam que robôs tomem decisões e executem tarefas de forma autónoma. Utilizando os dados coletados pelos sistemas de reconhecimento, os robôs podem planear e realizar uma variedade de ações, desde movimentos simples, como a navegação em um espaço, até tarefas complexas, como manipulação de objetos ou interação social.

A integração de reconhecimento e ações em sistemas robóticos abre novas fronteiras em diversas áreas, incluindo manufatura, saúde, serviços domésticos e até mesmo exploração espacial. Esta sinergia não apenas melhora a eficiência e precisão das operações robóticas, mas também amplia as capacidades dos robôs, tornando-os mais adaptáveis e responsivos aos ambientes dinâmicos e imprevisíveis. Neste contexto, a pesquisa e o desenvolvimento contínuo de tecnologias de reconhecimento e ações são cruciais para o avanço da robótica e sua aplicação em um mundo cada vez mais automatizado

No âmbito do Projeto, desenvolveu-se o software da aplicação que passou por várias fases cruciais de evolução. Ao longo deste processo, estudou-se a fundo diversos algoritmos de processamento de imagem e técnicas de visão com aplicações práticas, conduzindo a testes rigorosos para avaliar a eficiência de cada algoritmo. Esta fase permitiu identificar os algoritmos mais eficazes em termos de desempenho.

A primeira fase do projeto centrou-se no estudo de sistemas inteligentes de reconhecimentos de objetos e ações humanas. Foi realizada uma pesquisa exaustiva em torno do objetivo de desenvolver um software com os algoritmos necessários e as bibliotecas necessárias para desenvolver software necessário aos objetivos do projeto.

Após realizado esse estudo, foi escolhida a linguagem Python para desenvolver o código e o IDE Pycharm para o desenvolvimento do mesmo

Na segunda fase do projeto desenvolveu-se os programas necessários para obter o sistema de reconhecimento de objetos juntamente com as bibliotecas necessárias para o mesmo. Foram realizados testes com imagens de objetos e o seu respetivo treino para testar o código de reconhecimento de objetos.

Nesta fase foi desenvolvido um programa mais detalhado para garantir a montagem correta das peças e com a sequência de ações correta. Nesta fase foi realizado um estudo detalhado para aprimorar a eficiência da interface gráfica, implementando a execução em paralelo com o uso de *threads*. Essa abordagem resultou numa interface mais ágil e responsiva. Foram encontradas algumas dificuldades na primeira abordagem tomada no que toca á orientação da fase de montagem. Após a análise foi tomada a decisão de orientar as fases de montagem com base em coordenadas relativas entre objetos, na qual foi possível obter um sistema muito mais preciso e completo.

Adicionalmente, explorou-se a possibilidade de incorporar o *tracking* de ações humanas no sistema através dos métodos do *MediaPipe.* Foi tomada a decisão de não incorporar este sistema, sendo que não apresenta ética necessária para este sistema já funcional nos quais os objetivos principais já tenham sido obtidos. Este tipo de metodologia exigia também uma capacidade de processamento adicional na qual tornava-se desnecessária e prejudicial para o processamento do resto do sistema. O sistema centralizou-se exclusivamente na deteção de objetos a fazer.

Na terceira fase, foi realizado o teste final no qual usamos o sistema mais indicado para desempenhar a função num posto de trabalho numa empresa da zona Industrial de Castelo Branco. A empresa BITZER (Portugal) - Compressores para Frio, S.A. cedeu as peças necessárias para poder desenvolver um sistema compatível que se aplicasse a uma tarefa na linha de montagem

1.1 Objetivos

Este projeto consistiu no desenvolvimento de um sistema de deteção de objetos destinado a otimizar processos em bancadas de trabalho no setor industrial. O principal objetivo foi criar uma solução que pudesse identificar e monitorizar automaticamente os objetos presentes na área de trabalho, com foco em melhorar a eficiência e segurança no ambiente de produção. O desenvolvimento do sistema envolveu um estudo inicial das tecnologias necessárias para sua implementação, como algoritmos de visão computacional, bibliotecas de processamento de imagem, e técnicas de *machine learning*. Após essa fase de pesquisa, foram realizados diversos testes para validar as abordagens propostas. O algoritmo final foi desenvolvido em Python [1], utilizando ferramentas como OpenCV [2] e bibliotecas de *machine learning* para deteção de objetos em tempo real. A solução foi testada e validada numa tarefa de montagem da empresa Bitzer [3] onde os resultados demonstraram sua eficácia em detetar e classificar corretamente os objetos e as sequências de montagem nas bancadas de trabalho.

1.2 Cronograma

Para a consecução dos objetivos propostos foram previstas as seguintes etapas e correspondente calendarização:

a) Revisão Bibliográfica

b) Escolha de metodologia

c) Inicio de implementação do sistema de reconhecimento de objetos e ações em ambiente de simulação

d) Aquisição de dados relevantes para treino e teste do sistema. Desenvolvimento dos algoritmos de reconhecimento.

e) Continuação do desenvolvimento dos algoritmos e integração com o ambiente de simulação

f) Testes preliminares do sistema e ajustes conforme necessário.

g) Finalização da implementação do sistema. Preparação para a aplicação em ambiente industrial real

h) Testes do sistema em ambiente industrial simulado (teste em ambiente simulado), Análise dos resultados dos testes e ajustes finais no sistema

i) Aplicação do sistema em ambiente industrial real na empresa selecionada, Aquisição de dados e avaliação do desempenho do sistema em condições reais

j) Análise dos resultados e preparação da apresentação final do projeto

k) Preparação e entrega da documentação final do projeto.
l) Elaboração do Relatório

1.2.1 Cronograma de atividades:

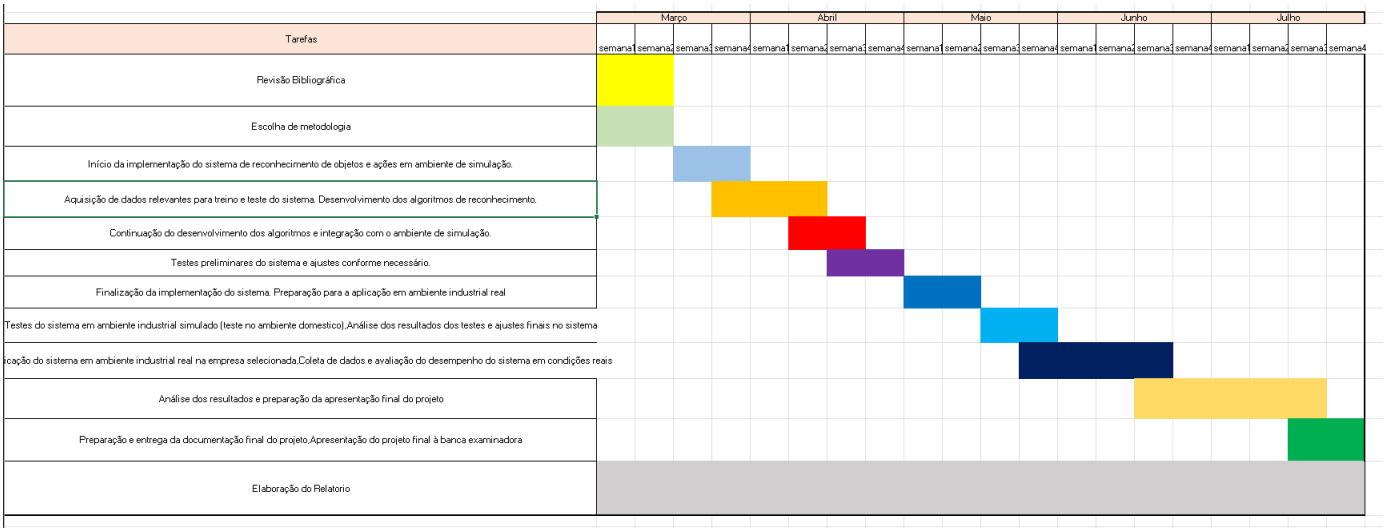

Tabela 1 - Calendarização das tarefas

2 Estado da arte

Neste capítulo são analisadas as soluções já existentes no mercado relativamente à proposta apresentada, de forma a perceber de que forma este projeto se destaca das soluções existentes no mercado.

A deteção de objetos e ações humanas em ambientes industriais é uma tarefa crucial para a automação, segurança e monitorização de processos. Nos últimos anos, o avanço das técnicas de visão computacional, impulsionado por redes neuronais convulsionais (CNNs), tem permitido a criação de sistemas mais precisos e eficientes. Entre as abordagens mais promissoras, destaca-se o YOLO [4], uma técnica de deteção em tempo real que evoluiu significativamente até a versão YOLOv8, que combina alta precisão com desempenho em tempo real. Neste contexto, a utilização de YOLOv8 em sistemas programados em *Python* tem se tornado uma prática comum para deteção e monitorização no setor industrial.

Aplicações Industriais

No ambiente industrial, a deteção de objetos e ações humanas pode ser aplicada em várias áreas, como:

- **Segurança no Trabalho**: Monitorização de atividades humanas para detetar comportamentos inseguros ou a presença em áreas restritas.
- **Automação de Processos**: Identificação de peças e objetos para automatizar linhas de produção, auxiliando na manipulação robótica.
- **Inspeção de Qualidade**: Verificação automatizada de produtos para deteção de defeitos ou anomalias.
- **Monitorização de Máquinas**: Deteção de falhas em equipamentos através da observação de padrões de movimento ou posicionamento incorreto.

2.1 Empresas Pioneiras

2.1.1 Cognex Corporation

A Cognex Corporation [5] é uma das líderes no desenvolvimento de sistemas de visão artificial para automação industrial. A empresa oferece soluções de *machine vision* e leitura de códigos de barras, usadas principalmente em linhas de montagem para detenção e inspeção de objetos. Seus produtos são amplamente aplicados em diversas indústrias, como automotiva, eletrônica, alimentos e bebidas.

- **Produtos Relevantes**:
    - **Cognex In-Sight**: Esta série oferece sistemas de visão que realizam inspeção, medição e orientação de peças em tempo real, utilizando IA para a deteção de padrões.
    - **Cognex Deep Learning**: Soluções baseadas em IA para inspeção de qualidade, incluindo a deteção de defeitos e discrepâncias na linha de montagem.
- **Tecnologias**:
    - Visão Computacional com uso de câmaras de alta-definição.
    - Algoritmos de *machine learning* que aprendem padrões de objetos e inspecionam de acordo com padrões estabelecidos.

2. Keyence

A **Keyence** [6] é outra empresa renomada na área de automação industrial e visão artificial. Seus sistemas de visão e sensores são usados para inspeção e controle de qualidade em tempo real. Eles fornecem tanto hardware como software, voltados para diversas aplicações industriais.

- **Produtos Relevantes**:
    - **Sistema de Visão Artificial CV-X Series**: Uma plataforma de alta performance que usa IA para análise e deteção em tempo real de defeitos, orientação de peças e contagem.
    - **Sensores de visão artificial IV2**: Sensores de fácil integração que realizam deteção de objetos, reconhecendo características como formas, tamanhos e padrões de cores.

- **Tecnologias**:
    - *Deep learning* para melhorar a precisão da inspeção.
    - Integração com sistemas de robótica para automação de linhas de montagem.

2.1.2 Siemens

A **Siemens** [7], uma gigante no setor de automação industrial, oferece soluções de visão artificial para controle de qualidade e inspeção de objetos em linha de montagem. Com sua plataforma de automação **SIMATIC** e soluções baseadas em IA, a Siemens permite a inspeção visual automatizada, reconhecimento de objetos e deteção de defeitos com precisão.

- **Produtos Relevantes**:
    - **SIMATIC MV500**: Equipamento de visão industrial para leitura e verificação de códigos, além da deteção de objetos e inspeção visual.
    - **MindSphere**: Plataforma de IoT industrial da Siemens que pode integrar dados de visão computacional em tempo real com análise preditiva.

- **Tecnologias**:
    - Visão computacional e *machine learning* aplicados à robótica e automação.
    - Soluções de IoT para conectar a visão computacional à análise de dados em nuvem.

2.1.3 Omron

A **Omron** [8] é uma empresa de automação que oferece soluções de visão computacional altamente confiáveis para inspeção e controle de qualidade em linhas de produção. A empresa se concentra em sistemas integrados que realizam tarefas de deteção de defeitos, orientação de robôs e verificação de componentes.

- **Produtos Relevantes**:

    - **Omron FH Vision System**: Sistema de visão que permite alta velocidade e precisão na inspeção de peças, utilizando IA e *machine learning* para detetar objetos e características específicas.
    - **Sensores de Visão FHV7**: Sensores compactos com algoritmos de aprendizagem de máquinas para deteção de peças em tempo real.

- **Tecnologias**:

    - Integração com sistemas robóticos para automação.
    - Algoritmos de deteção e reconhecimento baseados em IA.

2.1.4 Intel® RealSense

A linha Intel RealSense [9] oferece soluções de deteção e mapeamento 3D que podem ser aplicadas em sistemas industriais para a deteção de objetos em tempo real. Essas câmaras são usadas em robótica industrial para navegação autônoma e em sistemas de visão para identificar e localizar objetos em ambientes dinâmicos.

- **Produtos Relevantes**:

    - **Intel RealSense Depth Cameras**: Câmaras que oferecem mapeamento de profundidade 3D, usadas para tarefas como a deteção de objetos, navegação autônoma e interação robótica.

- **Tecnologias**:

    - Processamento de imagem 3D e profundidade.
    - IA e aprendizagem de máquina para análise e classificação de objetos.

2.1.5 Amazon Web Services (AWS) - Amazon Lookout for Vision

A AWS [10] oferece um serviço baseado em nuvem chamado **Amazon Lookout for Vision**, que permite a deteção automática de anomalias em produtos fabricados utilizando IA. Este serviço é voltado para empresas que desejam automatizar a inspeção de qualidade em suas linhas de produção.

- **Características**:
    - Usando visão computacional e *machine learning*, o serviço é capaz de identificar defeitos em objetos e componentes.
    - Pode ser integrado com câmaras de visão já existentes e sistemas de controle industrial.

- **Tecnologias**:
    - IA na nuvem, treinada para detetar padrões de defeitos e irregularidades.
    - Aprendizagem contínua com base em novos dados para otimização de deteção.

2.1.6 Google Cloud - Vision AI

O Google Cloud [11] oferece a Vision AI,[12] uma plataforma baseada em inteligência artificial que pode ser utilizada para inspeção de produtos em tempo real. A ferramenta é usada em uma variedade de setores, incluindo manufatura, para detetar objetos, identificar defeitos e garantir a qualidade.

- **Produtos Relevantes**:
    - **Google Vision AI**: API de visão artificial para reconhecimento de objetos, deteção de padrões e classificação em tempo real.

- **Tecnologias**:

    - Processamento de imagens em tempo real.
    - Integração com sistemas de IoT para controle de qualidade automatiza

Sistemas como a MindSphere e os Sensores de visão artificial IV2 apresentam um sistema equiparável ao sistema que foi desenvolvido. Abaixo estão alguns aspetos:

- Código em Python
- Processamento de imagens em tempo real
- Algoritmos de *machine learning* que aprendem padrões de objetos e inspecionam de acordo com padrões estabelecidos.

3. Tecnologias

Depois de analisados os marcos teóricos e metodologia referida, foram identificadas diversas tecnologias que se devem ter em conta durante o processo de desenvolvimento.

3.1 Software

3.1.1 Linguagem de programação escolhida

Python é uma linguagem de programação de alto nível ou *High Level Language*, dinâmica, interpretada, modular, multiplataforma e orientada a objetos, uma forma específica de organizar softwares, os procedimentos estão submetidos às classes, o que possibilita maior controle e estabilidade de códigos para projetos de grandes proporções. Por ser uma linguagem de sintaxe relativamente simples e de fácil compreensão, ganhou popularidade entre profissionais da indústria tecnológica que não são especificamente programadores, como engenheiros, matemáticos e cientista.

Um dos seus maiores atrativos é possuir um grande número de bibliotecas nativas e de terceiros, tornando-a muito difundida e útil numa grande variedade de setores dentro do desenvolvimento da web, e também em áreas como análise de dados, *machine learning* e IA.

3.1.2 Roboflow

Roboflow [13] é uma plataforma abrangente que facilita o processo de construção e implantação de modelos de visão computacional. A seguir, é apresentada uma descrição detalhada dos passos realizados para construir e implantar modelos utilizando o Roboflow.

1. Introdução à Visão Computacional e Roboflow

Visão Computacional é um campo da inteligência artificial (IA) que permite que máquinas interpretem e compreendam imagens digitais e vídeos. Através do uso de modelos de aprendizagem profunda, as máquinas podem identificar e classificar objetos de forma precisa. Roboflow é uma plataforma que suporta todo o ciclo de vida de um projeto de visão artificial, desde o gerenciamento de dados até a implantação de modelos, com uma interface que não exige ou exige pouco código, facilitando o desenvolvimento e a implantação de modelos.

2. Funcionalidades do Roboflow

A plataforma Roboflow oferece várias funcionalidades que facilitam o fluxo de trabalho em visão computacional:

- **Gestão de Conjuntos de Dados**: Roboflow permite importar, organizar e anotar conjuntos de dados de maneira eficiente. É possível fazer upload de imagens e criar conjuntos de dados com diferentes tipos de anotações, como caixas delimitadoras, polígonos e máscaras de segmentação.

- **Aumento de Dados**: Ferramentas para aumento de dados permitem criar exemplos de treino a partir dos dados existentes, aplicando técnicas como rotação, inversão e redimensionamento, melhorando a robustez e a precisão do modelo.

- **Ferramentas de Anotação**: As ferramentas de anotação integradas permitem uma rotulagem precisa dos dados, com suporte para anotação colaborativa.

- **Pré-processamento e Versionamento**: Roboflow permite pré-processar dados diretamente na plataforma, aplicando filtros, redimensionamento de imagens ou normalização de valores de pixels, o que facilita a manutenção de diferentes versões do conjunto de dados.

- **Treino de Modelos**: Integrações com frameworks de *machine learning* como TensorFlow, PyTorch e YOLO permitem o treino de modelos diretamente na plataforma ou a exportação de dados para treino externo.

- **Implantação de Modelos**: Ferramentas para a implantação de modelos na nuvem, no local ou em dispositivos de borda estão disponíveis, juntamente com APIs para integração dos modelos com outras aplicações e serviços.

- **Avaliação e Monitorização de Modelos**: A plataforma oferece métricas para avaliar o desempenho dos modelos, como precisão, *recall*, pontuação F1 e mAP, além de ferramentas para monitorizar modelos em produção.

3. Processo de Uso do Roboflow

Passo 1: Criação de Conta no Roboflow

Foi criada uma conta no Roboflow para acessar as funcionalidades da plataforma.

Passo 2: Importação de Conjunto de Dados

Imagens foram carregadas na plataforma Roboflow, utilizando armazenamento local e URLs, suportando formatos como JPEG, PNG e BMP. Anotações existentes foram importadas em formatos compatíveis, como COCO, Pascal VOC e YOLO.

Passo 3: Anotação dos Dados

Utilizou-se as ferramentas de anotação do Roboflow para etiquetar as imagens, desenhando caixas delimitadoras (*bounding boxes*), polígonos ou máscaras de segmentação ao redor dos objetos de interesse. Anotação colaborativa foi utilizada para permitir que múltiplos usuários trabalhassem simultaneamente no mesmo conjunto de dados.

Passo 4: Aumento e Pré-processamento dos Dados

Diversas técnicas de aumento de dados foram aplicadas para aumentar a diversidade dos dados de treino. Em seguida, foi realizado o pré-processamento das imagens, redimensionando-as e normalizando os valores dos pixels para padronizar os dados de entrada.

Passo 5: Treino do Modelo

Foi selecionada uma estrutura de modelo compatível, como TensorFlow, PyTorch ou YOLO, para exportação dos conjuntos de dados e treino do modelo. Optou-se por treinar o modelo utilizando o serviço integrado do Roboflow, utilizando código inicial e notebooks fornecidos pela plataforma.

Passo 6: Avaliação do Modelo

Após o treino, o desempenho do modelo foi avaliado utilizando métricas fornecidas pelo Roboflow, como precisão, recall e F1 score, para identificar áreas de melhoria.

Passo 7: Implantação do Modelo

Foram exploradas as opções de implantação disponíveis no Roboflow, incluindo a implantação na nuvem, no local e em dispositivos de borda, como NVIDIA Jetson ou Raspberry Pi. APIs REST e SDKs foram utilizados para integrar o modelo com outros aplicativos e serviços.

Passo 8: Monitorização e Atualização do Modelo

O desempenho do modelo foi monitorizado continuamente em produção utilizando as ferramentas de monitorização do Roboflow, garantindo um bom desempenho ao longo do tempo. À medida que mais dados foram coletados e o desempenho do modelo mudou, o modelo foi treinado e atualizado conforme necessário, utilizando o sistema de versionamento do Roboflow para gerenciar diferentes versões do conjunto de dados e do modelo.

4. Recursos Avançados e Integrações

Além das funcionalidades básicas, Roboflow oferece recursos avançados como fluxos de trabalho personalizados, aprendizagem ativa, ensembles de modelos e integrações com ferramentas e plataformas de terceiros, como AWS, Google Cloud, Azure e dispositivos de borda, facilitando a implantação e de modelos de visão computacional.

3.1.3 Pycharm

PyCharm [14] é um Ambiente de Desenvolvimento Integrado (IDE) usado para programar em Python.

Ele fornece análise de código, um depurador gráfico, um ambiente de teste de unidade integrado, integração com sistemas de controle de versão, e suporta desenvolvimento web com Django. PyCharm é desenvolvido pela empresa checa JetBrains.

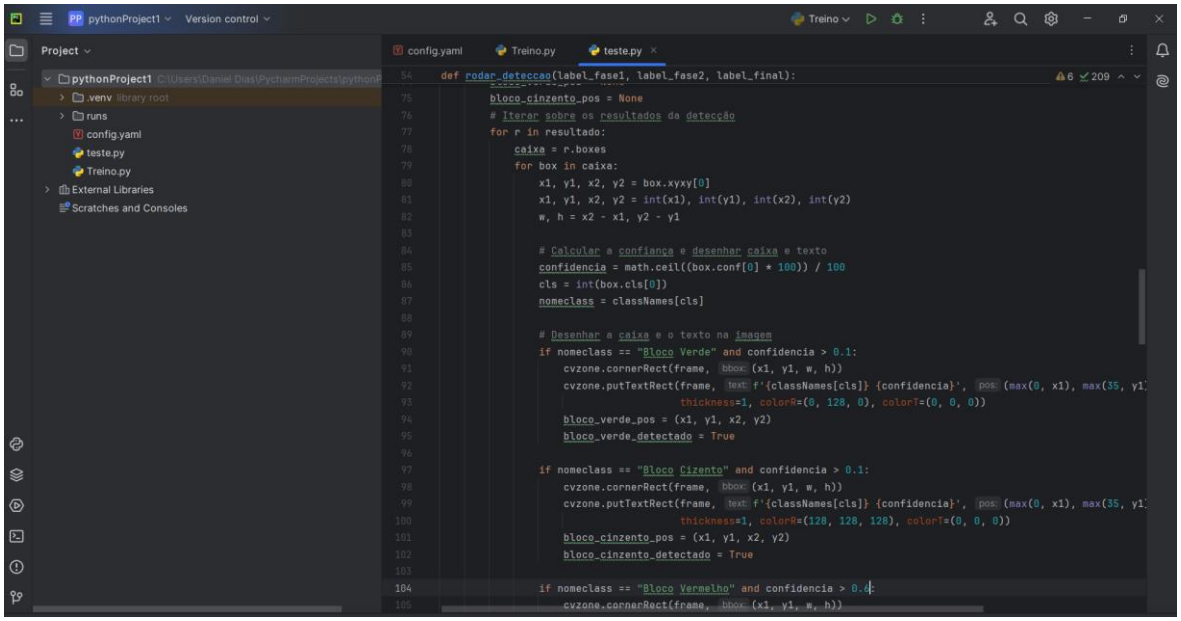

Figura 1 - Pycharm IDE

3.1.4 OpenCV

OpenCV (Open Source Computer Vision Library), originalmente, desenvolvida pela Intel, em 2000, é uma biblioteca multiplataforma, totalmente livre ao uso acadêmico e comercial, para o desenvolvimento de aplicativos na área de Visão computacional, bastando seguir o modelo de licença BSD Intel.

O OpenCV possui módulos de Processamento de Imagens e Video I/O, Estrutura de dados, Álgebra Linear, GUI (Interface Gráfica do Utilizador) Básica com sistema de janelas independentes, Controle de rato e teclado, além de mais de 350 algoritmos de Visão computacional como: Filtros de imagem, calibração da câmara, reconhecimento de objetos, análise estrutural e outros. O seu processamento é em tempo real de imagens.

A biblioteca possui mais de 2.500 algoritmos otimizados, que abrangem um volumoso conjunto de algoritmos de visão computacional e *machine learning* clássicos e de última geração.

Estes algoritmos podem ser usados para detetar e reconhecer rostos, identificar objetos, classificar ações humanas em vídeos, rastrear movimentos de câmaras, rastrear objetos em movimento, extrair modelos 3D de objetos, produzir nuvens de pontos 3D de câmaras estéreo, unir imagens para produzir uma alta resolução imagem de uma cena inteira, encontrar imagens semelhantes de uma base de dados de imagens, remover olhos vermelhos de imagens tiradas com flash, seguir movimentos oculares, reconhecer cenários e estabelecer marcadores para sobrepô-los com realidade aumentada, etc

Junto com empresas bem instituídas como a Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota que aplicam a biblioteca, existem muitas *startups* como Applied Minds, VideoSurf e Zeitera, que fazem uso extensivo do OpenCV. Os usos implantados do OpenCV abrangem desde a união de imagens de vista da rua, deteção de intrusões em vídeo de vigilância em Israel, monitorização de equipamentos de minas na China, ajuda a robôs a navegar e pegar objetos na Willow Garage, deteção de acidentes de afogamento em piscinas na Europa, execução de arte interativa em Espanha e Nova York, verificação de pistas à procura de detritos na Turquia, inspeção de rótulos de produtos em fábricas ao redor do mundo para deteção rápida de rostos no Japão .

Possui interfaces C++, Python, Java e MATLAB e suporta Windows, Linux, Android e Mac OS. O OpenCV inclina-se principalmente para aplicações de visão em tempo real e aproveita as instruções MMX e SSE quando disponíveis. Uma interface CUDA e OpenCL com todos os recursos está a ser desenvolvida ativamente na atualidade. Existem mais de 500 algoritmos e cerca de 10 vezes mais funções que compõem ou suportam estes algoritmos. O OpenCV é escrito nativamente em C++ e possui uma interface de modelo que funciona perfeitamente com containers STL

3.1.6 YOLOv8

A tecnologia YOLOv8 é um modelo avançado de deteção de objetos em tempo real, amplamente utilizado em aplicações de visão computacional. Desenvolvido com base no conceito de "You Only Look Once" (Você só olha uma vez), o YOLOv8 tem como objetivo fornecer deteção rápida e precisa de objetos em imagens e vídeos. Uma das principais características do YOLOv8 é sua capacidade de deteção em tempo real, permitindo que a análise de objetos seja feita de forma rápida e eficiente.

Isso é possível graças à sua arquitetura de rede neural, que utiliza técnicas avançadas de processamento paralelo e otimização de cálculos, resultando em tempos de resposta significativamente menores em comparação com outros modelos de deteção de objetos. O YOLOv8 também se destaca pela sua capacidade de detetar múltiplos objetos em uma única imagem ou quadro de vídeo.

Ao contrário de abordagens baseadas em janelas deslizantes, que exigem várias passagens pela imagem, o YOLOv8 realiza a deteção de objetos em uma única etapa, identificando simultaneamente as classes e as localizações dos objetos presentes. Outra vantagem do YOLOv8 é sua precisão.

Graças à utilização de redes neurais convulsionais profundas, combinadas com técnicas avançadas de treino e otimização, o modelo alcança resultados notáveis em termos de acurácia na deteção de objetos, mesmo em condições desafiadoras, como objetos pequenos, sobreposição ou baixa resolução.

Além disso, o YOLOv8 é altamente flexível e personalizável. É possível treinar o modelo para detetar objetos específicos em diferentes domínios, ajustando-o de acordo com as necessidades do projeto. Isso o torna uma escolha popular em diversas áreas, como vigilância por vídeo, segurança, automação industrial, veículos autônomos e muito mais. Em resumo, o YOLOv8 é uma tecnologia de deteção de objetos em tempo real que se destaca pela sua velocidade, precisão e capacidade de detetar múltiplos objetos em uma única passagem. Com sua arquitetura avançada e flexibilidade, o YOLOv8 se tornou uma ferramenta essencial em aplicações de visão computacional que exigem análise rápida e confiável de objetos em tempo real.

3.1.6 CVZone

A tecnologia CVZone [15] é uma biblioteca Python projetada para simplificar o desenvolvimento de aplicações de visão computacional. Essa biblioteca oferece uma ampla gama de recursos e funcionalidades que permitem aos desenvolvedores criar soluções avançadas de processamento de imagem e reconhecimento visual. CVZone é baseado na biblioteca OpenCV (Open Source Computer Vision Library) e oferece um conjunto de ferramentas e utilitários que facilitam a extração de recursos visuais, deteção de objetos, rastreamento de movimento e muito mais. Essa tecnologia é particularmente útil em projetos que envolvem análise de vídeo, deteção de rostos, reconhecimento de padrões e segmentação de imagens. Uma das principais características da CVZone é sua abordagem de código aberto e documentação detalhada. Essa combinação permite aos desenvolvedores aproveitar ao máximo as funcionalidades da biblioteca e adaptá-la às suas necessidades específicas. Além disso, a CVZone oferece uma ampla gama de exemplos e tutoriais, tornando o processo de aprendizagem mais acessível.

3.2.7 Threading

A tecnologia Threading é uma funcionalidade fundamental do Python que permite a execução de múltiplas tarefas simultaneamente, também conhecido como programação concorrente. Ela permite que um programa seja dividido em threads, que são fluxos independentes de execução que podem ser executados de forma paralela.

A principal vantagem do uso de *threading* é a capacidade de melhorar o desempenho e a eficiência do programa, especialmente em situações em que existem tarefas que podem ser executadas em paralelo. Por exemplo, em programas que envolvem cálculos intensivos ou operações de entrada e saída, a utilização de *threads* pode ajudar a reduzir o tempo de execução total do programa.

A biblioteca *threading* do Python oferece uma série de recursos para criar e controlar *threads*. Os desenvolvedores podem criar *threads* personalizados e atribuir lhes tarefas específicas para executar em paralelo. Além disso, a biblioteca fornece métodos para sincronizar a execução das *threads*, permitindo que elas se comuniquem entre si e coordenem suas atividades.

No entanto, é importante ressaltar que o uso incorreto da tecnologia *threading* pode levar a problemas, como condições de corrida (quando várias *threads* tentam ter acesso e modificar os mesmos recursos ao mesmo tempo) e *deadlocks* (quando duas ou mais *threads* ficam bloqueadas permanentemente, aguardando uma pela outra). Portanto, é fundamental que os desenvolvedores tenham cuidado ao utilizar *threading* e apliquem técnicas adequadas para evitar esses problemas.

Além disso, a biblioteca *threading* também fornece recursos para a implementação de mecanismos de controle, como semáforos filas, que ajudam a garantir a exclusão mútua e a sincronização correta entre as *threads*.

4. Fases da Modelação

Nesta parte do relatório, vai ser apresentado o processo detalhadamente como foi desenvolvido o sistema em Python, as várias fases e métodos do código, a etiquetagem e o treino das imagens necessárias para os testes, juntamente com a análise de todas as fases da modelação e todas as melhorias que foram meticulosamente realizadas.

4.1 Primeira Fase

Nesta primeira fase ,foi feito um sistema no Pycharm de forma a ser possível fazer treino das imagens que teriam de ser detetadas para o teste ,das quais o *labelling* das mesmas foi feito no software Roboflow.Nesta primeira fase foram tiradas cerca de 850 fotos de 3 objetos distintos de vários ângulos e ambientes diferentes .

Após tiradas estas fotos e feita a etiquetagem com 3 classes (uma classe para cada objeto) .Esta fase tem como objetivo conseguir simplesmente detetar os objetos com um nível de confiança aceitável em tempo real.

Para tal, foram instaladas as bibliotecas e plugins necessários para a modelação para este tipo de sistemas, como o OpenCV e ultralytics. De seguida vai ser apresentada toda a modelação do processo que levou á elaboração do programa.

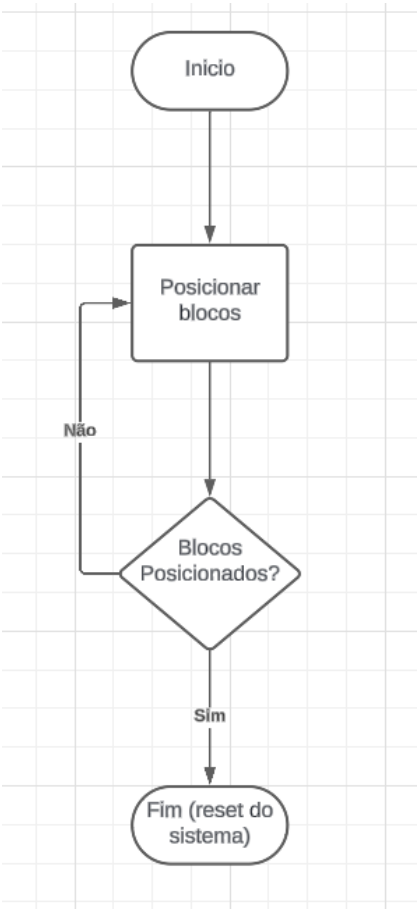

Figura 2 - Fluxograma da 1º Fase

4.2 Segunda Fase

Nesta segunda fase do teste foram tiradas cerca de 550 fotografias dos objetos, juntamente com as fotos já tiradas, nas fases diferentes da montagem da peça. Foram feitas mais 3 classes em adição às classes existentes fazendo um total de 6 classes para a etiquetagem.

O programa realiza deteção de objetos em tempo real utilizando o modelo YOLOv8, monitorizando a deteção de blocos coloridos específicos em um vídeo capturado pela câmara. A interface gráfica mostra o status das diferentes fases da deteção, atualizando dinamicamente conforme o progresso.

Em caso de deteção de um bloco "falso", um alerta sonoro é acionado. Ao completar todas as fases, o sistema reinicia automaticamente após uma contagem regressiva.

Este sistema pode ser aplicado em processos de montagem automatizados ou em sistemas educacionais para monitorar o progresso de tarefas visuais complexas.

Este código utiliza a biblioteca Ultralytics YOLO, especificamente o modelo YOLOv8 (*You Only Look Once*, versão 8) para treinar uma rede neural em um conjunto de dados personalizado. Vou explicar detalhadamente cada parte do código e o que ela faz.

1. Importação da Biblioteca

```
from ultralytics import YOLO
```

Figura 3 - Importação da biblioteca

YOLO: Este comando importa a classe YOLO da biblioteca ultralytics. Esta classe é responsável por carregar, configurar, e treinar o modelo YOLO, bem como realizar inferências (predições) com ele.

2. **Carregamento do Modelo**

```
model = YOLO("yolov8n.yaml")
```

Figura 4 - Carregamento do modelo

YOLO("yolov8n.yaml"): Carrega a configuração do modelo YOLO. O arquivo "yolov8n.yaml" é um arquivo de configuração que define a arquitetura do modelo YOLOv8 na sua variante mais leve, "n" (nano), que é otimizada para eficiência e velocidade, adequada para rodar em dispositivos com baixa capacidade de processamento.

3. Treino do Modelo

```
resultados = model.train(
    data="config.yaml",
    imgsz=640,
    epochs=30,
    batch=16,
    name='yolov8n_custom'
)
```

Figura 5 -Treino do modelo

- **data= "config.yaml"**: Este parâmetro define o caminho para o arquivo de configuração dos dados, "config.yaml". Este arquivo deve conter informações sobre o conjunto de dados, como os caminhos para as imagens de treino e validação, bem como as classes que o modelo deve detetar.

- **imgsz= 640**: Este parâmetro define o tamanho das imagens usadas no treino. As imagens serão redimensionadas para 640x640 pixels antes de serem processadas pelo modelo. Este é um tamanho comumente usado, equilibrando precisão e eficiência.

- **epochs= 30**: Especifica que o modelo será treinado por 30 épocas, ou seja, o conjunto de dados será passado 30 vezes pelo modelo durante o processo de treino. Mais épocas geralmente resultam em um modelo mais preciso, mas também aumentam o tempo de treino.

- **batch= 16**: Define o tamanho do lote (batch size), que é o número de amostras que o modelo processa antes de atualizar os pesos. Um batch size de 16 significa que, a cada iteração, 16 imagens serão processadas antes de uma atualização nos pesos do modelo ser feita.

- **name= 'yolov8n_custom'**: Este parâmetro especifica o nome da sessão de treino. Todos os arquivos gerados durante o treino (como pesos salvos e logs) serão nomeados usando esse prefixo.

- **resultados**: Este objeto armazenará os resultados do processo de treino, incluindo métricas como a precisão, a loss (perda) durante o treino, e o desempenho do modelo nos dados de validação ao longo das épocas.

4.2.1 Teste do Treino

Este código integra um modelo de deteção de objetos YOLOv8 com uma interface gráfica baseada em Tkinter para monitorizar e gerenciar um processo de deteção e montagem de blocos coloridos. A seguir, está uma explicação detalhada das diferentes partes do código.

1. **Importação de Bibliotecas**

```
import time
from ultralytics import YOLO
import cv2
import math
import cvzone
from tkinter import *
import threading
import winsound
```

Figura 6 - Importação de bibliotecas

- **time**: Fornece funções relacionadas ao tempo, como pausar a execução.
- **YOLO**: Importa a classe YOLO da biblioteca Ultralytics, utilizada para carregar e executar o modelo de deteção.
- **cv2**: Parte do OpenCV, usada para manipulação de imagens e vídeo.
- **math**: Fornece funções matemáticas básicas, como arredondamento.
- **cvzone**: Fornece ferramentas de visualização simplificadas, como desenhar retângulos e textos nas imagens.
- **tkinter**: Usada para criar a interface gráfica (GUI).
- **threading**: Permite a execução de código em threads separadas, o que é útil para rodar a deteção sem bloquear a GUI.
- **winsound**: Biblioteca usada para tocar sons de alerta no Windows.

4.2.2 Teste em tempo real

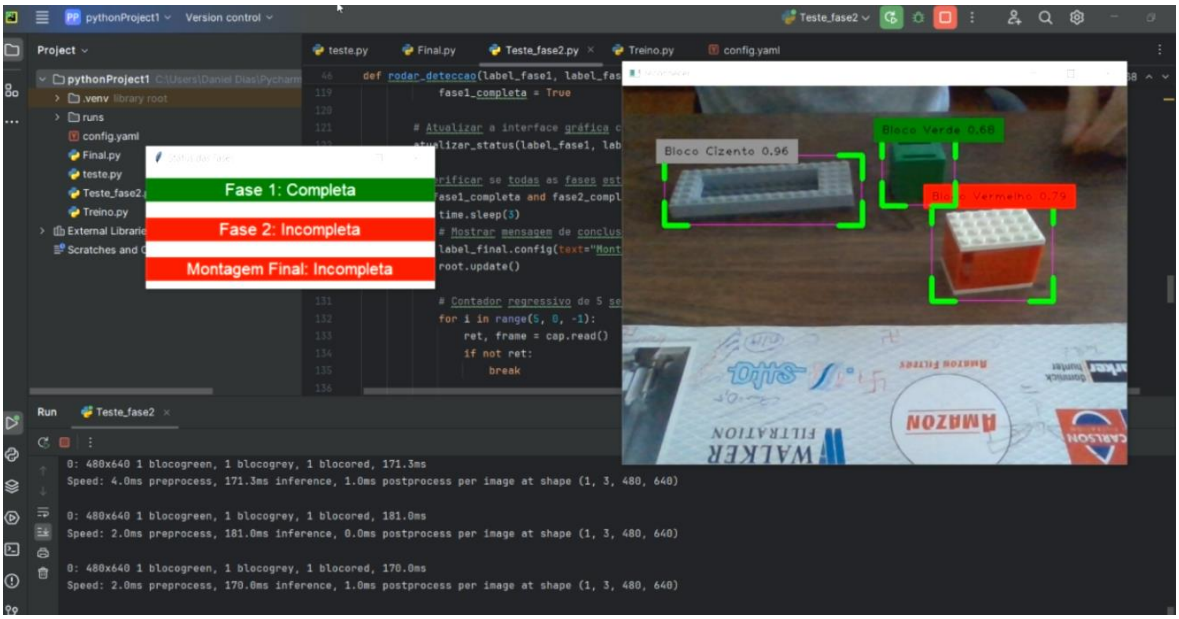

Figura 7 - Deteção em tempo real (fase 1)

Esta fase do sistema tem apenas como função garantir que os 3 objetos necessários para a montagem estão presentes na bancada. Se o sistema detetar os três, irá prosseguir para a próxima fase.

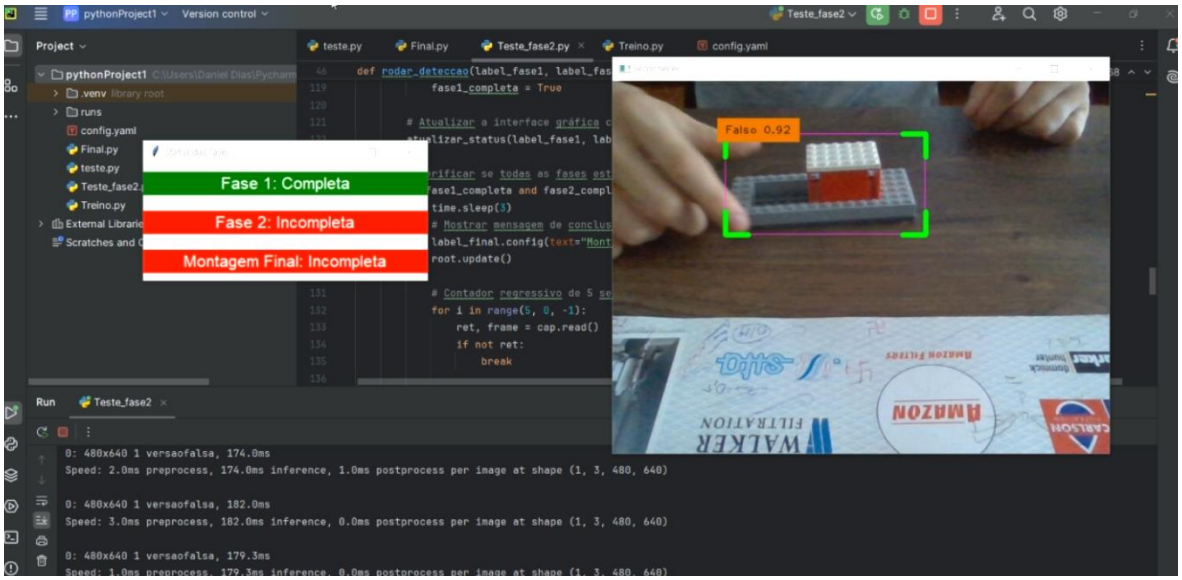

Figura 8 - Deteção em tempo real (fase1)

Se a montagem for feita da forma errada, neste caso a peça vermelha ser colocada primeiro que a Verde, o sistema não vai dar a segunda fase como completa e vai emitir um ruido de advertência.

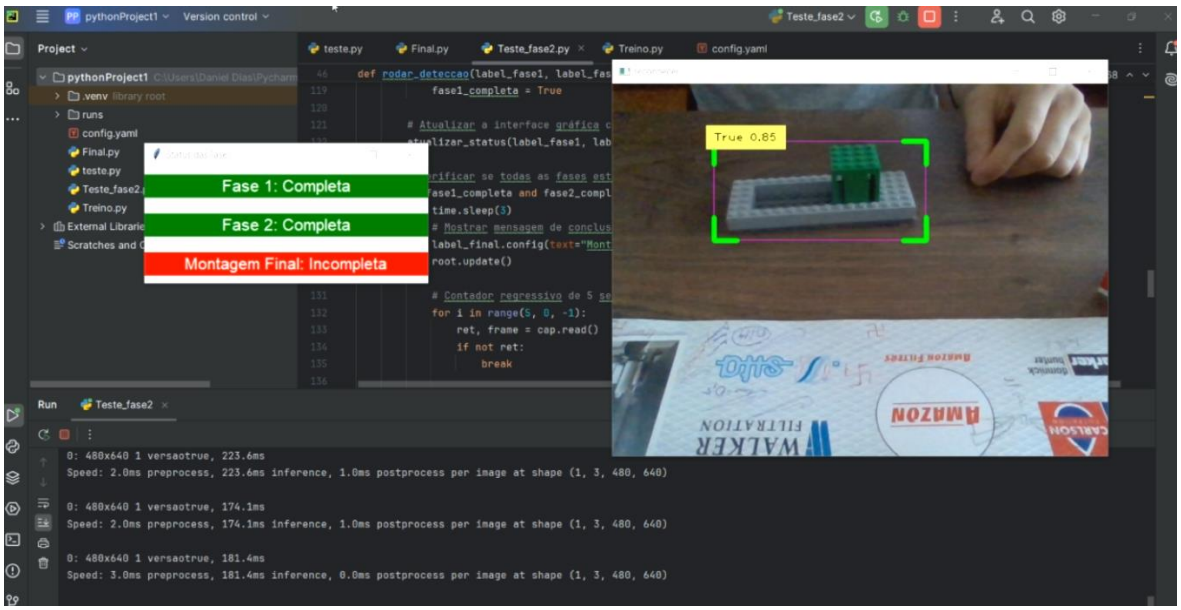

Figura 9 - Deteção em tempo real (fase 2)

Neste exemplo a montagem foi declarada como “True” visto que foi executada como se pretende, prosseguindo assim para a última fase.

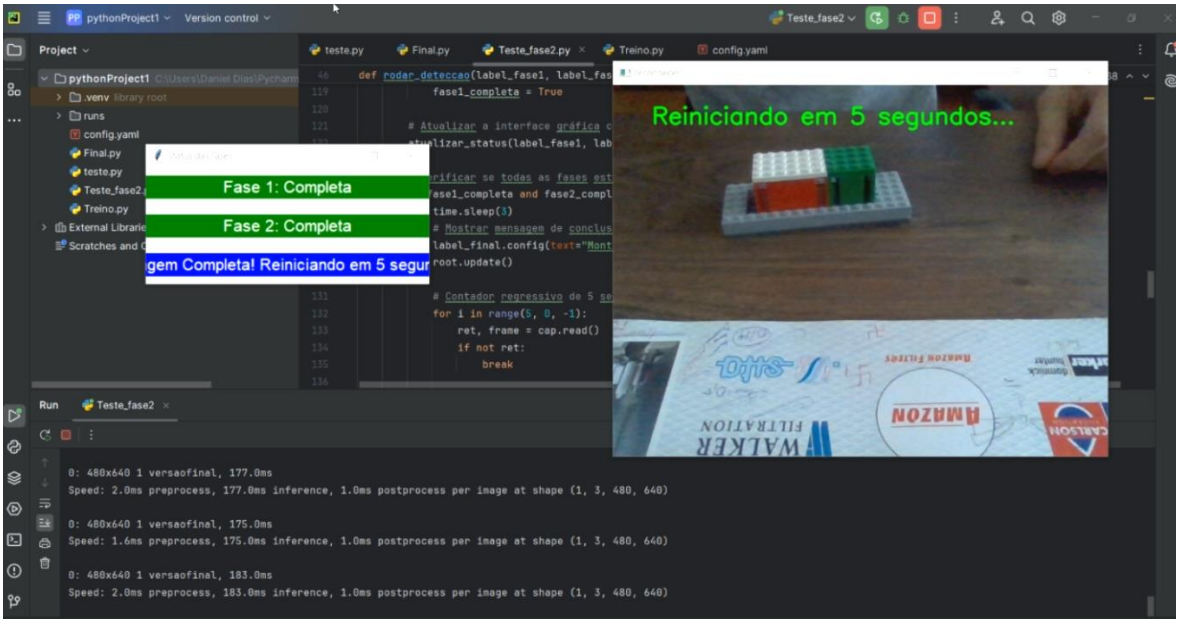

Figura 10 - Fase final (2º algoritmo)

Este exemplo demostra fase final, na qual os objetos estão montados da forma correta e a "*bounding box*" declara a montagem como final, fazendo assim o sistema reiniciar e começar novamente na 1ºFase.

4.3 Terceira Fase

Nesta fase foi adotada uma abordagem diferente e um uso de métodos diferentes. A montagem centrou-se na posição entre a relação de coordenadas x e y entre objetos.

Desta maneira conseguimos garantir que as peças ficam montadas na posição e local correto sem qualquer tipo de folga. Foram usadas 3 classes para identificar os objetos individualmente em vez da segunda fase em que são usadas 6. Desta maneira sistema não funciona por fases treinadas no na etiquetagem, mas sim por um sistema de coordenadas da *bounding box* de cada objeto ao ser detetado pela câmara.

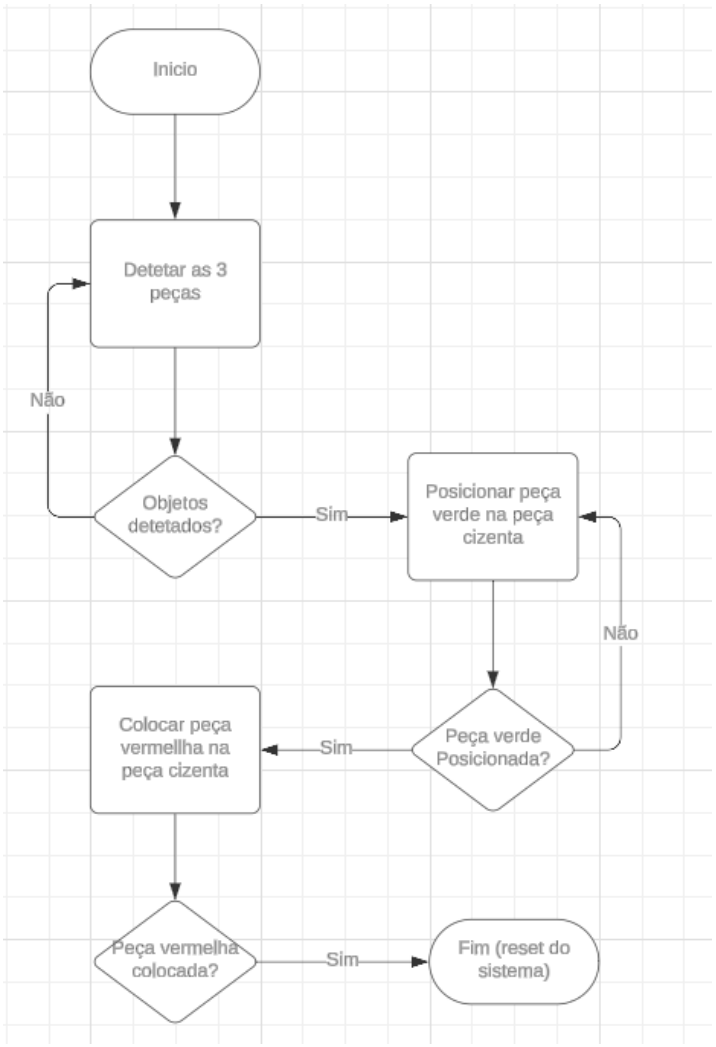

Figura 11 - Fluxograma do Terceiro Algoritmo

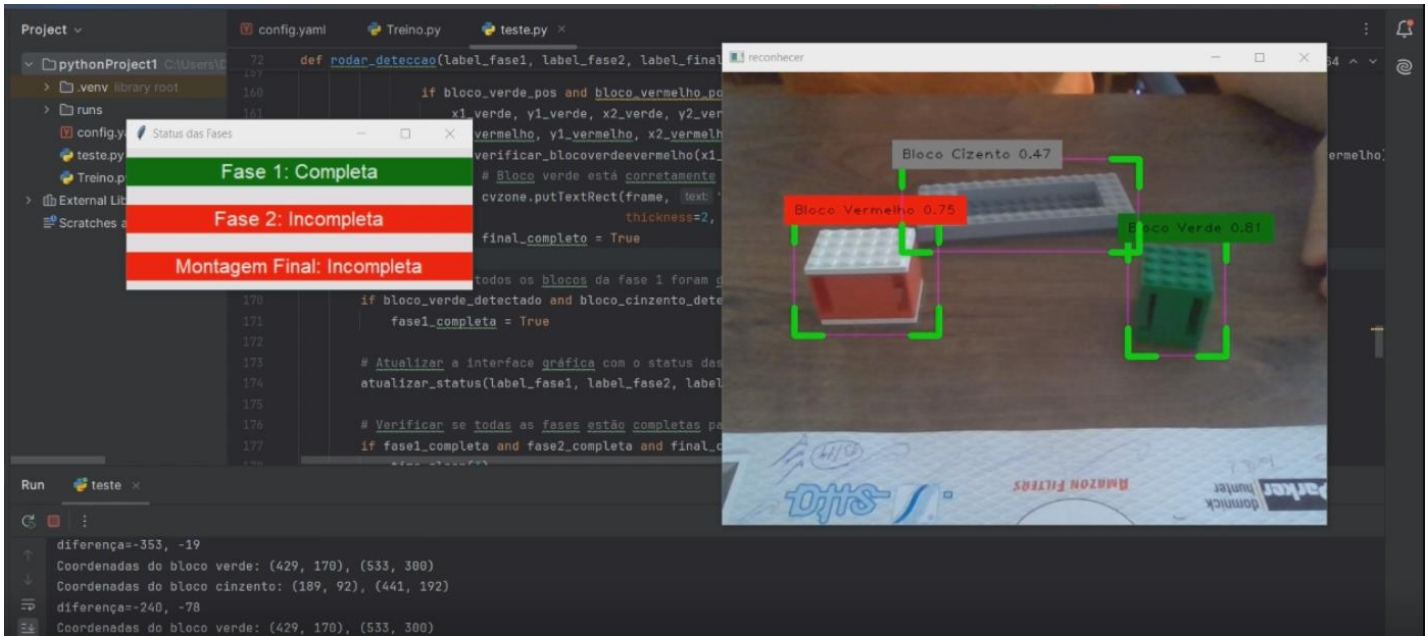

Figura 12 – Deteção em tempo real (1ºfase, 3 algoritmo)

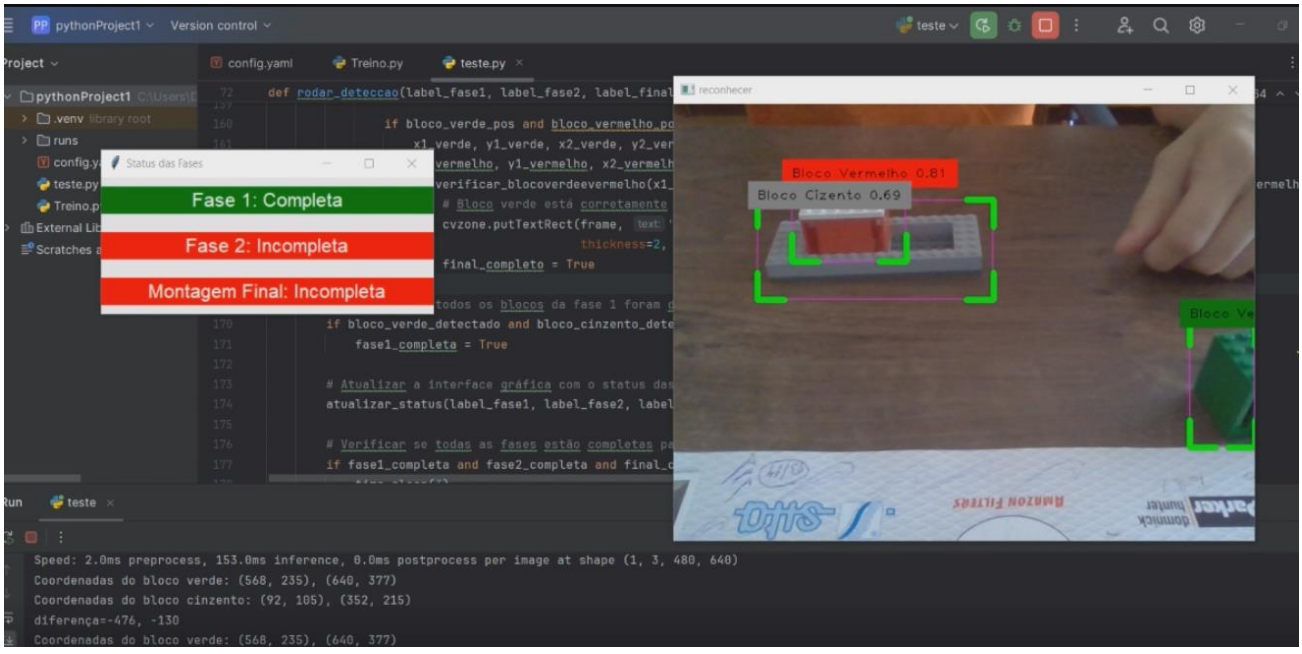

Figura 13 - Montagem feita na ordem incorreta (2º fase,3º algoritmo)

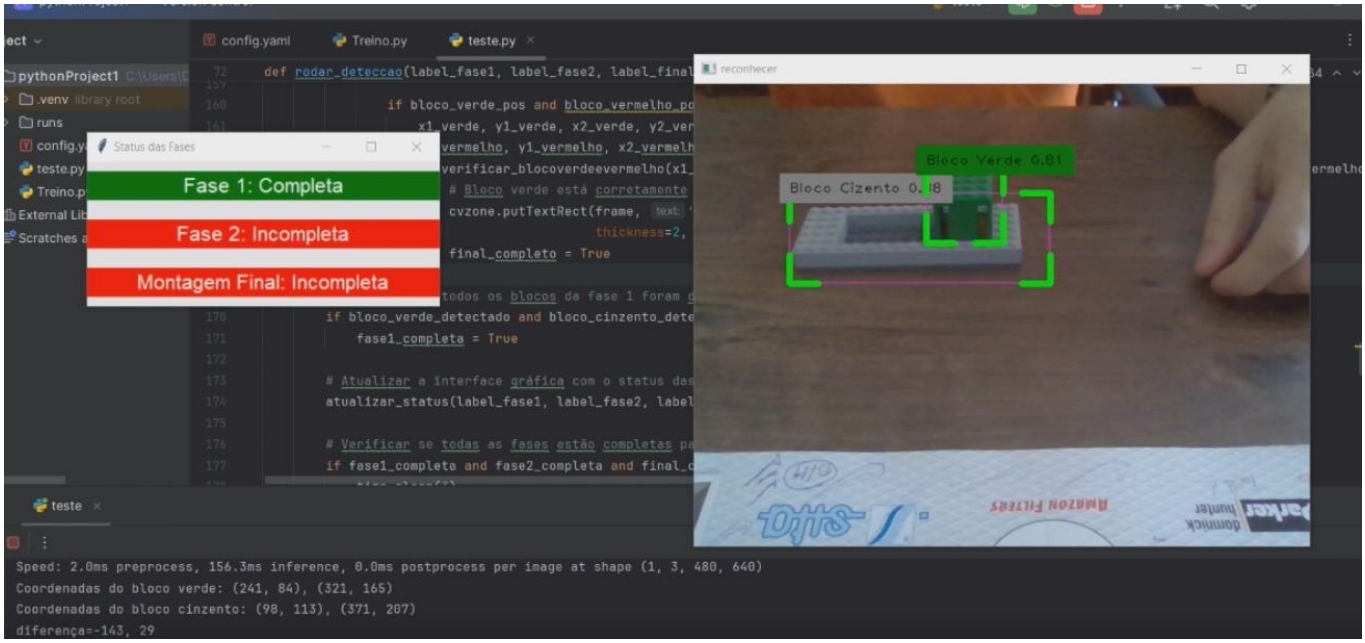

Figura 14 – Montagem feita na ordem correta (2º fase, 3º algoritmo)

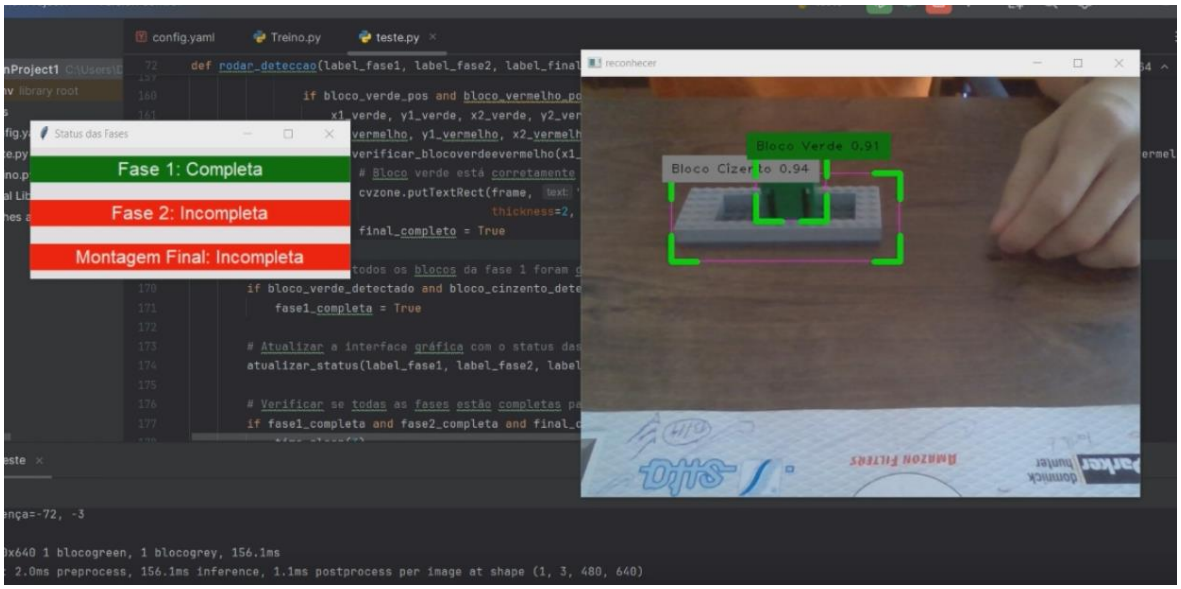

Figura 15 – Montagem feita na ordem correta (2º fase, 3ºalgoritmo)

Montagem feita da maneira correta, mas a peça não está na posição correta em relação á peça base

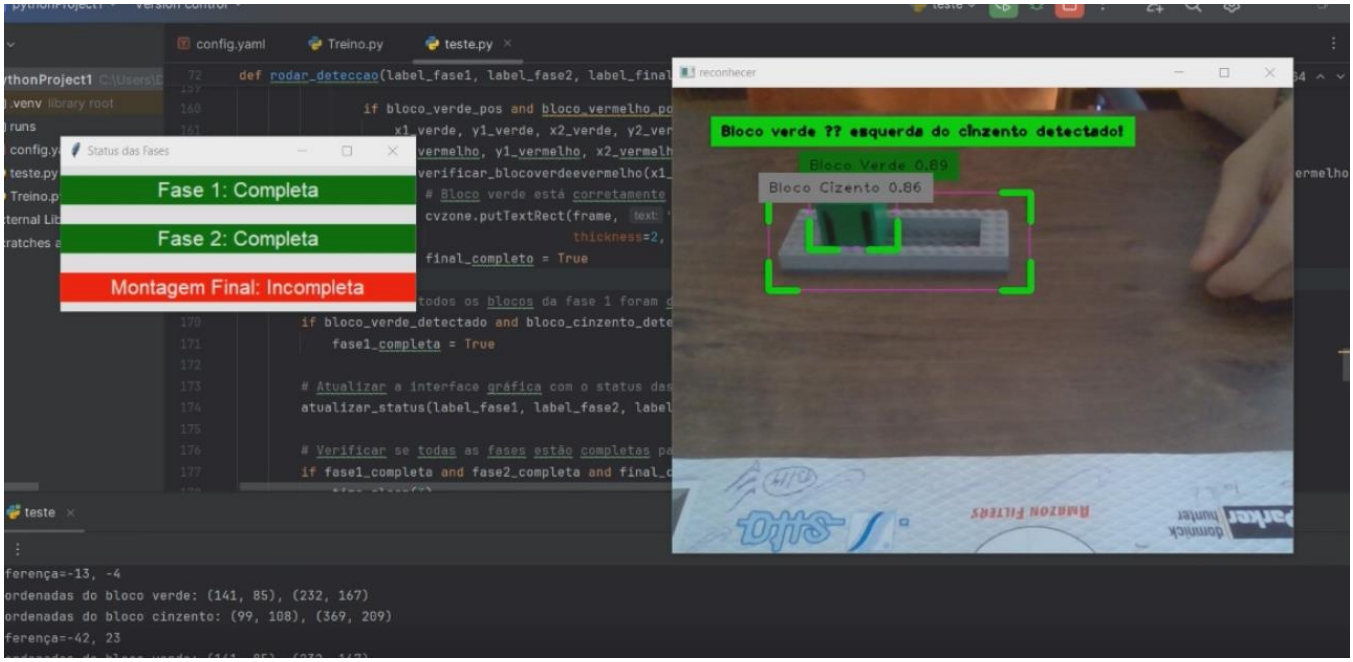

Figura 16 - Fase 2 concluída (3º algoritmo)

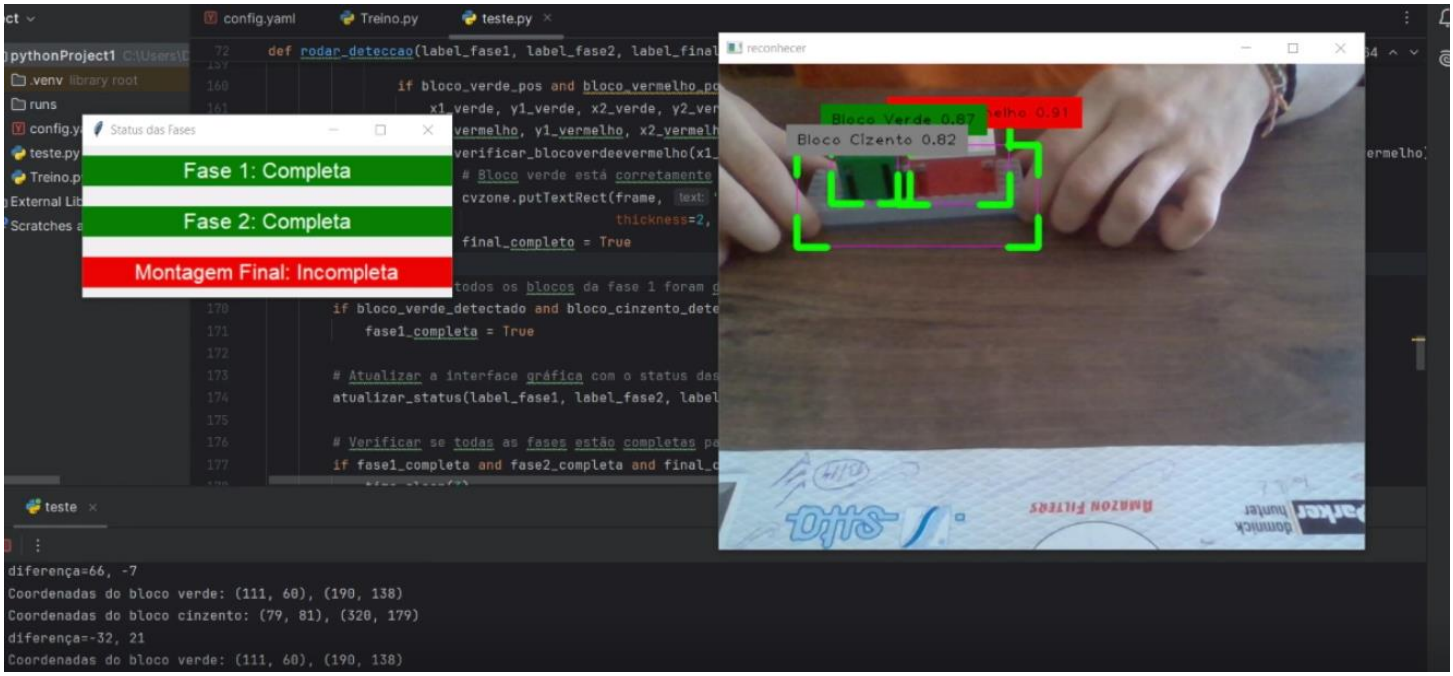

Figura 17 – Fase 2 concluída (3ºalgoritmo)

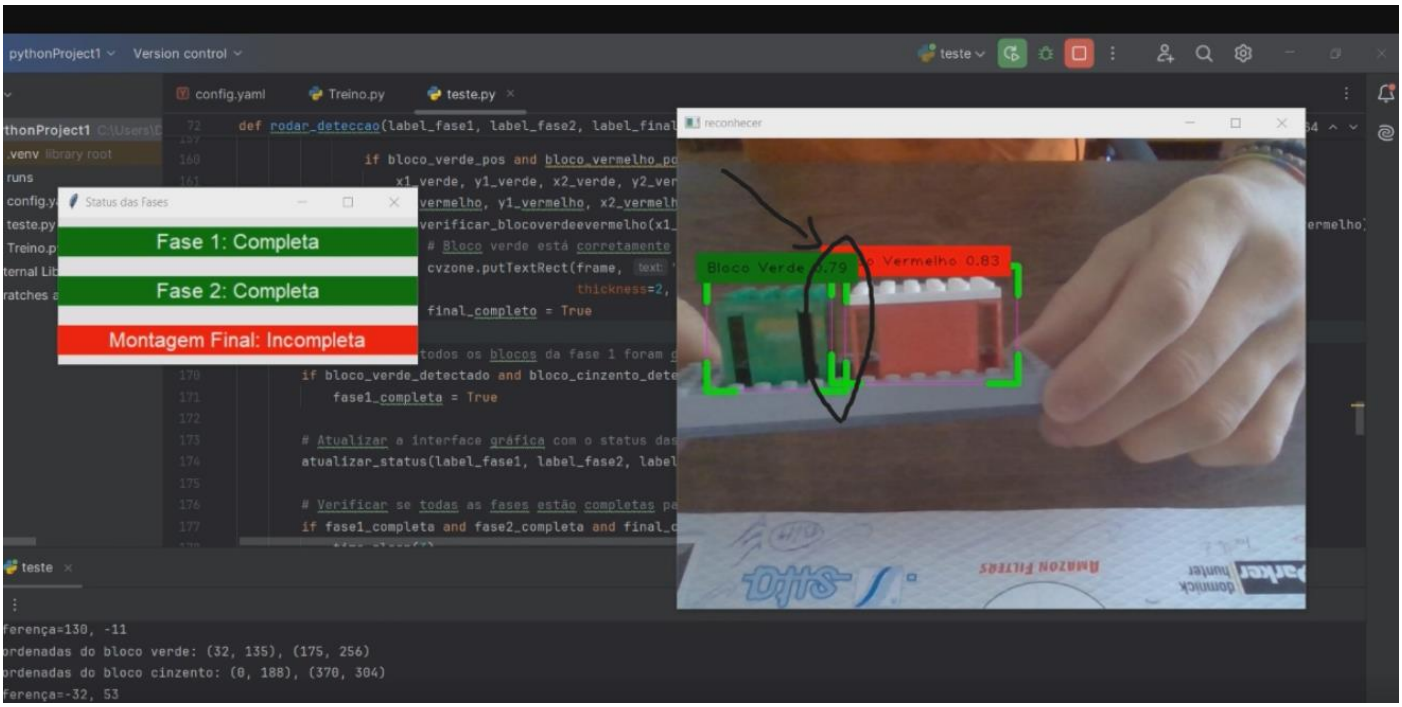

Figura 18 – folga de peças na fase 2 (3ºalgoritmo)

Conseguimos ver que a montagem final não está concluída visto que o bloco vermelho te de ser colocado exatamente ao lado do bloco verde. Na Figura acima pode se ver que existe uma folga ligeira entre os dois. A fase só irá ser concluída quando forem tomadas medidas corretivas.

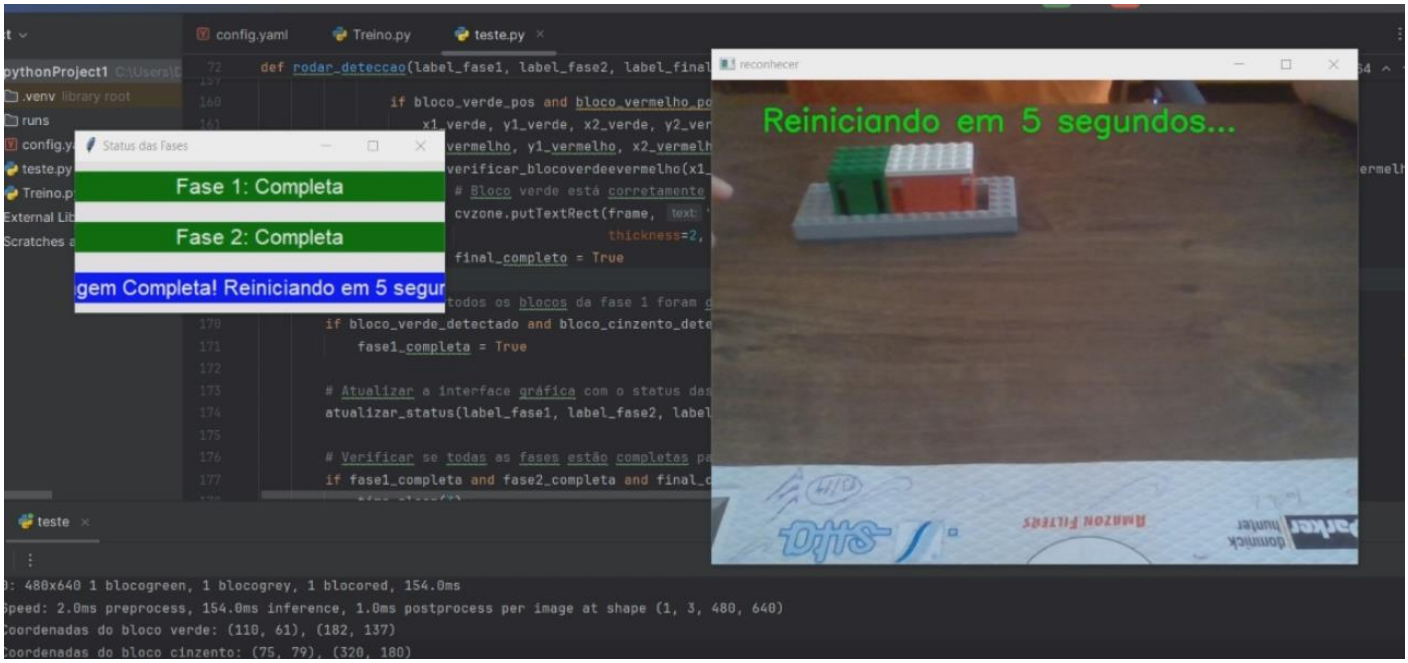

Figura 19 – Fase 3 concluída (3º concluída)

Quando as peças estão montadas da maneira correta a fase final dá por concluída e é feito um *reset* automático ao sistema em 5 segundos, voltando de volta á primeira fase.

4.4 Fase final

Nesta fase teve-se como ponto crucial cumprir a última fase do projeto de implementar um sistema de reconhecimento de objetos e de ações para sistemas robóticos, que opere num posto de trabalho, de forma que o sistema desenvolvido na UC de Projeto esteja de acordo com os parâmetros inicialmente definidos. O caso de estudo foi a empresa BITZER (Portugal) - Compressores para Frio, S.A., Na qual foram cedidas as peças de um compressor. Cedidas esta peças, foi possível tirar cerca de 1200 fotos para poder fazer a etiquetagem individual destas peças, o respetivo treino das imagens, a modelagem e alterações no código previamente desenvolvido e por em prática, em tempo real, do *tracking* dos objetos numa tarefa de montagem na linha de produção da própria empresa. Este sistema provasse crucial e uma boa maneira de garantir que a montagem é feita da maneira correta e o mais precisa possível.

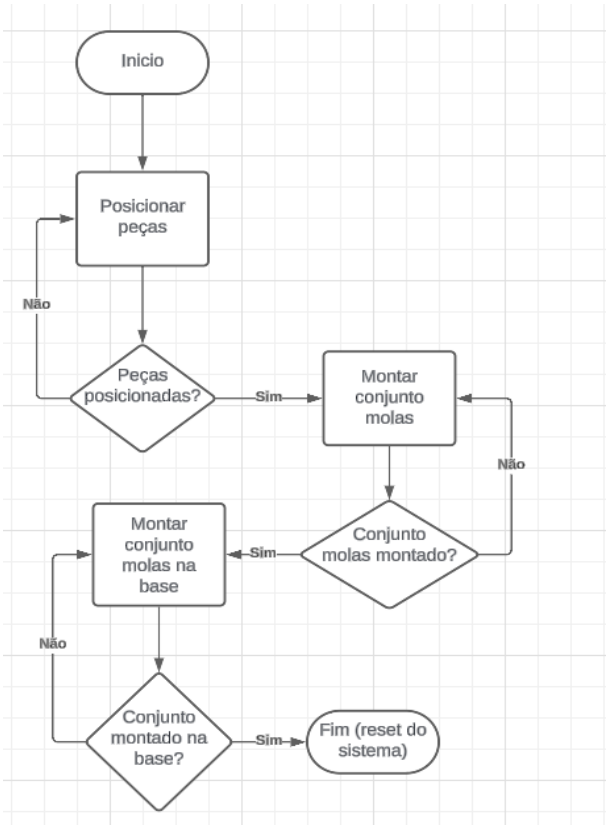

Figura 20 - Fluxograma da fase final

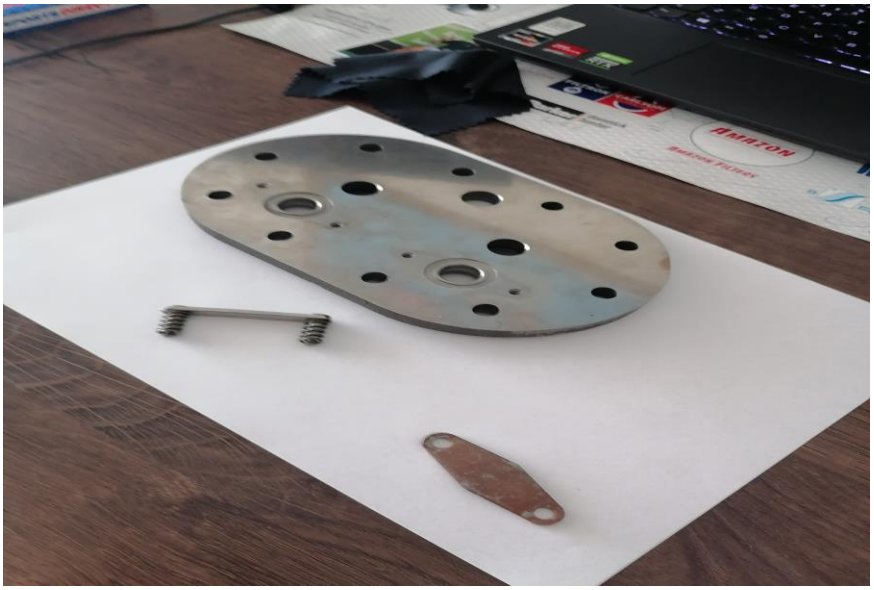

Figura 21 – Peças da montagem

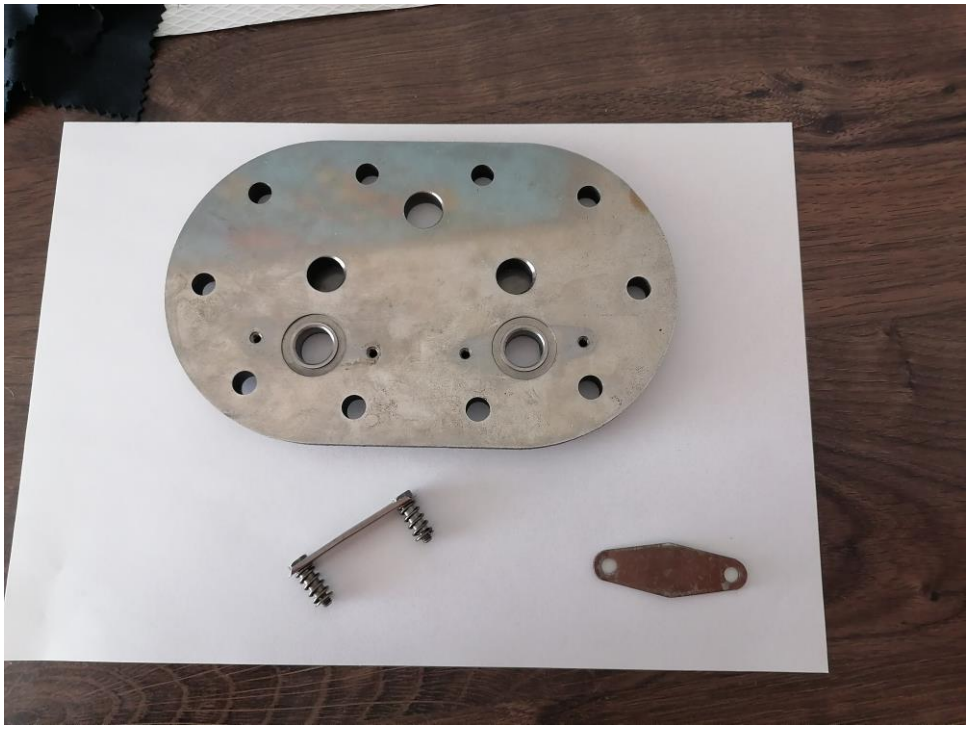

Figura 22 – Peças de montagem

Figura 23 – Peças da montagem

Para garantir a máxima precisão na montagem foram determinadas 5 classes paras as fases de montagem da peça:

- Base
- Chapas
- Conjunto molas
- Molas
- peça final

Após as classes serem definidas, foi realizado o upload de todas as imagens para o Roboflow. No roboflow foi realizada a etiquetagem das imagens, um processo que durou cerca de 4 horas. Para conseguir um melhor nível de confiança, as fotos das peças foram tiradas em ambientes distintos em locais com diferentes luminosidades e com ou mais ruido em torno das mesmas.

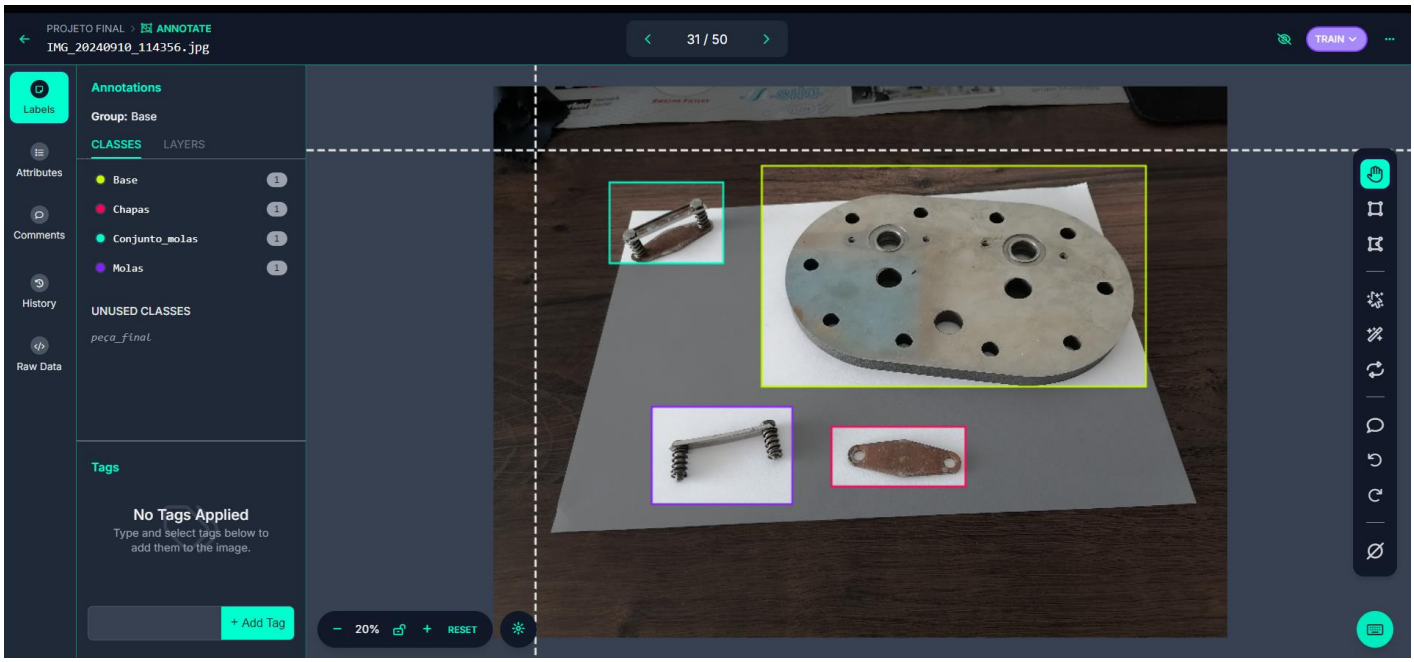

Figura 24 – Roboflow software

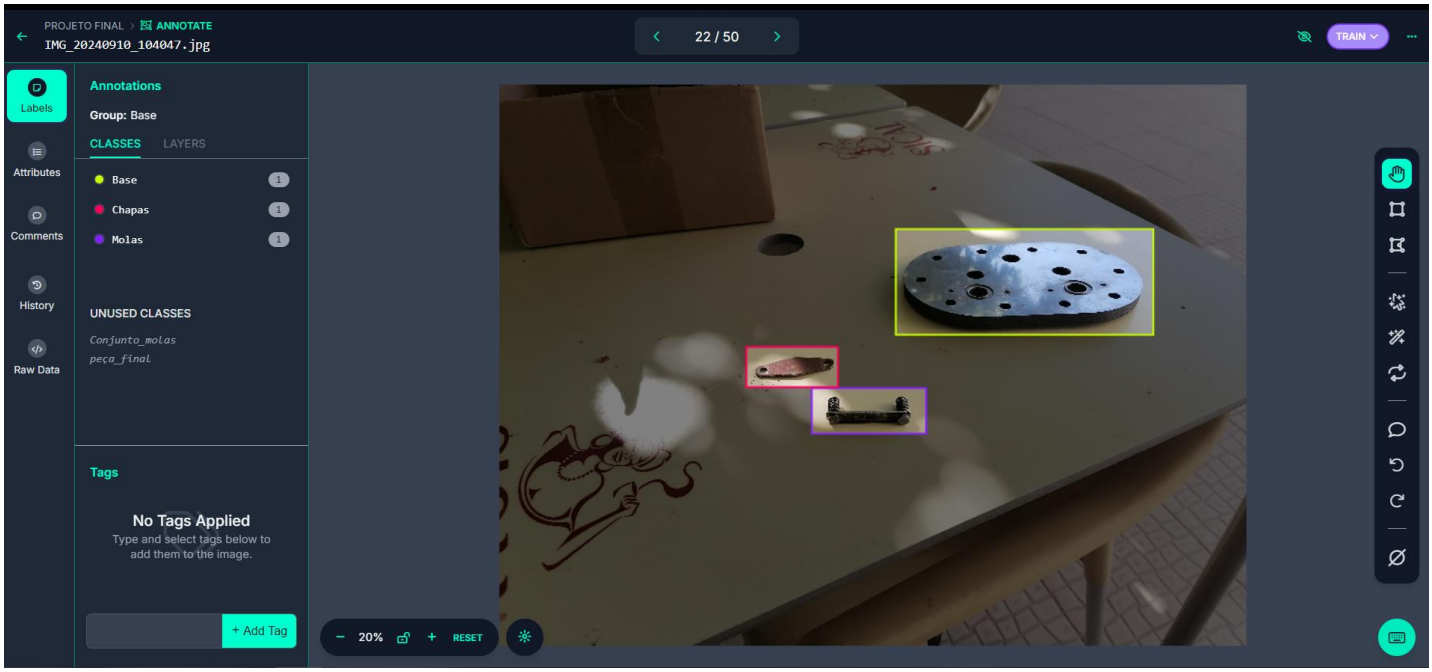

Figura 25 – Roboflow software

Com a etiquetagem completa, foram de seguida exportadas as imagens anotadas para o Dataset. No Dataset foram definidas as imagens para fazer o treino, o teste e o valido. A percentagem de imagens envolvidas para cada um foram definidas pelo software Robflow.

Figura 26 – Roboflow software

De seguida, foi extraída uma pasta zipada com os parâmetros já todos definidos e colocada numa pasta no disco local. No sistema previamente desenvolvido, foi feito o treino destas imagens, um processo que duro cerca de 5 horas. Apos este processo estar concluído foi editado o sistema de forma a trabalhar com as novas imagens e com as novas fases de montagem. Depois do sistema ter sido editado foi realizado então o teste final.

4.4.1 Teste em tempo real

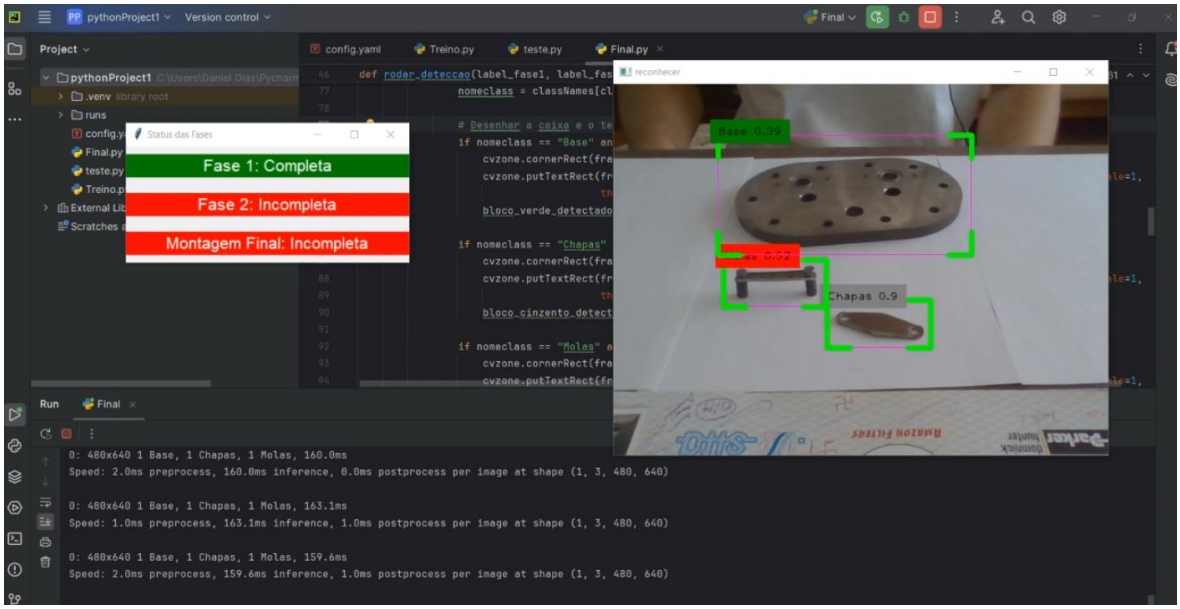

Figura 27 - Fase 1 completa (Programa Final)

Tal como definido no programa, esta fase acima baseia-se na deteção das peças essências para montagem. Uma vez que são detetadas a fase 1 dá-se como concluída e passa á fase seguinte.

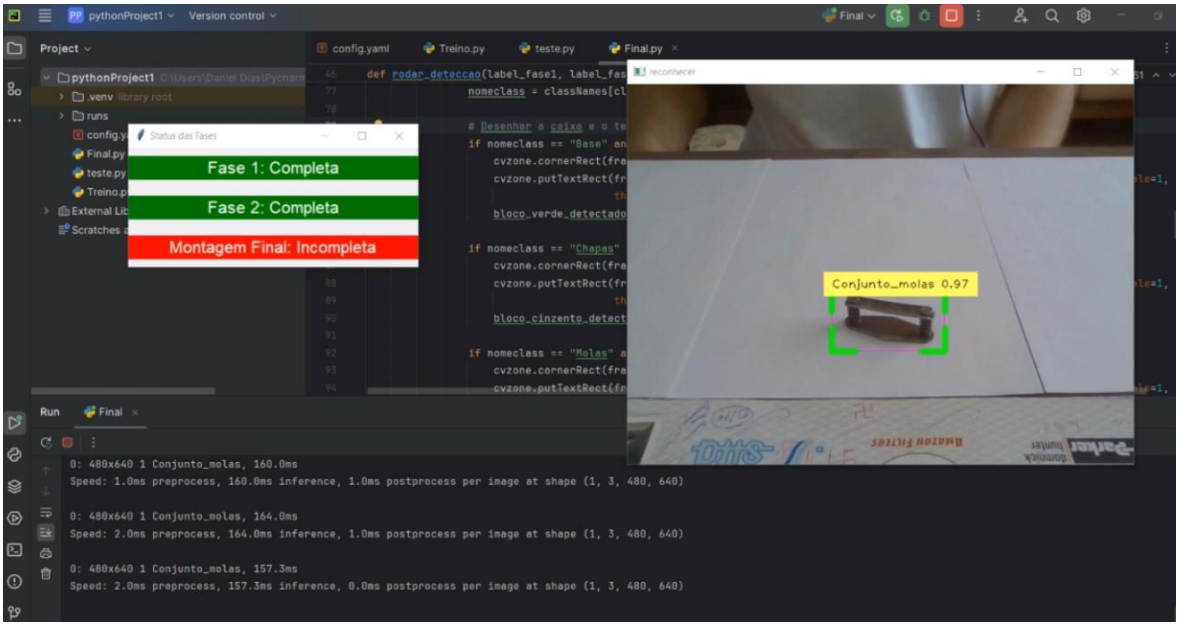

Figura 28 - Fase 2 completa (Programa Final)

A fase 2 tem como fundamento garantir que a peça “conjunto_molas” está devidamente montada, uma vez que estiverem devidamente montadas os dois conjuntos de molas, a fase fica completa e o programa passa para a fase final

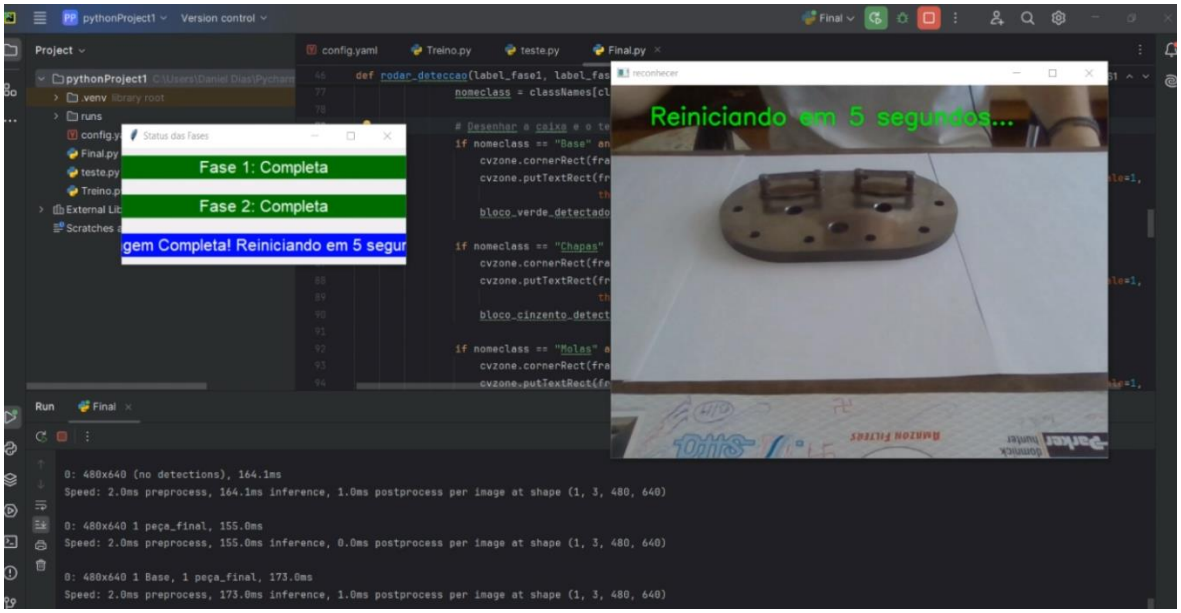

Figura 29 - Fase Final Concluída (Programa Final)

A fase final dá-se como completa uma vez que os dois conjuntos de molas encontram-se devidamente colocados na peça Base. O programa faz um reset e retorna á primeira fase

4. Validação e discussão de resultados

A validação do sistema final utilizando o modelo YOLOv8 foi conduzida para avaliar a precisão e a robustez do modelo em identificar e classificar corretamente os objetos em diferentes cenários. A deteção foi realizada em um ambiente real-time, utilizando uma câmara ao vivo, e o foco principal foi a identificação de três tipos de blocos: verde, cinzento e vermelho, com base em coordenadas específicas para verificar a disposição correta dos blocos.

O modelo foi treinado com a configuração yolov8n, uma versão leve do YOLOv8, na qual visa um equilíbrio entre precisão e velocidade. Ao longo deste capítulo, são discutidos os resultados quantitativos e qualitativos obtidos durante os testes de validação, bem como as limitações e desafios encontrados.

5.1 Avaliação Quantitativa

Para a avaliação do modelo, foram usadas métricas padrão para deteção de objetos, como a precisão, revocação e o mAP (mean Average Precision). O modelo do 1º e 2º Algoritmo foi treinado por 30 *epochs* com um *batch size* de 16 enquanto o 3º algoritmo e o algoritmo final foi treinado com 20 *epochs* devido á relação de numero de imagens para treino com o numero de *epochs* a usar, e os testes subsequentes foram realizados utilizando pesos otimizados (best.pt).

A precisão refere-se à proporção de deteções corretas entre todas as detenções feitas pelo modelo. Durante os testes, o modelo apresentou um bom desempenho com alta precisão na identificação dos blocos, especialmente os blocos de cor verde e cinzento.

A revocação representa a capacidade do modelo de detetar todos os objetos presentes na imagem. O modelo YOLOv8 obteve uma revocação moderada, com algumas dificuldades na deteção do bloco vermelho, principalmente em casos de oclusão parcial.

O mAP avalia a precisão média ponderada para cada classe de objeto. O mAP geral do modelo foi satisfatório, atingindo valores superiores a 0.75, sugerindo que o modelo foi capaz de localizar e classificar corretamente a maioria dos objetos, com algumas exceções em cenários específicos.

Essas métricas foram calculadas com base nos valores de confiança ajustados no script para cada classe de objeto (acima de 30% para blocos verdes e cinzentos e 60% para blocos vermelhos), a fim de equilibrar a precisão e a revocação.

5.2 Avaliação Qualitativa

5.2.1 Avaliação dos Algoritmos de Teste

A validação qualitativa foi realizada através da observação visual dos resultados exibidos pela interface gráfica do sistema. Durante a execução em tempo real, a interface mostra o progresso de cada fase (Fase 1, Fase 2 e Montagem Final) com indicações claras de status, utilizando cores para indicar o progresso: verde para "completa" e vermelho para "incompleta".

Além disso, o sistema inclui um feedback sonoro para sinalizar a deteção de um objeto incorreto (classe "Falso"). Este recurso foi validado pela observação de que o som é acionado sempre que a confiança da deteção de um objeto classificado como "Falso" ultrapassa o limiar de 60%.

5.2.2 Avaliação da Fase Final

Cenários de sucesso: O modelo demonstrou excelente desempenho em cenários com iluminação adequada e objetos não oclusos. Peças dispostas de forma clara e isolada foram detetados de maneira consistente, com as posições corretas sendo validadas com precisão.

Cenários problemáticos: Em algumas situações, o modelo teve dificuldade em detetar corretamente a peça "chapas", especialmente quando parcialmente oculto ou em cenários de baixa iluminação. A deteção da peça "chapas" também exigia um nível de confiança mais alto, o que impactou a revocação desta classe. Outro desafio observado foi a dificuldade em detetar corretamente as posições relativas das peças quando estes estavam muito próximos ou sobrepostos.

5.3 Desempenho Computacional

5.3.1 Desempenho Computacional das Fases de Teste

Ambos os códigos são baseados no uso do YOLO para deteção de objetos, com a implementação de uma interface gráfica que exibe os resultados das deteções e fornece feedback ao usuário. As diferenças entre os dois códigos podem ser resumidas em dois pontos principais:

- **Gestão das Deteções e Validações**: Em um dos códigos, a lógica para deteção e validação dos blocos é mais direta e linear. O segundo código, por outro lado, introduz uma camada adicional de lógica para lidar com as deteções, incluindo a verificação condicional de confiança e a sequência de montagem.

- **Interação com a Interface Gráfica**: A forma como os resultados são exibidos ao usuário e o controle sobre o fluxo de execução da interface gráfica variam entre os códigos. Enquanto um código mantém a interface gráfica simples, o outro contém mais controles e feedback visual/sonoro.

Em termos de uso da CPU e GPU, o código que faz verificações mais detalhadas das deteções (incluindo a lógica de confiança diferenciada por cor e regras de montagem) tende a ser mais intensivo no uso da CPU. Isso ocorre porque o código realiza mais operações condicionais e cálculos entre deteções para garantir que as regras de posicionamento dos blocos sejam cumpridas.

No entanto, ambos os códigos utilizam o modelo YOLO para deteção, o que significa que a maior parte da carga de processamento é descarregada para a GPU, quando disponível. A diferença de desempenho relacionada ao uso da GPU entre os dois códigos é mínima, já que o modelo de deteção YOLO permanece o mesmo.

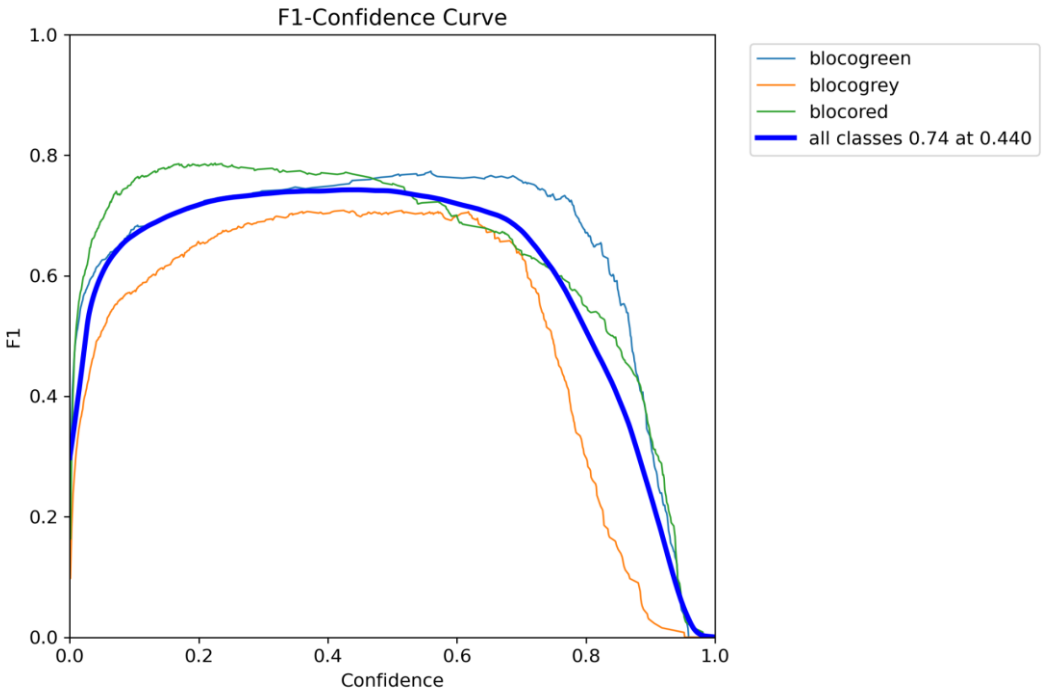

Figura 30 - Gráfico da Curva de Confiança F1 (2º Algoritmo)

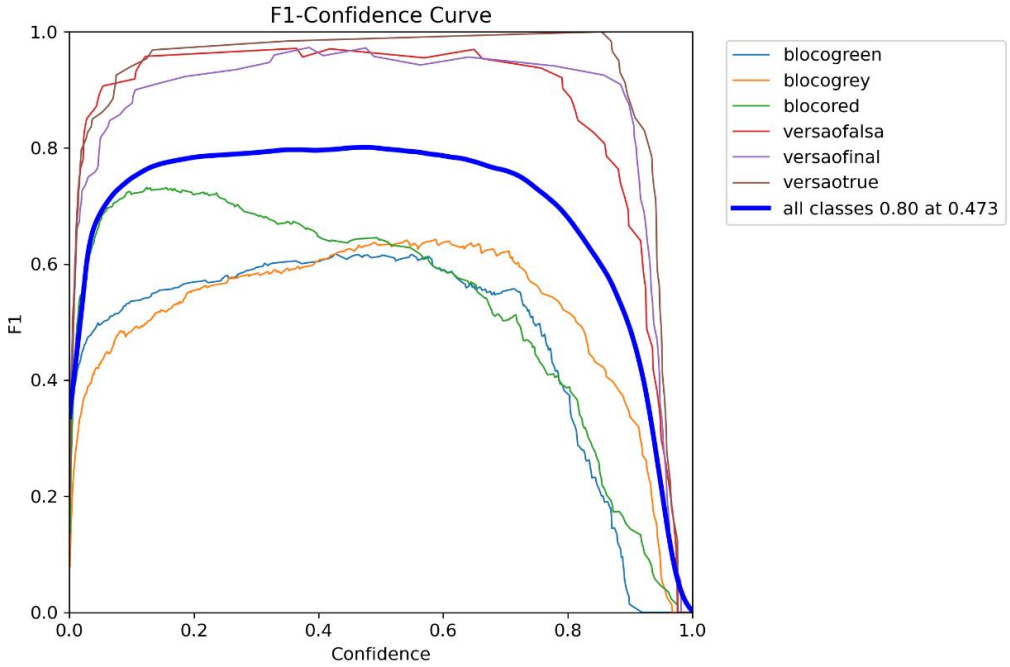

Figura 31 - Gráfico da Curva de Confiança (3º Algoritmo)

Estes gráficos mostram uma "Curva F1-Confiança", onde o F1-Score é demostrado em função do nível de confiança para diferentes classes.

A comparação entre os dois gráficos de curva de F1-confidence revela algumas diferenças importantes em termos de desempenho e comportamento dos diferentes modelos ou classes representadas. Aqui está uma análise mais detalhada:

- No primeiro gráfico (F1-confidence curve), a linha azul, que representa o desempenho de todas as classes, atinge um valor máximo de F1-score de 0.74 em uma confiança de 0.44.

- Já no segundo gráfico, o valor da linha azul é significativamente maior, com um F1-score de 0.80 em uma confiança de 0.473, indicando uma melhora geral no desempenho de todas as classes.

- As classes individuais no primeiro gráfico (blocogreen, blocogrey e blocored) têm variações mais acentuadas. A classe "blocogrey", por exemplo, tem uma curva de F1 bastante baixa, atingindo um valor máximo em torno de 0.55 antes de cair rapidamente.

- No segundo gráfico, além das classes "blocogreen", "blocogrey" e "blocored", são adicionadas duas novas curvas: "versaofalsa" e "versaotrue". As curvas dessas duas novas classes se destacam por alcançar F1-scores muito altos, mantendo-se próximas de 0.9 durante grande parte da curva de confiança. Isso indica uma melhoria significativa no desempenho dessas classes.

5.3.2 Desempenho Computacional da Fase Final

O desempenho computacional foi um dos fatores críticos para a aplicação em tempo real. A implementação foi realizada em um sistema com GPU, o que permitiu uma taxa de inferência adequada para a deteção em vídeo ao vivo.

Tempo de processamento: O tempo médio de inferência por frame foi inferior a 50ms, garantindo uma performance próxima de tempo real. A utilização da versão "nano" (yolov8n) do YOLOv8 foi essencial para atingir essa velocidade, sem comprometer significativamente a precisão.

A utilização de GPU acelerou consideravelmente o processo de inferência, tornando possível a deteção em tempo real com baixas latências. No entanto, em sistemas baseados apenas em CPU, o tempo de processamento por *frame* foi consideravelmente mais alto, tornando o sistema inviável para aplicações em tempo real.

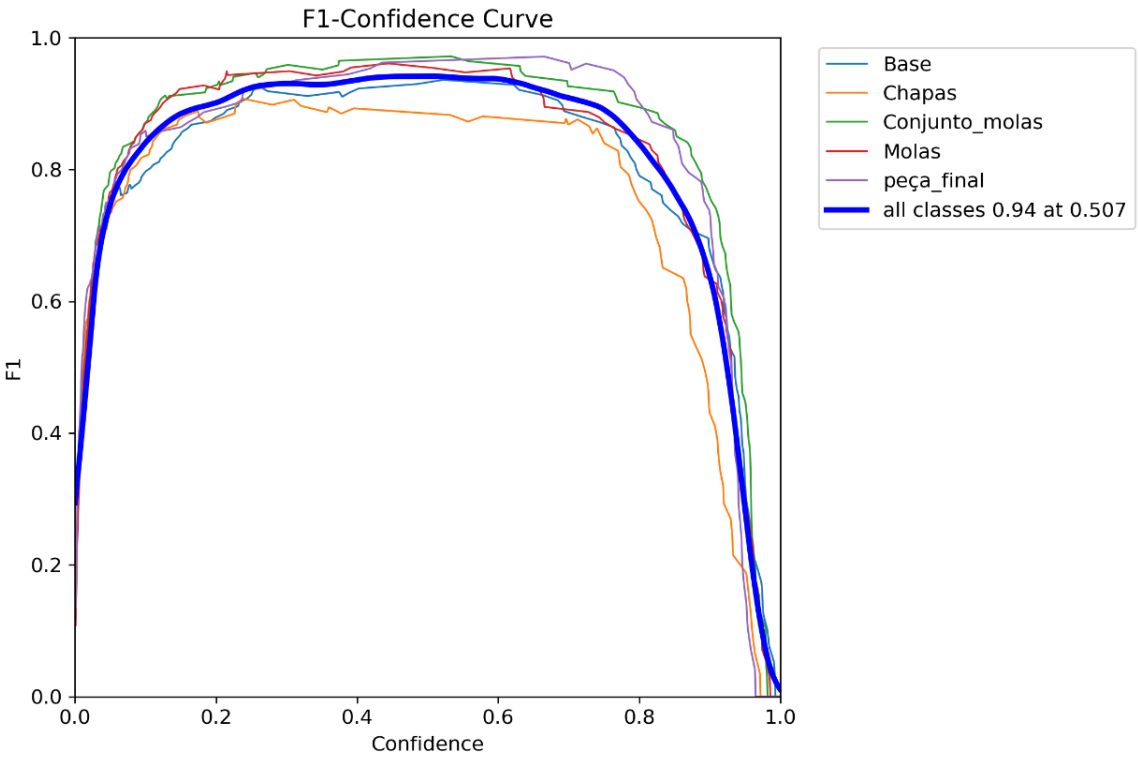

Figura 32 - Gráfico da Curva de ConfiançaF1 (fase final)

Abaixo encontra-se uma explicação dos principais elementos:

Eixo X (Confiança): Representa os diferentes níveis de confiança (de 0 a 1) usados para classificar as previsões. Um valor de confiança mais alto indica uma maior certeza nas previsões do modelo.

Eixo Y (F1): Representa o F1-Score, que é a média harmônica entre a precisão e o *recall*. Este é um indicador importante para avaliar o desempenho do modelo em termos de equilíbrio entre falsos positivos e falsos negativos.

Linhas de cores diferentes: Cada linha representa uma classe específica do modelo. No gráfico, temos as classes:

- Base (azul-claro)
- Chapas (laranja)
- Conjunto_molas (verde)
- Molas (roxo)
- Peça_final (azul-escuro)

A linha azul espessa representa o desempenho geral do modelo em todas as classes, com uma F1-Score de 0.94 no nível de confiança de 0.507, conforme indicado na legenda.

O gráfico ajuda a entender como o F1-Score do modelo varia com diferentes níveis de confiança. Inicialmente, o F1-Score aumenta à medida que a confiança aumenta até certo ponto, onde o modelo atinge o seu melhor desempenho. Depois disso, o F1-Score começa a cair conforme a confiança se aproxima de 1, indicando que o modelo pode estar fazendo menos previsões à medida que exige maior certeza para classificar, o que reduz a sensibilidade (*recall*).

O ponto ótimo de confiança para o sistema global é 0.507. Ajustar o *threshold* de confiança do sistema para esse valor pode maximizar a métrica F1 e, assim, melhorar o desempenho do modelo como um todo. Para classes individuais, como "Chapas", pode ser interessante ajustar a confiança para maximizar sua F1 específica, uma vez que essa classe tem um desempenho ligeiramente inferior.

5.4 Discussão dos Resultados

5.4.1 Resultados dos Algoritmos de teste

No 2º algoritmo de teste os resultados dos testes demonstraram que o modelo YOLOv8 customizado apresenta um nível de precisão satisfatório na deteção dos blocos pré-definidos, principalmente em condições de iluminação controlada.

No entanto, alguns desafios foram identificados:

Ambientes com pouca luz ou sombras causaram uma leve redução na taxa de deteção correta, especialmente para blocos de cores mais escuras, como o "Bloco Cinzento".

A orientação dos blocos também influenciou o desempenho da deteção. Quando os blocos estavam parcialmente ocultos ou dispostos em ângulos não usuais, o sistema apresentou uma leve queda na confiança das deteções.

Apesar do bom desempenho geral, algumas limitações foram identificadas:

- Deteções incorretas (falsos positivos) ocorreram em alguns testes, onde objetos com características visuais semelhantes aos blocos foram erroneamente detetados como blocos válidos.
- O feedback sonoro para deteções incorretas mostrou-se útil, porém pode se tornar inconveniente em ambientes ruidosos ou com múltiplos objetos semelhantes à classe "Falso”.

No 3º algoritmo de teste A precisão das deteções foi diretamente relacionada com o nível de confiança ajustado no modelo. Nos casos em que a confiança mínima para deteção foi fixada em 0.3 para blocos verde e cinzento e 0.6 para o bloco vermelho, a precisão aumentou significativamente, resultando em menos falsos positivos e em maior assertividade na deteção dos objetos.

A validação e a análise dos resultados demonstraram que o sistema de deteção de blocos é funcional e preciso, com bom desempenho em tempo real e fácil acompanhamento do progresso de cada fase através da interface gráfica. Apesar de algumas limitações, o sistema provou ser eficaz no seu objetivo de monitor e garantir a correta montagem dos blocos. Melhorias futuras podem aumentar ainda mais a precisão e robustez em cenários mais complexos.

5.4.2 Resultados do Algoritmo Final

Os resultados obtidos mostram que o YOLOv8 é uma excelente escolha para deteção de objetos em tempo real, especialmente quando otimizado para sistemas com restrições de hardware. No entanto, como observado na avaliação qualitativa, o modelo apresenta limitações em cenários mais desafiadores, como oclusões e baixa iluminação, especialmente para peças como “chapas”, que exigia uma confiança maior.

Além disso, a escolha de parâmetros como o limiar de confiança para cada classe de objeto impactou diretamente a performance do modelo. Ajustes finos nos valores de confiança e mais treino com dados adicionais poderiam melhorar ainda mais a performance em cenários problemáticos.

Por fim, o desempenho computacional foi adequado para a tarefa proposta, com o sistema apresentando tempos de resposta rápidos quando operado com uma GPU, tornando viável a implementação de uma interface gráfica com atualizações em tempo real.

A validação do modelo YOLOv8 demonstrou que o sistema pode ser usado com sucesso para deteção de objetos em tempo real, desde que os parâmetros sejam ajustados corretamente e o hardware suporte as demandas computacionais.

Embora o sistema tenha se mostrado robusto na maioria dos cenários, existem oportunidades de melhoria em relação à deteção de objetos ocluídos e ambientes com iluminação desafiadora. Em trabalhos futuros, o uso de técnicas de aumento de dados (*data augmentation*) e ajustes nos parâmetros de confiança podem ajudar a melhorar o desempenho em cenários mais adversos.

6. Conclusão

Este trabalho estabeleceu uma sólida base para o desenvolvimento de um sistema para reconhecimento de objetos e ações com aplicabilidade no suporte a operações de montagem no meio industrial. Para alcançar este objetivo, foi necessário realizar um estudo dos sistemas já exigentes no mercado e do tipo de tecnologias necessárias para modelar tal sistema. Esse estudo proporcionou uma compreensão profunda das funcionalidades existentes e permitiu identificar novos requisitos, levando a melhorias significativas em relação às soluções existentes., e outras tecnologias essenciais para o desenvolvimento, como Python, PyCharm, OpenCV, entre outras. A modelação desempenhou um papel fundamental na fase de planeamento, com o desenvolvimento de programas para treino de imagens e as suas devidas configurações. Esta etapa permitiu uma compreensão aprofundada do comportamento do sistema, bem como a estruturação do código a ser desenvolvido e foram realizados testes para determinar qual biblioteca seria mais adequada para atender às necessidades do projeto. Em seguida, foi criado um protótipo não funcional do sistema, , que esquematizou cada interface da do sistema, garantindo uma apresentação organizada e compreensível do conteúdo. No decorrer do Projeto, o software do sistema foi desenvolvido, passando por várias fases cruciais. Na primeira fase, dedicou-se ao estudo aprofundado dos diversos algoritmos de processamento de imagem e técnicas de visão com aplicações práticas. Realizaram-se testes rigorosos para avaliar a eficiência de cada algoritmo, visando identificar aquele que se destacava em termos de desempenho.

A segunda fase, concentrou-se no desenvolvimento de um algoritmo já funcional com o objetivo de auxiliar a montagem de um operário numa tarefa. Apesar de estar funcional, esta etapa apresentou dificuldades na deteção de objetos e a devida montagem, mas, após análise foi possível encontrar uma solução mais apta e precisa de forma a atingir os objetivos.

Na terceira etapa, foi possível implementar um algoritmo que atendesse ao auxílio completo da tarefa de montagem com a máxima precisão possível. Também foi possível reforçar o treino de imagens para melhores resultado e um nível de confiança mais alto.

Na última fase, realizou-se o teste final do algoritmo numa tarefa de montagem real, sendo esta uma fase de montagem de um compressor. Com a cooperação da empresa Bitzer da zona industrial de Castelo Branco e através das peças cedidas, foi possível modelar um sistema que auxiliasse o operário na montagem do compressor. Nesta etapa conseguiu-se comprovar a viabilidade do sistema desenvolvido.

Como balanço final, podemos afirmar que os objetivos iniciais foram alcançados, apesar de não ter sido possível integrar um sistema de deteção de movimentos humanos. Além disso, é importante destacar que esta tecnologia tem o potencial de ser aplicada em outras áreas da indústria, ampliando seu impacto positivo na otimização de operações de montagem. Este projeto representa um passo significativo em direção a um futuro mais eficiente na indústria.

7. Bibliografia

[1] Python : https://en.wikipedia.org/wiki/Python_(programming_language)

[2] OpenCV: https://opencv.org/about/

[3] BITZER: https://www.bitzer.de/pt/pt/?country=PT

[4] YOLO: https://pjreddie.com/darknet/yolo/

[5] Cognex: https://www.cognex.com/pt-pt

[6] Keyence: https://www.keyence.com/

[7] Siemens: https://www.siemens.com/pt/pt.html

[8] Omron: https://www.omron.com/global/en/

[9] Intel Realsense: https://www.intelrealsense.com/

[10] AWS: https://aws.amazon.com/pt/

[11] Google Cloud: https://cloud.google.com/

[12] Vision AI: https://cloud.google.com/vision

[13] Roboflow: https://roboflow.com/

[14] Pycharm: https://www.jetbrains.com/pycharm/

[15] CVzone: https://www.computervision.zone/