



MINISTÉRIO DA EDUCAÇÃO INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO AMAZONAS
CAMPUS MANAUS DISTRITO INDUSTRIAL



CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO

LUCAS DE SOUZA MARQUES

**SISTEMA DE DETECÇÃO DE OBJETO PERSONALIZADO
UTILIZANDO INTELIGÊNCIA ARTIFICIAL**

MANAUS

2020

LUCAS DE SOUZA MARQUES

**SISTEMA DE DETECÇÃO DE OBJETO PERSONALIZADO
UTILIZANDO INTELIGÊNCIA ARTIFICIAL**

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

Orientador: Prof. Dr. Vitor Bremgartner da Frota.

MANAUS

2020

LUCAS DE SOUZA MARQUES

SISTEMA DE DETECÇÃO DE OBJETO PERSONALIZADO UTILIZANDO INTELIGÊNCIA ARTIFICIAL

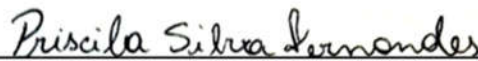
Trabalho de conclusão de curso apresentado ao curso de Engenharia de Controle e Automação do Campus Manaus Distrito Industrial do Instituto Federal de Educação, Ciência e Tecnologia do Amazonas – IFAM, como requisito obrigatório para obtenção do grau de Bacharel em Engenharia de Controle e Automação.

Aprovado em 06 de novembro de 2020.

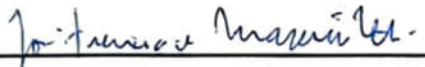
BANCA EXAMINADORA



Prof. Dr. Vitor Bremgartner da Frota
Orientador



Prof. MSc. Priscila Silva Fernandes
Professor Avaliador



Prof. Dr. José Magalhães Netto
Professor Avaliador

Dedico este trabalho à Deus, minha família, professores e amigos.

AGRADECIMENTOS

A Deus por tudo o que fez por mim e por permitir que eu conclua meus objetivos conforme a sua vontade.

A minha família por cuidar de mim, por todo apoio e por ser a minha maior motivação e inspiração para seguir minha carreira. Ao professor Vitor Bremgartner por acreditar neste projeto e oferecer todo o suporte possível para sua realização e reconhecimento.

Aos amigos e professores que estiveram comigo durante a graduação, agradeço pela amizade e pelos conhecimentos compartilhados.

Aos engenheiros, técnicos e operadores da empresa pelo suporte na implementação do projeto.

Agradeço por todos aqueles que me apoiaram, me ajudaram e que torceram por mim.

“Os homens devem moldar seu caminho. A partir do momento em que você vir o caminho em tudo o que fizer, você se tornará o caminho.”

Miyamoto Musashi.

RESUMO

MARQUES, Lucas. **SISTEMA DE DETECÇÃO DE OBJETO PERSONALIZADO UTILIZANDO INTELIGÊNCIA ARTIFICIAL**. 2020. Trabalho de Conclusão de Curso Bacharelado em Engenharia de Controle e Automação – Instituto Federal de Educação, Ciência e Tecnologia do Amazonas – Campus Manaus Distrito Industrial. Manaus, Amazonas, 2020.

Este trabalho tem por objetivo a verificação automática de objetos numa linha de produção. Atualmente existe um mercado para detecção de objetos que compreende a valores acima dos R\$ 100 mil reais para a implementação do mesmo projeto. No entanto, o trabalho aqui apresentado tem como base ser desenvolvido em uma plataforma completamente *Open Source*. A proposta consiste em criar um sistema de baixo custo com Visão Computacional em linguagem *Python* que resolva os problemas de paradas de linhas instantâneas e problemas em campo relacionados à falta do objeto e fazê-lo portátil para qualquer computador. Devido ao tempo rápido da linha, algumas caixas passam despercebidas aos olhos do operador e com o sistema implementado, espera-se que todas as caixas sejam controladas e detectem todos os manuais, tomando suas próprias ações de forma autônoma sem a necessidade de intervenção humana seguindo o conceito de Indústria 4.0 e Internet das Coisas. O sistema desenvolvido teve respostas rápidas (cerca de 40 milissegundos) para avaliação de cada imagem e já se encontra implementado na fábrica.

Palavras-chave: Visão Computacional, Internet das Coisas, Indústria 4.0.

ABSTRACT

MARQUES, Lucas. **PERSONALIZED OBJECT DETECTION SYSTEM USING ARTIFICIAL INTELLIGENCE. 2020.** Course Conclusion Work Bachelor in Control and Automation Engineering - Federal Institute of Education, Science and Technology of Amazonas - Campus Manaus Distrito Industrial. Manaus, Amazonas, 2020.

This work aims to develop an automatic object verification system on a production line. Currently, there is a market for object detection that comprises values above R\$ 100 thousand reais for the implementation of the same project. The project is based on being developed on a completely Open Source platform. The proposal consists of creating a low-cost Python language system that solves the problems of instantaneous line stops and field problems related to the lack of the object and make it portable to any computer. Due to the fast time of the line some boxes go unnoticed in the eyes of the operator and with the implemented system it is expected that all boxes are controlled and detect all manuals and take their own actions autonomously without the need for human intervention following the concept of industry 4.0. The developed system had quick responses (about 40 milliseconds) for the evaluation of each image and is already implemented in the factory.

Keywords: Computer Vision, IOT, Industry 4.0.

Lista de Siglas

API	<i>Application Programming Interface</i>
CPU	<i>Central Process Unit</i>
CUDA	Compute Unified Device Architecture
CUDNN	<i>CUDA Deep Neural Network</i>
DRY	<i>Don't Repeat Yourself</i>
GPU	<i>Graphics Processing Unit</i>
GUI	<i>Graphical User Interface</i>
IA	Inteligência Artificial
IDE	<i>Integrated Development Environment</i>
IHM	Interface Homem Máquina
IoT	<i>Internet Of Things</i>
IP	<i>Internet Protocol</i>
MBAP	<i>Modbus Application Header</i>
PC	<i>Personal Computer</i>
PDU	<i>Protocol Data Unit</i>
RAD	<i>Rapid Application Development</i>
RTU	<i>Remote Terminal Unit</i>
SCADA	<i>Supervisory Control and Data Acquisition</i>
SQL	<i>Structured Query Language</i>
TCP	<i>Transmission Control Protocol</i>

Lista de Figuras

Figura 1- Python Logo	19
Figura 2- Estrutura Python	20
Figura 3-PyCharm.....	20
Figura 4- Funcionamento do PyQt5	21
Figura 5- Projeto de monitoramento de energia.....	22
Figura 6- OpenCV logo	23
Figura 7-Detecção de Carros.....	24
Figura 8-Inspeção Chassi	25
Figura 9- Face Recognition AI	25
Figura 10-Tensorflow	26
Figura 11- Reconhecimento de gestos em libras.....	27
Figura 12- Sistema Biogás.....	28
Figura 13- Estrutura Modbus	29
Figura 14- Protocolo Modbus.....	30
Figura 15- Webcam C270	32
Figura 16- CLP LSIS XGB Ethernet.....	33
Figura 17- Notebook Ideapad 330	33
Figura 18- Interface Pycharm.....	34
Figura 19- Comando para Instalação de Biblioteca	35
Figura 20- QtDesigner Interface.....	35

Figura 21- Linha de comando para instalação do <i>Protobuf</i>	36
Figura 22- NVIDIA CUDA logo.....	36
Figura 23- Caminhos CUDA	37
Figura 24- Linha de Comando para Geração de Código	39
Figura 25- Código Gerado em Python	39
Figura 26- Interface Projeto	40
Figura 27- Abas da interface.....	41
Figura 28- Amostra de imagem do manual.....	41
Figura 29- Linha de comando para abrir LabelImg	42
Figura 30- Interface LabelImg.....	42
Figura 31- Formato Pascal	43
Figura 32- Algoritmo para conversão em formato CSV	43
Figura 33- Formato CSV.....	44
Figura 34- Trecho de código <i>generatetfrecord.py</i>	44
Figura 35- Linha de comando para gerar arquivo .record.....	45
Figura 36- labelmap.pbtxt	45
Figura 37- Trecho do arquivo <i>.config</i>	46
Figura 38- Linha de comando para treinar a inteligência	46
Figura 39- Saída train.py	47
Figura 40- Pasta de treinamento.....	48
Figura 41- Linha de comando para exportar inteligência	48

Figura 42- Arquivo gerado pronto para teste	49
Figura 43- Saída do algoritmo de avaliação	49
Figura 44- Mudança de <i>ROI</i>	50
Figura 45- Falso negativo do manual em Faster RCNN	50
Figura 46- Boa detecção com SSD.....	51
Figura 47- Painel de Controle XAMPP.....	52
Figura 48- Interface do banco de dados	53
Figura 49- Estrutura do banco de dados do projeto.....	53
Figura 50- CLP LSIS.....	54
Figura 51- Tela de configuração do CLP	54
Figura 52-Configuração Modbus.....	55
Figura 53-Programa Ladder.....	57
Figura 54- Interface com câmera	58
Figura 55- Aba de configuração.....	58
Figura 56- Comando CamChange	59
Figura 57- Tela de conexão e erro do Modbus/TCP.....	59
Figura 58- CLP Modbus/TCP conectado.	60
Figura 59- Código não detectado.....	60
Figura 60- Objeto não detectado	61
Figura 61- Tela detectou OK.....	62
Figura 62- Histórico do sistema	63

Sumário

1 INTRODUÇÃO	15
1.1 JUSTIFICATIVA.....	16
1.2 OBJETIVOS.....	17
1.3 METODOLOGIA	18
1.4 ESTRUTURA DO TRABALHO	18
2 REFERENCIAL TEÓRICO.....	19
2.1 PYTHON.....	19
2.1.1 PYCHARM	20
2.1.2 PYQT	21
2.1 VISÃO COMPUTACIONAL.....	23
2.1.1 <i>OPENCV</i>	23
2.2 INTELIGÊNCIA ARTIFICIAL (IA).....	24
2.2.1 <i>MACHINE LEARNING</i>	25
2.2.1.1 TENSORFLOW.....	26
2.3 INDÚSTRIA 4.0.....	27
2.4 COMUNICAÇÃO <i>MODBUS</i>	28
3 MATERIAIS E DESENVOLVIMENTO DO SISTEMA	32
3.1 MATERIAIS.....	32
3.2 CONFIGURAÇÕES INICIAIS	34

3.2.1	INSTALAÇÃO <i>TENSORFLOW</i> API.....	36
3.2.2	INSTALAÇÃO NVIDIA CUDA TOOLKIT E CUDNN.....	36
3.3	DESENVOLVIMENTO GRÁFICO DA APLICAÇÃO	38
3.4	DESENVOLVIMENTO DO SISTEMA.....	41
3.4.1	LABELIMG	42
3.4.2	TREINAMENTO	43
3.5	FORMATANDO OS DADOS.....	43
3.6	TREINANDO A INTELIGÊNCIA ARTIFICIAL	45
3.7	TESTANDO A INTELIGÊNCIA ARTIFICIAL	48
3.8	COMPARATIVO SSD VS FASTER RCNN	49
3.9	BANCO DE DADOS	52
3.10	Modbus/TCP	53
4	TESTES E RESULTADOS OBTIDOS	57
5	CONCLUSÃO	63
6	REFERÊNCIAS	64

1 INTRODUÇÃO

A indústria busca, cada vez mais, implementar sistemas capazes de se interligarem e comunicarem a partir do conceito de indústria 4.0, que visa a interconexão entre dispositivos por meio de uma comunicação robusta e acessível para monitoramento e controle inteligente dos sistemas.

Conforme Carvalho (2018), a indústria 4.0 não é um conceito novo, mas uma redefinição no qual as tecnologias em sinergia podem proporcionar diversos benefícios que impactam diretamente na manufatura como: redução de custo, economia de energia, aumento de segurança, redução de erros e uma maior transparência nos dados da linha de produção. Para desenvolvimento de aplicações com conceito de indústria 4.0 há múltiplos protocolos de comunicação utilizados e uma das mais utilizadas é a *Modbus* que se divide em dois tipos: o TCP que funciona por *Ethernet* e o RTU que é conectado por meio de porta Serial. Para Freitas (2014), a comunicação *Modbus* é amplamente utilizada na indústria, em uma das 3 variações: RS232, RS485 e *Ethernet*.

Para um melhor desempenho na linha de produção, onde não há mais possibilidade de melhoria em um posto que é realizado de forma manual, a indústria 4.0 recorre para automações que conseguem realizar as mesmas operações de forma eficiente e em um tempo menor. Para a realização de tais atividades é necessário sensores, segurança, flexibilidade e comunicação. Além desses requisitos, a indústria 4.0 propõe que a automação trabalhe de forma inteligente por meio de da tomada de decisões de acordo com as informações que ele recebe no ambiente por meio de Inteligência Artificial (IA). Segundo Tessarini (2018), a IA consegue realizar múltiplos trabalhos complexos no chão de fábrica quando aplicada com *Internet of things* (IoT), mesmo em diferentes condições que seriam difíceis para um ser humano.

Para identificação do manual de instrução é usado uma câmera como uma fonte de informação e a comunicação entre a câmera e o sistema é mediada entre os drivers e técnicas de visão computacional. Segundo Marengoni (2010), visão computacional é onde a entrada é a imagem real e a saída são valores numéricos em forma de matriz e por meio desses valores a visão humana é emulada.

A linguagem de programação é o principal meio para o desenvolvimento de um sistema e a linguagem *Python*, que segundo Silva(2018), a linguagem *Python* é simples e robusta, contendo uma variedade de aplicações e ampla versatilidade para áreas comerciais, indústrias e pesquisa.

Com a tendência de expansão nesta área, muitas comunidades estão em atividade em prol do desenvolvimento de sistemas para indústria 4.0 por meio do uso de bibliotecas e *softwares Open Source*. Como a falta do manual de instrução na caixa é um problema recorrente na linha de produção, há a necessidade de uma tomada de decisão rápida com o fim de reduzir paradas de linha, problemas em campo ou retenção de lote, então para a resolução deste problema, teremos como base recursos de inteligência artificial, indústria 4.0 e visão computacional para que, com baixo custo, se faça a verificação automática dos manuais e a tomada de ação por parte do sistema, de acordo com as variáveis do ambiente. Portanto, o trabalho consiste em usar tais tecnologias para a criação de um posto de verificação inteligente de objeto utilizando visão computacional. Sendo capaz de encontrar o objeto e tomar ações de acordo com os dados obtidos de forma autônoma.

1.1 JUSTIFICATIVA

Devido ao alto nível de produção numa fábrica, há a possibilidade de falta de algum componente físico que, devido ao desgaste humano ou cansaço, pode passar despercebido na inspeção visual. Tais falhas de inspeção representam prejuízo para a empresa, seja em paradas instantâneas, lotes parados ou problema com o consumidor final, ou seja, qualquer erro mínimo no produto que deveria chegar em perfeito estado ao consumidor pode significar uma perda e prejuízo para a imagem da empresa.

Para o trabalho, foi optado pela detecção de um manual de instruções que, cerca de 6 vezes ao mês há reclamação do cliente final sobre a falta de manual e 20 vezes a falta é detectada durante a produção, causando parada instantânea. Tais problemas são resolvidos com um sistema automático que tira toda a responsabilidade de inspeção de uma pessoa, que pode falhar durante o decorrer da produção, para uma inteligência que responderá de maneira a reduzir erros na linha de produção sem sofrer nenhum desgaste.

Outra justificativa para a execução do projeto é que foram oferecidas soluções prontas para a fábrica, porém elas estavam muito acima do orçamento destinado para este projeto, preço este que chegava à R\$100.000,00 e mais pagamento anual de licença. Antes da empresa pagar tal preço, sugerimos desenvolver um projeto abaixo de R\$ 1000,00 na empresa onde o trabalho foi aplicado.

1.2 OBJETIVOS

Este trabalho tem como objetivo detectar a presença ou falta de um manual de instruções dentro de uma caixa de um produto em linha de montagem e tomar ações de forma automática avaliando as condições do ambiente.

O sistema tem seu foco na exibição e registro da presença de manuais de instrução dentro da caixa de micro-ondas, durante a produção em um curto tempo de teste a fim de não interferir no *Takt-Time* (tempo que leva para o posto completar sua função de inspeção de caixa) da linha e evitar problemas de campo devido à falta de manual.

Sendo assim, os objetivos específicos do presente trabalho são:

- Desenvolver sistema de baixo custo para detecção de manuais
- Facilitar a visualização de manuais em tempo real
- Otimizar a verificação de manuais com um tempo de resposta rápido

A preocupação principal para o projeto é fazer um sistema de baixo custo que seja robusto o suficiente para detectar o manual em condições variáveis. A principal linguagem de desenvolvimento é a Python, que contém os recursos para desenvolvimento do *Back-end* e *front-end* para o trabalho.

O projeto conta com uma interface gráfica que mostrará a detecção do objeto, o rastreamento da caixa, histórico de detecção e atuador para retirada da caixa. O sistema tem a opção de detecção automática ou desativada. Na automática, o sistema terá controle sobre a linha e tomará ações necessárias de acordo com as condições avaliadas.

1.3 METODOLOGIA

O projeto se configura como um trabalho de pesquisa e desenvolvimento já que é abordada as consequências da falta de manual e uma ação será desenvolvida. O pretendido é definir uma maneira de detectar o objeto dentro da caixa e realizar os seguintes passos:

- Levantamento de Pesquisas Bibliográficas;
- Fazer interface;
- Leitura de código da caixa;
- Conexão com Controlador Lógico Programável (CLP) *Modbus* para tomada de decisão;

Para este projeto foi escolhido a detecção por meio de IA, apesar da demora do aprendizado da inteligência no momento de execução ela se mostra mais rápida e efetiva que outras opções estipuladas, permitindo uma flexibilidade para futuros projetos no processo mantendo o custo baixo.

1.4 ESTRUTURA DO TRABALHO

O trabalho é dividido de forma macro em cinco capítulos. Além desta Introdução, temos:

- Referencial teórico: Mostra com mais detalhes os conceitos que foram apresentados na introdução.
- Materiais e Desenvolvimento: Levanta os materiais necessários para o desenvolvimento do projeto, configurações iniciais, apresentação das IDEs, instalações, criação de IA, formatação do banco de dados e testes do sistema.
- Testes e resultados: Demonstra os testes e resultados obtidos.

- Conclusão: Faz o fechamento deste trabalho, apontando as contribuições deste trabalho, com direcionamentos futuros.

2 REFERENCIAL TEÓRICO

Neste tópico serão abordados os conceitos sobre as tecnologias que serão usadas no projeto proposto.

2.1 PYTHON

A linguagem *Python* (Figura 1) é uma das linguagens mais populares atualmente, apesar de ser popular agora a linguagem *Python* se iniciou no final dos anos 80 e conforme Vaz (2018), a linguagem foi criada por Guido van Rossum e seu nome foi feito em homenagem aos comediantes do Monty Python. Atualmente a linguagem *Python* é gerida pela *Python Foundation* no site <https://www.python.org/>, que disponibiliza as últimas versões do *Python*, que atualmente se encontra na versão 3.8.

Figura 1- Python Logo



Fonte: Python Org

O *Python* conta com uma grande comunidade, que já desenvolveu mais de 300 mil bibliotecas com aplicações em várias áreas do ramo científico, como: ciência de dados, microcontroladores, visão computacional, e robótica. De acordo com Silva (2019), a linguagem *Python* é voltada para o RAD que é o desenvolvimento ágil através de metodologias e DRY que tem como proposta manter o código com uma estrutura limpa e fácil de entender. Atualmente o *Python* encontra-se difundida em várias aplicações como: Netflix, Google e Nasa.

Para Severance (2017) a linguagem *Python* é uma linguagem muito flexível que possibilita várias aplicações como: comunicação *WEB*, banco de dados e aplicativos. Apesar da *Python* contar com funções e estruturas parecidas com

qualquer outra linguagem, ela tem algumas peculiaridades, onde uma delas é a delimitação por indentação sem a necessidade de chaves, como a maioria das linguagens, e também não é necessário pontuação no final de uma linha de código (Figura 2).

Figura 2- Estrutura Python

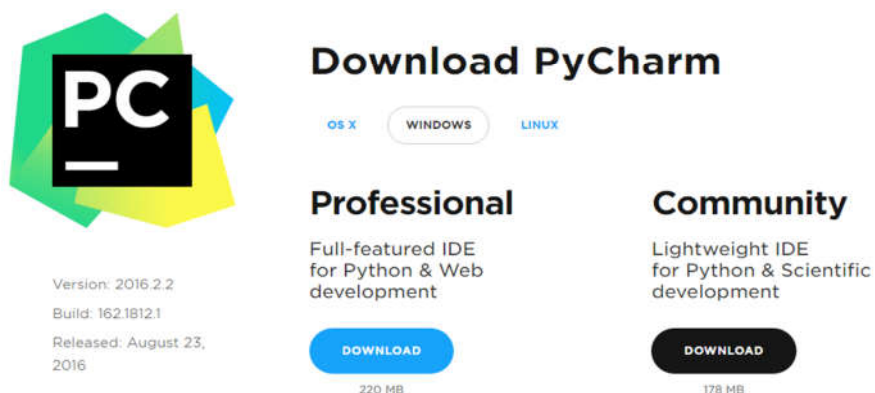
```
>>> def maior(a,b):
...     if a>b:
...         print("A é maior")
...     else:
...         print("B é maior")
... maior(4,3)
A é maior
```

Fonte: Próprio autor.

2.1.1 PYCHARM

O PyCharm é uma IDE desenvolvida pela *JetBrains* especialmente para *Python*, que conta com vários recursos de suporte para desenvolvimento. Para o projeto foi escolhida a versão *Community* a versão livre disponibilizada pela empresa, que contém os recursos necessários para o desenvolvimento. Para baixar basta visitar o site da desenvolvedora *JetBrains* em sua página oficial e escolher a versão *Community*.

Figura 3-PyCharm



The image shows the PyCharm download page. On the left is the PyCharm logo (PC) and version information: Version: 2016.2.2, Build: 162.1812.1, Released: August 23, 2016. In the center, there are three tabs for OS X, WINDOWS, and LINUX. Below these are two columns for 'Professional' and 'Community' editions. The Professional edition is described as a 'Full-featured IDE for Python & Web development' and has a 'DOWNLOAD' button with a size of 220 MB. The Community edition is described as a 'Lightweight IDE for Python & Scientific development' and has a 'DOWNLOAD' button with a size of 178 MB.

Fonte: Dos Reis(2016)

Como destaca Dos Reis (2016), o PyCharm é uma IDE completa para desenvolvedores *Python*. Contando com vários recursos de suporte para agilizar a programação, tais como:

- Debugger gráfico
- Unidade de testes integrada
- Integração com sistemas de controle de versão, como Git, Mercurial e Subversion
- Análise de código
- Code Completion
- Sintaxe e Erros destacados
- Refatoração
- Suporte a desenvolvimento com Django

O *PyCharm* oferece recursos de gerenciamento de projetos para uma visão macro da sua aplicação, além de disponibilizar ferramentas de console para programação em *console* e execução de *scripts*.

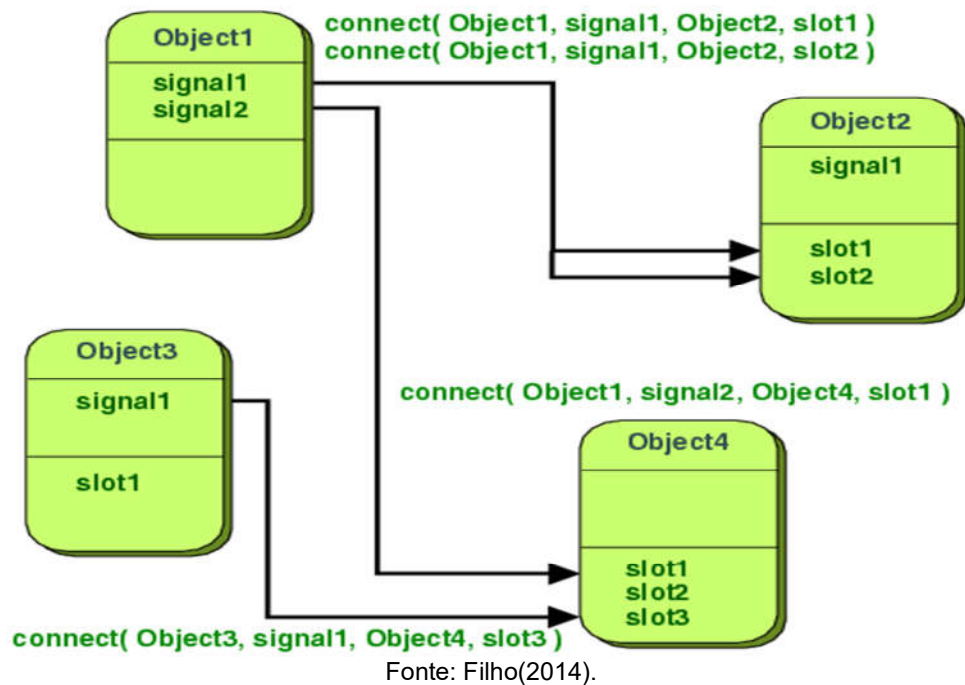
2.1.2 PYQT

Para o desenvolvimento de uma interface é necessário o uso de bibliotecas que seja capaz de gerar uma janela e interagir com o *back-end* do sistema. Uma das principais bibliotecas utilizadas para o uso de janelas com *Python*, é a biblioteca *PyQt* que se encontra na sua quinta versão.

O *PyQt* é capaz de gerar aplicativos *Desktop* para *Linux*, *Windows* e *MAC*, de forma nativa. A vantagem do *PyQt* é que não é necessário fazer mudanças para diferentes sistemas operacionais e o uso de *slots* e *signals* para comunicação. Conforme Filho (2014), o *PyQt* diferente de outras GUIs, não utiliza métodos de *call-backs*, por não ser seguro e não garantir que a comunicação seja feita, para resolver este problema o *PyQt* utiliza *signals* e *slots* (

Figura 4). Caso tenha sido feita alguma mudança em alguma *widget*, um *signal* é enviado pelo elemento e os *slots* são chamados para poder executar as ações correspondentes. Ações que caracterizam os sinais são: o pressionar de um botão, seleção de item e edição de um texto. Os *slots* são os métodos que são chamados por esses sinais e dentro desses métodos o programador determina o que fazer de acordo com a seleção feita pelo usuário

Figura 4- Funcionamento do PyQt5



A biblioteca *PyQt* através de seus recursos de *slot* e *signals*, se tornou uma opção flexível para desenvolvimento de *softwares* com janelas em *Python*, como o apresentado por Da Silva (2016) que desenvolveu um *software* para diagnóstico de consumo de energia elétrica~(Figura 5) e como mesmo relata, para interfaces mais robustas e com funções mais complexas a que melhor se adequou às necessidades dos projetos foi a biblioteca *PyQt5*.

Figura 5- Projeto de monitoramento de energia

	Local	Nome	Potência_Watts	Quantidade	Horas	Energia_
5	Quarto 2	Notebook	60.0	1	2	0.120
6	Quarto 1	Monitor	60.0	2	3	0.180
7	Quarto 2	Monitor	60.0	1	3	0.180
8	Sala	Carregador Cel...	5.0	2	2.0	0.020
9	Sala	Tv Led	60.0	1	0.5	0.030
10	Banheiros	Chuveiro	5500.0	2	0.26	2.860
11	Lavanderia	Maquina Lavar	500.0	1	0.3	0.150
12	Cozinha	Geladeira	200	1	8	1.600

Atualizar Cancelar Adicionar Deletar Deletar Tudo Registrar

Potência Total Instalada: 12.560 kVA Consumo Total(kWh): 166.620 kWh

Número de Cargas: 12 Fatura(R\$): 107.25329

Fonte: Da Silva(2016).

2.1 VISÃO COMPUTACIONAL

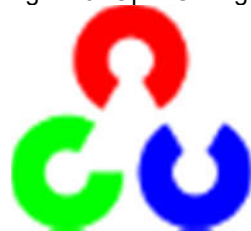
Segundo Victoriano (2012) visão computacional consiste em transformar uma imagem real em algo digital para possibilitar o computador, por meio de técnicas, a compreender o ambiente físico semelhante ao ser humano, conseqüentemente reduzindo a necessidade da visão humana para funções de inspeção.

Para Feliciano (2005), a visão computacional é a aplicação de técnicas de Processamento Digital de Imagens, que tem aplicações separadas em duas abordagens. A primeira refere-se a aplicações de reconhecimento e a segunda a inspeções automatizadas. Em ambas aplicações ocorre a extração das características do objeto a ser detectado em comparação com a imagem atual extraída.

2.1.1 OPENCV

Um das bibliotecas mais reconhecidas na área de visão computacional é o *OpenCV* (Figura 6), que conta com um vasto número de recursos para visão computacional com bibliotecas disponíveis para linguagens *Python* e *C++*. Contando com uma comunidade muito ativa que adiciona cada vez mais recursos e populariza para mais pessoas a visão computacional.

Figura 6- OpenCV logo



Fonte: *OpenCV* (2010)

Conforme Souza (2010), dentro da biblioteca *OpenCV* há diversos módulos como: Processamento de imagens e vídeos, estrutura de dados, controles e vários algoritmos preparados com técnicas de visão computacional. A biblioteca *Opencv* é capaz de funcionar desde a captação de imagem vinda de uma câmera até o resultado de um processamento complexo.

Um sistema de visão computacional utiliza diversas técnicas para detecção de um objeto como o realizado pelo Montanari (2010) que desenvolveu um sistema de

reconhecimento e rastreamento de veículos com filmagens aéreas, onde o robô processa os dados e passa uma informação visual dos veículos como mostrado na *Figura 7*. Todo o projeto foi desenvolvido com os recursos da biblioteca *OpenCV*.

Figura 7-Detecção de Carros



Fonte: Montanari (2015)

2.2 INTELIGÊNCIA ARTIFICIAL (IA)

Na visão de Birgonha (2014), IA é o desenvolvimento de sistemas capazes de demonstrar comportamentos que podem ser atribuídos a uma pessoa, seja em aprendizado ou resolução de algum problema, de acordo com o que foi ensinado à inteligência.

Para Gomes (2016), a inteligência artificial é separada em 4 linhas de pensamento:

1. Sistemas que pensam como humanos: Capazes de pensar semelhante ao ser humano
2. Sistemas que atuam como seres humanos racionalmente: Máquinas que executam funções de forma inteligente
3. Sistemas que pensam racionalmente: Estudo das faculdades mentais usando modelos computacionais
4. Sistemas que atuam racionalmente: Uso de agentes inteligentes

Com o desenvolvimento de máquinas mais potentes por preços mais acessíveis está havendo a popularização da IA. Um dos exemplos que utiliza esta tecnologia é a Autaza Tecnologia, que segundo Izique (2018) utiliza inteligência artificial capaz de fazer de forma autônoma a inspeção visual completa dos automóveis da General Motors (*Figura 8*).

Figura 8-Inspeção de Chassi



Fonte: Izique (2019)

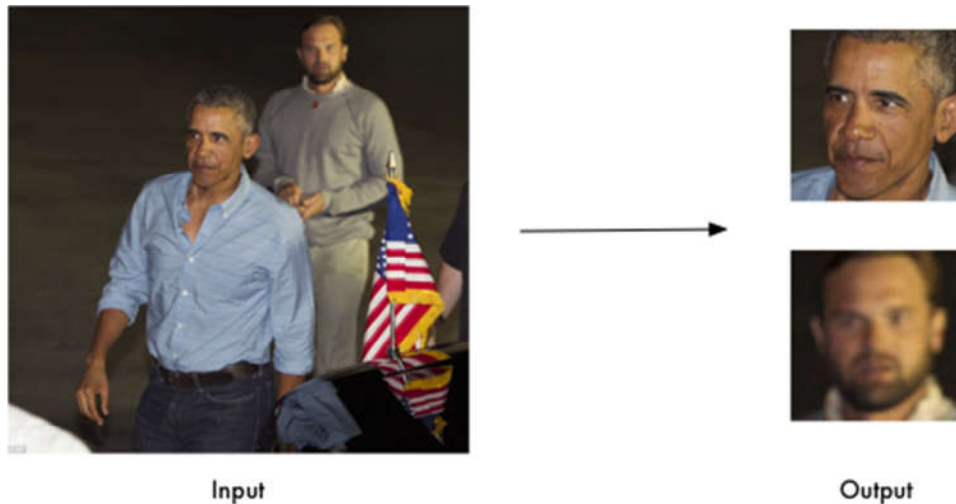
2.2.1 MACHINE LEARNING

Machine Learning é um ramo da inteligência artificial que gera, por meio de aprendizado e treinamento, um sistema capaz de identificar padrões ou tomar decisões baseados no *dataset* (dados inseridos no sistema) que lhe foi desenvolvido.

Segundo Magnus (2018), *Machine Learning* é uma subcategoria de IA, que é capaz de analisar uma grande quantidade de dados por métodos estatísticos específicos em cima dos padrões implantados e o sistema é semelhante a um jogo de xadrez, onde objeto inteligente aprende um conjunto de jogadas e quanto maior a quantidade de jogadas, maior a chance do objeto inteligente vencer.

Um projeto muito conhecido na área de *Machine Learning* é o reconhecimento de faces que foi desenvolvido pelo Geitgey (2019). Por meio de uma grande quantidade de imagens, a inteligência é capaz de rastrear as faces de múltiplas pessoas em várias posições (*Figura 9*).

Figura 9- Face Recognition AI.



Fonte:Geiget (2019)

No exemplo acima, a inteligência recebe o valor de entrada (Input) e verifica em toda a imagem as áreas de interesse (Faces) e ao final dá uma saída com as faces encontradas (Output). Como pôde ser visto, mesmo com a imagem embaçada a inteligência foi capaz de detectar todas as faces

2.2.1.1 TENSORFLOW

Uma das bibliotecas Open Source mais utilizadas para *machine learning* é o *Tensorflow* (

Figura 10), que contém uma ampla variedade de algoritmos. *Tensorflow* tem bibliotecas desenvolvidas para múltiplas linguagens como: Python, C++, Java, Haskell, GO e Rust. Conforme Yegulalp (2019), o *Tensorflow* pode treinar e executar redes neurais profundas para classificação manuscrita de dígitos, reconhecimento de imagem, etc. Todas as execuções podem ser feitas tanto por máquinas locais quanto por GPU, que torna o processamento dos dados mais rápido em uma execução local.

Figura 10-Tensorflow



TensorFlow

Fonte - Tensorflow (2019)

Cavalcante (2019) desenvolveu uma inteligência por meio de aprendizagem profunda, que identifica gestos em libras em tempo real, a pessoa realiza o gesto e as vogais são interpretadas automaticamente (*Figura 11*).

Figura 11- Reconhecimento de gestos em libras



Fonte:Geiget (2019)

2.3 INDÚSTRIA 4.0

Segundo Teles (2019), a indústria 4.0 é uma tendência industrial que envolve avanços na parte de comunicação e informação com o principal objetivo de aumentar as automações e rastrear todos os processos produtivos da fábrica criando o conceito de fábrica inteligente, aumentando a qualidade e reduzindo mão de obra humana.

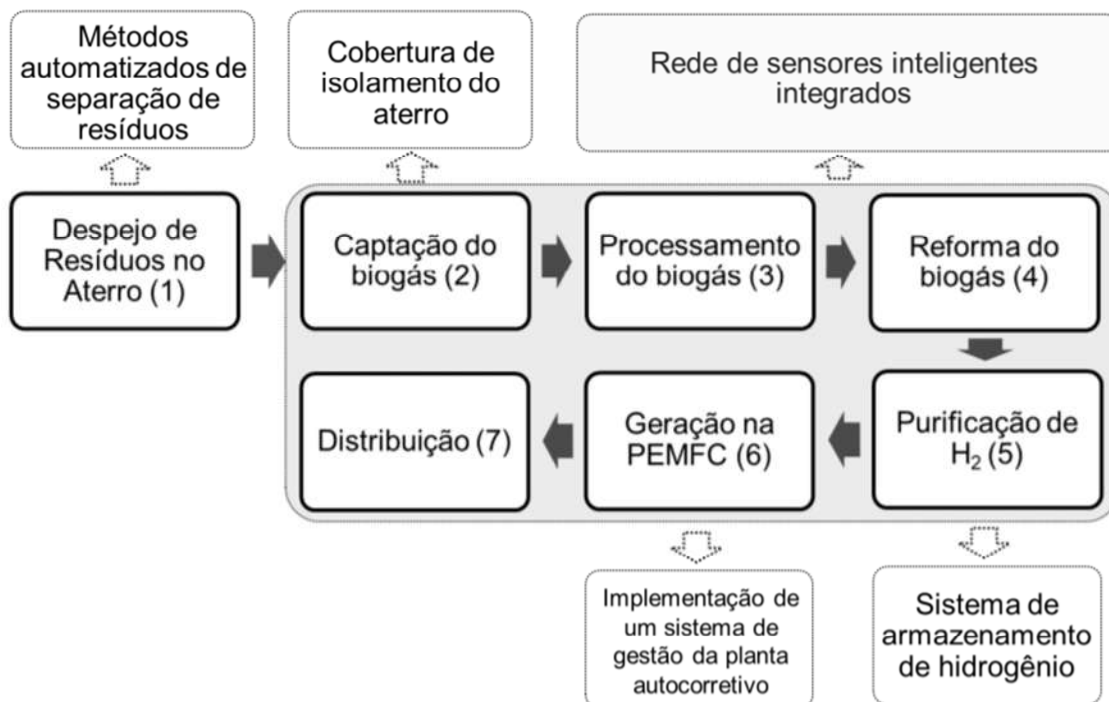
De acordo com Faustino (2016) os princípios são:

1. Virtualização: Prover simulações do processo, para redução de custo e tempo.

2. Descentralização: Autonomia para ajustes sem a intervenção humana.
3. Orientação a serviços: Desenvolvimento de softwares personalizados que são capazes de se comunicar com os dispositivos da linha de produção.
4. Modularidade: Permite a alteração do arranjo físico da linha por meio de do acoplamento e desacoplamento dos módulos produtivos entre si.
5. Interoperabilidade: Capacidade de troca de informação entre os equipamentos.

A indústria 4.0 já está atingindo vários setores da indústria como o apresentado por Cardoso (2018), que implementou um sistema de sensoriamento e supervisão numa Planta de Geração de Energia de Biogás.

Figura 12- Sistema Biogás



Fonte: Cardoso(2018)

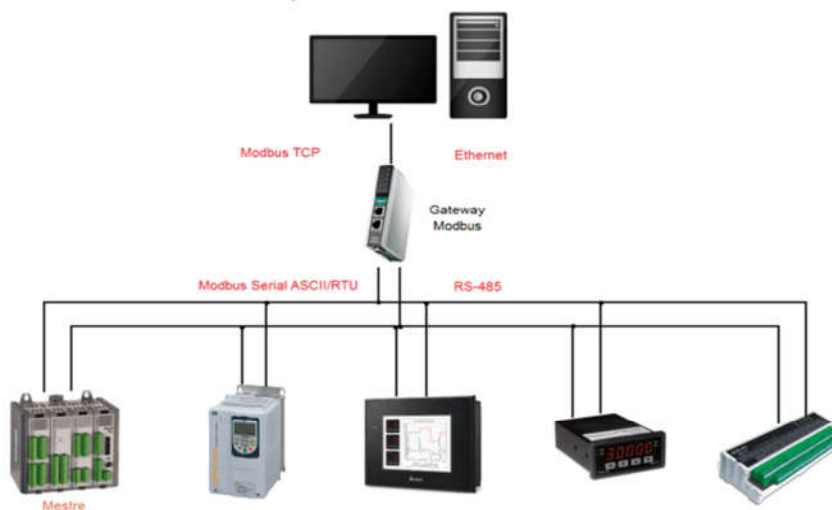
2.4 COMUNICAÇÃO MODBUS

O *Modbus* é um protocolo de requisição-resposta que utiliza um relacionamento mestre-escravo. Em um relacionamento mestre-escravo, a comunicação sempre ocorre em pares. Um dispositivo deve iniciar a requisição e

então aguardar por uma resposta e o dispositivo iniciador (o mestre) é responsável por iniciar cada interação. Tipicamente, o mestre é uma IHM ou sistema SCADA e o escravo é um sensor, controlador lógico programável (CLP) ou controlador programável para automação (CPA). O conteúdo dessas requisições e respostas e as camadas de rede pelas quais essas mensagens são enviadas são definidos pelas diferentes camadas do protocolo (Freitas, 2014).

Segundo Freitas (2014), o *Modbus* funciona por meio de um mestre que irá enviar informações para todos os escravos (Figura 13) e aguardará as informações do escravo solicitado, sejam dados discretos ou numéricos para processamento de algum comando.

Figura 13- Estrutura Modbus



Fonte: Freitas (2014)

Conforme Andrade (2018), o protocolo Modbus é separado em 4 tipos de objetos (*Tabela 1*) *coils*, *discrete input*, *input register* e *holding register*. Para *coils* e *discret input* trabalham apenas com 1 bit por endereço, enquanto o input e o holding funcionam com 16 bits por endereço.

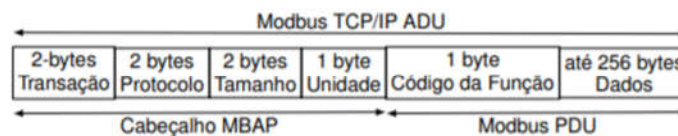
Tabela 1- Modbus: endereçamento

Tipo de objeto	Acesso	Tamanho	Endereçamento
Coils	Ler e Escrever	1-bit	00001-09999
Discrete input	Só Leitura	1-bit	10001-19999
Input Register	Só Leitura	16-bits	30001-39999
Holding Register	Ler e Escrever	16-bits	40001-49999

Fonte: Freitas (2014)

Segundo Ferst (2018), a vantagem do modelo TCP/IP é que para ligação entre equipamentos é feita já utilizando os próprios endereços IP da máquina e a verificação de erro já é feita pelo próprio protocolo TCP, além de ter a escalabilidade, desempenho de 10/100MBps e 1GBps. O protocolo *Modbus* TCP (Figura 14) é separado em duas partes, uma é o cabeçalho MBAP, que contém configurações de protocolo e dados, e a outra *Modbus* PDU onde se localiza o código da função e os dados a serem transmitidos, sendo que o *byte* de transação tem como objetivo emparelhar através de um número identificador. O *byte* de protocolo é utilizado para intercomunicar diferentes protocolos, no caso do *Modbus* TCP esse código é zero. O *byte* de tamanho passa o tamanho do dado a ser transmitido. O *byte* de unidade para possibilitar o uso de dispositivos como: *proxies*, *gateways* e roteadores.

Figura 14- Protocolo Modbus



Fonte: Ferst (2018).

Para poder receber o dado dos escravos é necessário enviar o dado que se deseja acessar. Para isso é necessário enviar ao escravo o código da função (

Tabela 2), para poder identificar qual tipo de registro você deseja ler ou escrever. A área de dados irá retornar ou enviar os dados de acordo com a função.

Tabela 2- Tabela de funções do Modbus

Código	Descrição
1	Leitura de bloco de bits do tipo coil(saída discreta).
2	Leitura de bloco de bits do tipo entradas discretas.
3	Leitura de bloco de registradores do tipo holding.
4	Leitura de bloco de registradores do tipo input.
5	Escrita em um único bit do tipo coil(saída discreta).
6	Escrita em um único registrador do tipo holding.
7	Ler o conteúdo de 8 estados de exceção.
8	Prover uma série de testes para verificação da comunicação e erro internos.
11	Modbus: Obter o contador de eventos.
12	Modbus: Obter um relatório de eventos.
15	Escrita em bloco de bits do tipo coil(saída discreta).
16	Escrita em bloco de registradores do tipo holding.
17	Ler algumas informações do dispositivo.
20	Ler informações de um arquivo.
21	Escrever informações em um arquivo.
22	Modificar o conteúdo de registradores de espera através de operações lógicas.
23	Combina ler e escrever em registradores numa única transação.
24	Ler o conteúdo da fila FIFO de registradores.
43	Identificação do modelo do dispositivo.

Fonte: Freitas (2014).

3 MATERIAIS E DESENVOLVIMENTO DO SISTEMA

O desenvolvimento do projeto ocorreu na plataforma Pycharm. Para isso foi necessário programar toda a interface gráfica e algoritmo por meio de Python utilizando o Windows. O projeto utiliza inteligência artificial para detectar a presença do manual e o *PyQt* é usado como interface do sistema.

As interfaces e o funcionamento principal foram divididos em dois softwares: Qt Designer e Pycharm. O Pycharm foi a IDE selecionada para desenvolvimento de todo o código *Python* e o Qt Designer é responsável pelo design da interface e contém os recursos de imagens, botões, abas e tabelas. Quando o design é finalizado há a opção de exportar o design para diferentes linguagens ao escolher o Python ele exporta em formato .py com o código equivalente ao design desenvolvido.

3.1 MATERIAIS

Para o desenvolvimento do projeto foi necessário usar câmeras para leitura e detecção do sistema, CLP para comunicação com a linha e o PC para interface e execução do sistema conforme (Tabela 3).

Tabela 3- Materiais para projeto

Item	Quantidade	Uso
WebCam	2	Leitura do código de barras e para encontrar o manual
CLP	1	Controle da esteira da linha de produção e leitura dos sensores
Cabo Ethernet	1	Comunicação física entre PC e CLP
PC Desktop	1	Parte central onde o software vai estar funcionando
Teclado	1	Para interação com usuário
Mouse	1	Para interação com usuário
Monitor	1	Mostrar software

Fonte: Próprio autor (2020).

Para o projeto foi necessário apenas comprar as duas *WebCams* pois a empresa já tinha os outros materiais. O modelo escolhido foi a Logitech C270 (

Figura 15), pois ela conta com um foco manual ajustado mecanicamente e uma vez que o projeto está montado, não vai ser mais preciso ajustar o foco.

Figura 15- Webcam C270.



Fonte: Próprio autor (2020).

O CLP escolhido foi o modelo XGB com módulo *Ethernet* (Figura 16). A escolha desse CLP foi devido ao fato de a empresa adotar este CLP como um modelo padrão para a fábrica, e o XGB Ethernet foi escolhido, pois ele possibilita a comunicação *Modbus*.

Figura 16- CLP LSIS XGB Ethernet



Fonte: Próprio autor (2020).

Para teste do sistema foi utilizado o modelo Lenovo *Ideapad 330* com processador i7 de oitava geração, 8GB de memória RAM, 1TB de HD e uma placa de vídeo NVIDIA MX150 para o uso dos recursos do *Tensorflow-GPU* que é capaz de acelerar o treinamento do sistema e a execução da inteligência artificial.

Figura 17- Notebook Ideapad 330.

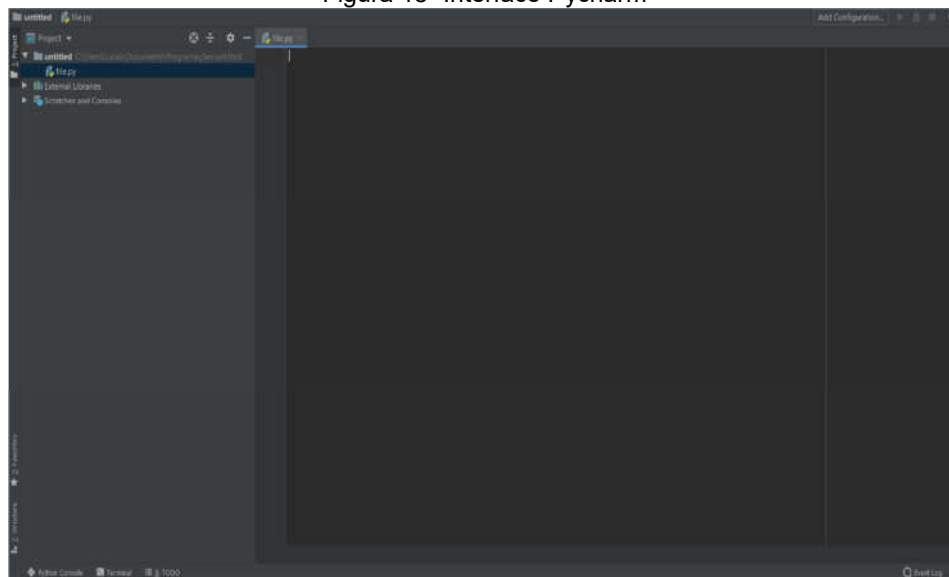


Fonte: Próprio autor (2020)

3.2 CONFIGURAÇÕES INICIAIS

Para iniciar o projeto deve-se abrir o Pycharm Novo Projeto -> Selecionar a versão do Python (3.5 no caso)-> Nome do Projeto e o local, após estes passos aparece uma janela (*Figura 18*) pronta para ser programada.

Figura 18- Interface Pycharm



Fonte: Próprio autor (2020).

O PyPi foi o repositório escolhido para a instalação das bibliotecas do projeto desenvolvido, já que este é o repositório principal da linguagem Python. Há duas maneiras de baixar uma biblioteca no Pycharm: em propriedades ou linha de

comando, este foi escolhido por já baixar a última versão estável, onde para baixar um pacote por prompt de comando basta escrever como demonstrado (na Figura 19).

Figura 19- Comando para Instalação de Biblioteca

```
pip install NOME_DO_PACOTE
```

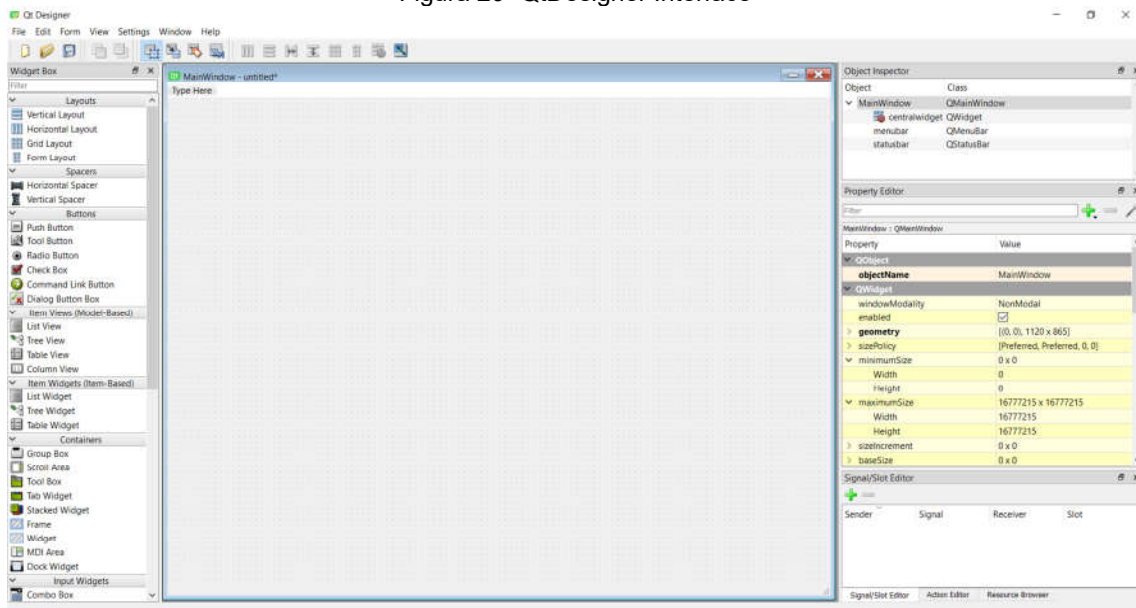
Fonte: Próprio autor (2020).

Para o desenvolvimento do projeto, foi necessário a instalação dos seguintes pacotes:

- *Tensorflow*: Para processamento e criação da inteligência artificial.
- *Opencv-Python*: Biblioteca *OpenCV* para *Python*.
- *PyQt5*: Biblioteca da interface do sistema.
- *Pymodbus3*: Biblioteca para comunicação *Modbus*.
- *Pyzbar*: Biblioteca para reconhecimento de código de barras.
- *PyMysql*: Biblioteca para execução de instruções SQL.

Para a programação do front-end é necessário abrir o programa QtDesigner->Novo Projeto->tipo de janela->nome do projeto e uma tela com uma janela em branco aparece, pronto para ser editado (*Figura 20*).

Figura 20- QtDesigner Interface



Fonte: Próprio autor (2020).

3.2.1 INSTALAÇÃO *TENSORFLOW* API

Após a edição das imagens, é necessário instalar o API de detecção de objetos fornecida pelo *Tensorflow* no link: <https://github.com/tensorflow/models>. É necessário baixar em formato zip e extrair em uma pasta onde o projeto será desenvolvido. Após a extração dos arquivos, os caminhos “models/research”, “models/research/” e “models/research/slim”. No Windows deve selecionar:

1. Entrar no painel de controle.
2. Sistema e segurança.
3. Sistema.
4. Configurações avançadas do sistema.
5. Selecionar a aba de avançados
6. Variáveis do ambiente.
7. Selecionar a opção *Path* e adicionar os caminhos citados acima

Em seguida deve-se baixar o *Protobuf* para a melhoria do tráfego dos dados a serem processados. Para instalação deve acessar o link: <https://github.com/protocolbuffers/protobuf/releases> e salvar o *download* na pasta *research* e executar o comando (Figura 21)

Figura 21- Linha de comando para instalação do *Protobuf*

```
./bin/protoc object_detection/protos/*.proto --python_out=.
```

Fonte: Próprio autor (2020).

3.2.2 INSTALAÇÃO NVIDIA CUDA TOOLKIT E CUDNN

O *Notebook* e PC de teste contam com placa de vídeo NVIDIA CUDA e a empresa fornece algoritmos e recursos para aceleração e melhor desempenho de sistemas que trabalham com inteligência artificial e afins. Já que o CUDA *toolkit* fornece o GPU para que as operações executem de forma dedicada e mais rápida que o próprio CPU do computador através de programação paralela.

Figura 22- NVIDIA CUDA logo



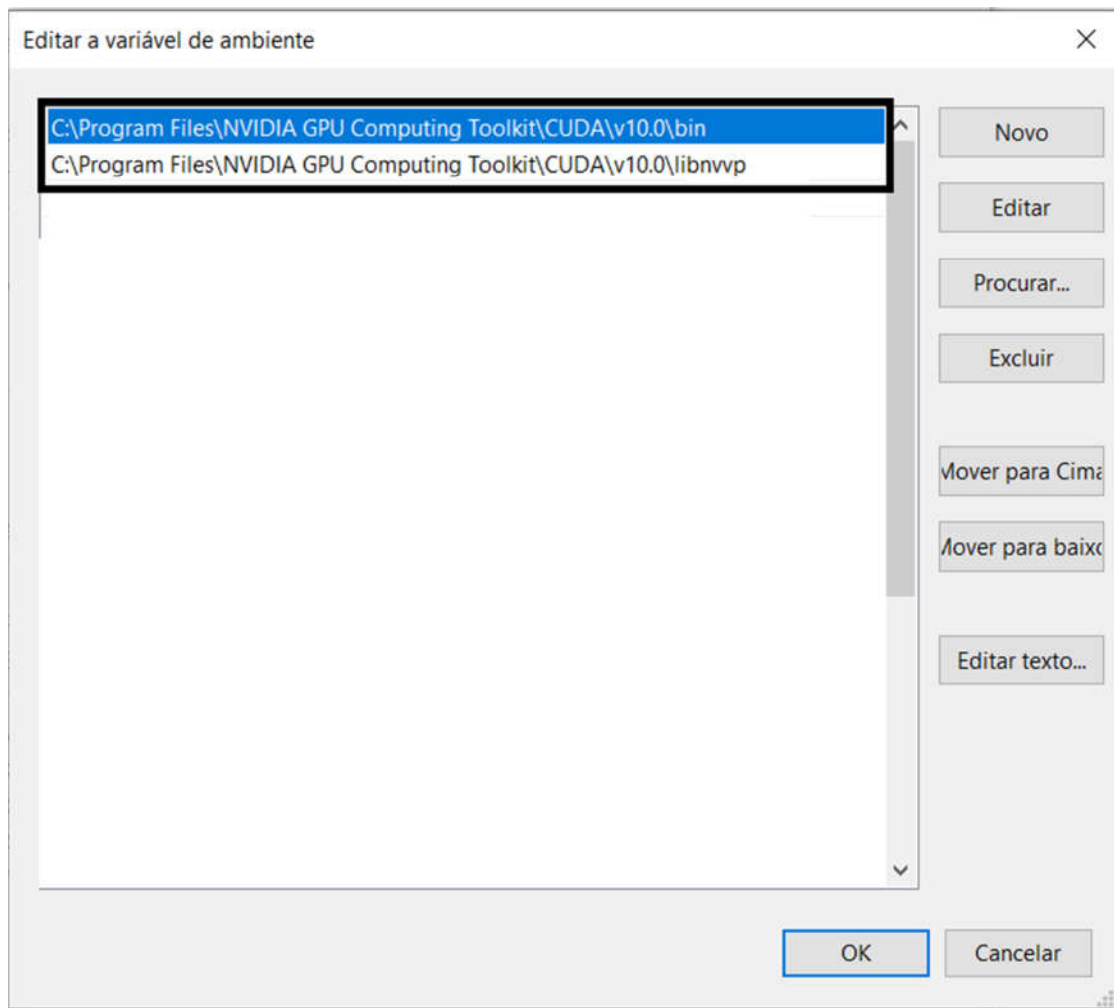
Fonte: CUDA toolkit (2020).

Para a instalação do CUDA TOOLKIT é necessário baixar no site oficial da NVIDIA: <https://developer.nvidia.com/cuda-toolkit>. Atualmente se encontra versão 10.2 e para o projeto foi utilizada a versão 10.0. Após a instalação do CUDA TOOLKIT é necessário instalar CUDNN que é uma biblioteca dedicada para *Deep Learning* que está disponível nesse link: <https://developer.nvidia.com/cudnn>. A versão deve ser a mesma que a do CUDA TOOLKIT. Para instalar o CUDNN deve seguir os seguinte passos:

1. Ao baixar o arquivo *.zip* deve extrair o arquivo.
2. Copiar o arquivo em `/cuda/bin/cudnnVersao.dll` e colar no caminho `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0\bin\`.
3. Copiar o arquivo em `\cuda\include\cudnn.h` e colar em `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0\include\`.
4. Copiar o arquivo em `\cuda\lib\x64\cudnn.lib` e colar em `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0\lib\x64\`.

Ao finalizar todo esse processo é recomendado que, no *Windows*, coloque os caminhos no *Path* (Figura 23) para que possa ser usado no *Python*.

Figura 23- Caminhos CUDA



Fonte: Próprio autor (2020).

3.3 DESENVOLVIMENTO GRÁFICO DA APLICAÇÃO

O QtDesigner usa as bibliotecas Qt e oferece múltiplas opções para a criação da interface gráfica do sistema. Para esta aplicação foram utilizados os seguintes recursos:

- QLabel: Recurso que serve para criação de caixas de texto e caixas de imagens.
- QWidget: Recurso utilizado para criação de tabelas.
- QMenuBar: Recurso que mostra as abas e opções localizadas acima.
- QProgressBar: Mostra o *status* atual do sistema.
- Horizontal Spacer: Faz o espaçamento horizontal da interface.
- Vertical Spacer: Faz o espaçamento vertical da interface.

- Vertical Layout: Alinha os elementos verticalmente.
- Horizontal Layout: Alinha os elementos horizontalmente.

Ao finalizar a interface do sistema é necessário escolher qual linguagem de programação vai ser utilizada ao executar a linha de comando (*Figura 24*) e toda a interface é decodificada para a linguagem equivalente escolhida, com todas as configurações de interface e *back-end* configuradas.

Figura 24- Linha de Comando para Geração de Código

```
pyuic5 -x arquivodesign.ui -o saida.(linguagem a ser programada)
```

Fonte: Próprio autor (2020).

No sistema desenvolvido foi utilizado o *Python*, então foi utilizada a extensão *.py* para exportar a interface equivalente na linguagem do projeto. Quando o arquivo é exportado ele cria um algoritmo em *Python* (*Figura 25*) que é dividido em três partes:

- *setupUI()*: Função pertencente à classe *MainWindow()*. É responsável pelas declarações e configurações de layout
- *retranslateUI()*: Função pertencente à classe *MainWindow()*. É onde os nomes e títulos dos elementos que compõem a interface são configurados
- *__main__*: Função responsável por chamar e executar a interface.

As bibliotecas principais para o desenvolvimento de interface são: *QtCore* que tem as funções básicas de sistema, *QtGui* que contém recursos de interface e *QtWidgets* que tem os elementos gráficos do sistema.

Figura 25- Código Gerado em Python

```

from PyQt5 import QtCore, QtGui, QtWidgets <- Biblioteca para interface

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):...

    def retranslateUi(self, MainWindow):...

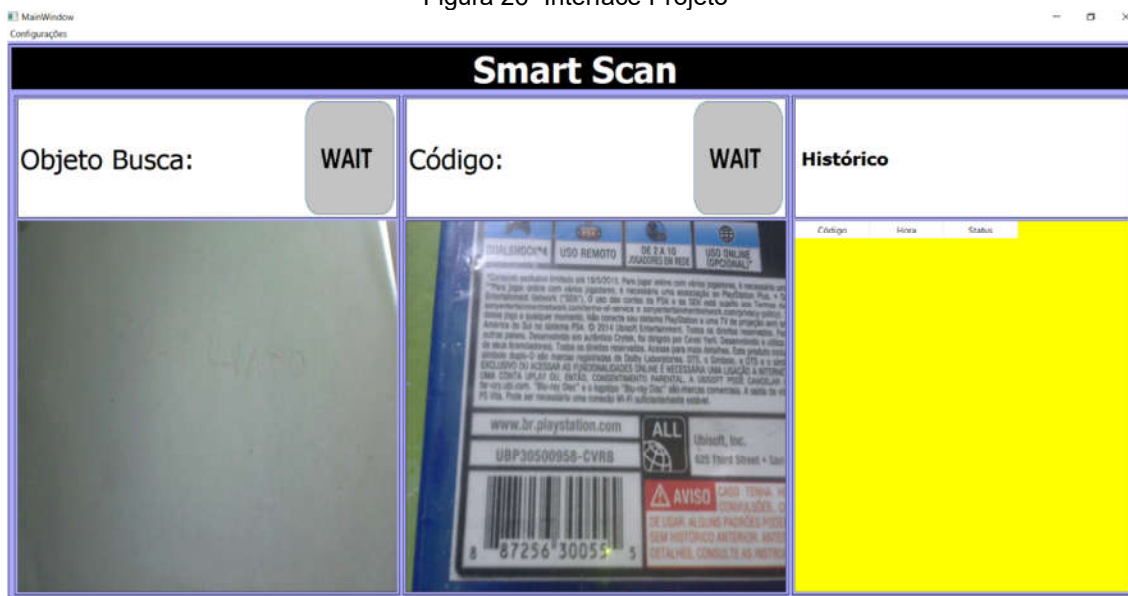
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())

```

Fonte: Próprio autor (2020).

O projeto tem uma tela principal que é dividida em partes (Figura 26). A primeira localizada à esquerda é responsável por rastrear o objeto a ser encontrado, a segunda faz a leitura do código do objeto e a terceira gera um histórico com 3 opções: código, hora de detecção e o *status* para determinar que o objeto foi detectado ou não. Para os textos e imagens foi utilizado o recurso *QLabel* e para a tabela foi utilizado o *QTableWidget*

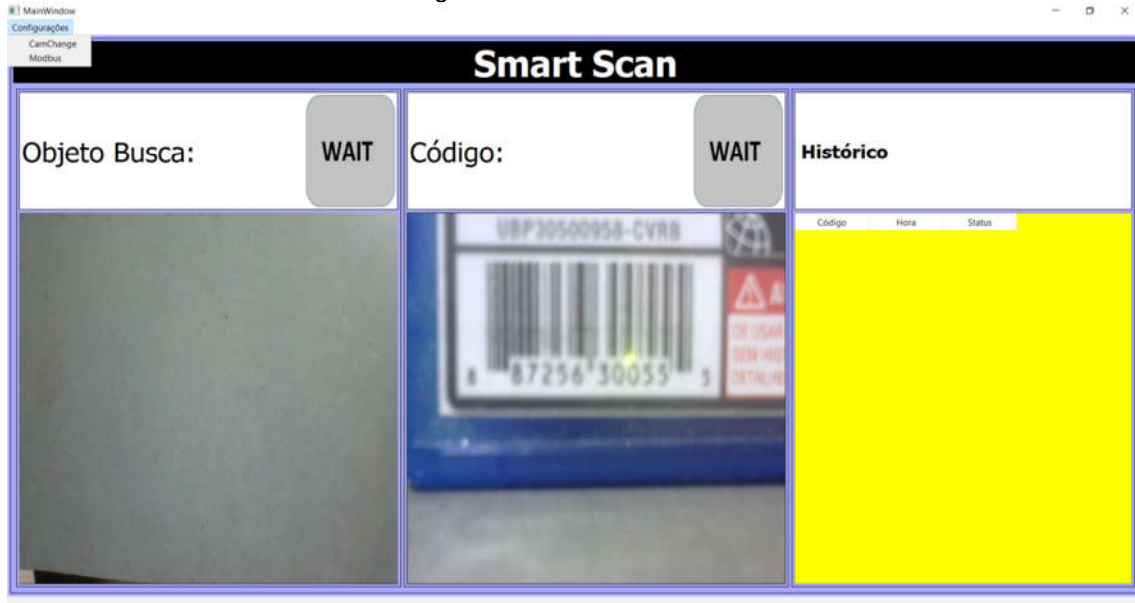
Figura 26- Interface Projeto



Fonte: Próprio autor (2020).

Na interface principal foi implementada uma aba de Configurações e nela há duas opções: *CamChange* e *Modbus*, que são ferramentas que dependem diretamente da ação do usuário.

Figura 27- Abas da interface.

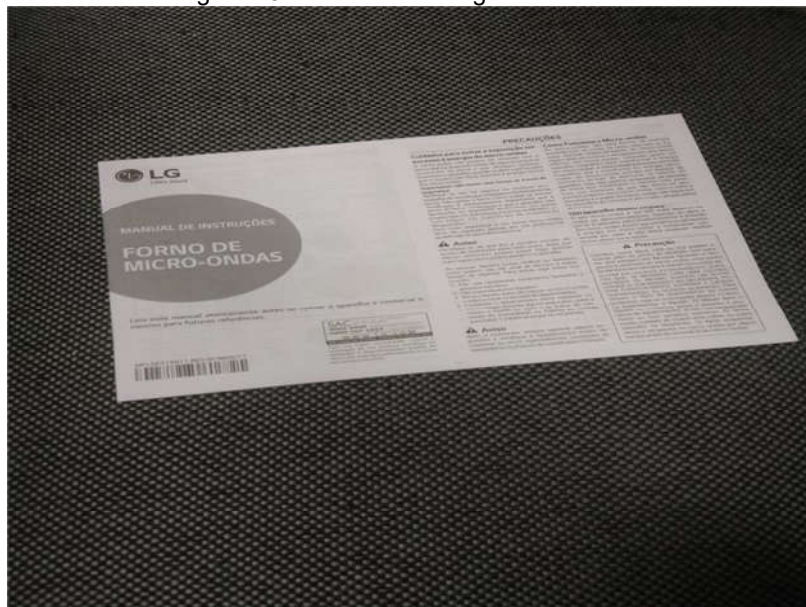


Fonte: Próprio autor (2020).

3.4 DESENVOLVIMENTO DO SISTEMA

Para iniciar o desenvolvimento da parte da inteligência artificial é necessário gerar um banco de imagens (Figura 28), então a primeira ação é tirar múltiplas fotos do objeto em diferentes localizações e posições semelhantes aos esperados dentro da caixa, para melhorar a capacidade de detecção do objeto.

Figura 28- Amostra de imagem do manual



Fonte: Próprio autor (2020).

3.4.1 LABELIMG

O Labelmg funciona como um *software* de edição que permite o usuário demarcar onde o objeto a ser detectado está localizado. Para abrir o Labelmg é necessário ter o *Python* instalado em seu *prompt* de comando e executar conforme a Figura 29. O Labelmg está disponível no *Github* em: <https://github.com/tzutalin/labelImg>.

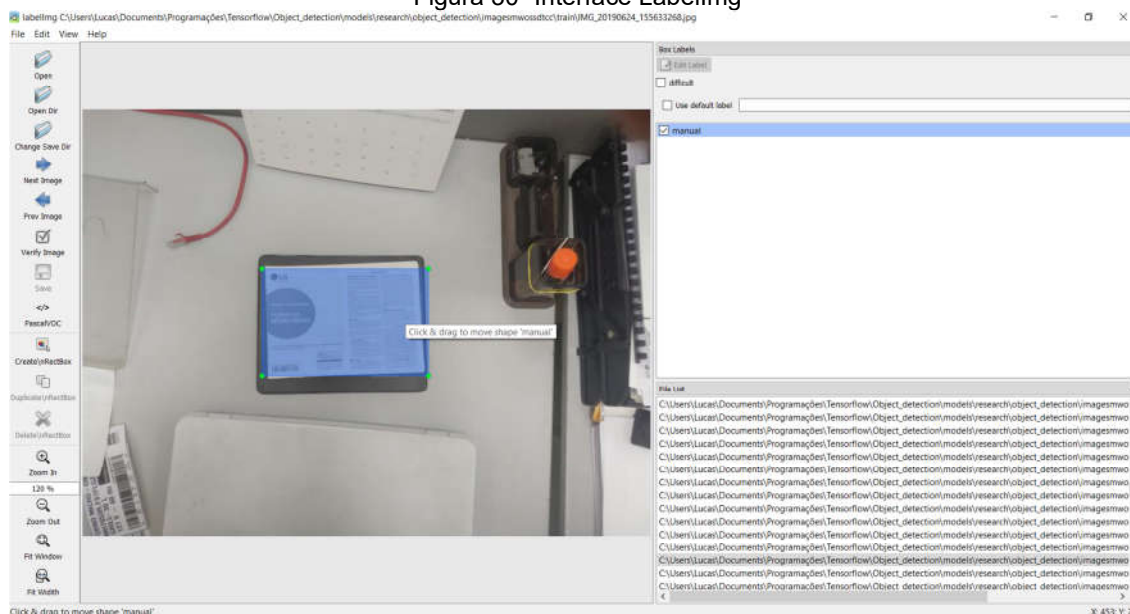
Figura 29- Linha de comando para abrir Labelmg

```
>python labelImg.py
```

Fonte: Próprio autor (2020).

Ao abrir o Labelmg (Figura 30) deve ser selecionado duas opções antes de começar: o diretório onde estão localizadas as imagens (*Open dir*) e onde deve ser salvo o arquivo de saída (*Change Save Dir*). Após as configurações serem escolhidas, a primeira imagem do diretório é mostrada. Após as opções serem escolhidas, o usuário deve marcar onde o objeto está localizado, na interface, e sinalizar qual o nome do objeto em *box labels*.

Figura 30- Interface Labelmg



Fonte: Próprio autor (2020)

Ao término da edição é necessário selecionar a opção “*save*” que vai gerar um arquivo em formato *Pascal* (Figura 31) que gerará informações sobre: Dimensão da imagem, localização do objeto, nome dos objetos e o caminho da imagem. No projeto foi feito cerca de edição em 700 imagens em várias condições de ambiente,

local, posição e iluminação. As imagens e saídas estão salvas em dois diretórios no *Tensorflow API* no diretório “*research/objectdetection/imagens/test*” e “*research/objectdetection/imagens/train*”.

Figura 31- Formato Pascal

```
<annotation>
  <folder>train</folder>
  <filename>b4.jpg</filename>
  <path>C:\Users\Lucas\Documents\Programações\Tensorflow\Object_detection\models\research\object_detection\image\mossdtcc\train\b4.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>800</width>
    <height>600</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>label</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>346</xmin>
      <ymin>257</ymin>
      <xmax>487</xmax>
      <ymax>336</ymax>
    </bndbox>
  </object>
</annotation>
```

Fonte: Próprio autor (2020).

Para o projeto, os diretórios de entrada e saída ficaram no mesmo caminho para facilitar a conversão dos arquivos e foi feita a separação de duas amostras diferentes: uma para treino e outra para teste, cerca de 20 imagens.

3.4.2 TREINAMENTO

Para o funcionamento da inteligência artificial é necessário treiná-la. Com as imagens já separadas e os arquivos formatados em *Pascal*, agora é necessário formatar os dados para que seja entendido pelo *Tensorflow*.

3.5 FORMATANDO OS DADOS

Para facilitar a formatação de dados Tanner (2019) disponibilizou alguns algoritmos para suporte do preparo do treinamento da inteligência. O *Tensorflow* interpreta os dados das imagens em formato CSV. Para isso há um algoritmo em *Python* que realiza essa conversão de *Pascal* para *CSV* (Figura 32). O usuário deve colocar no arquivo que tem as pastas ‘*Test*’ e ‘*Train*’, para acessar todos os dados gravados pelo *LabelImg*

Figura 32- Algoritmo para conversão em formato CSV.

```
# New:
def main():
    for folder in ['train', 'test']:
        image_path = os.path.join(os.getcwd(), ('images/' + folder))
        xml_df = xml_to_csv(image_path)
        xml_df.to_csv(('images/'+folder+'_labels.csv'), index=None)
        print('Successfully converted xml to csv.')
```

Fonte: Tanner (2019).

Após a execução do algoritmo, dois arquivos são gerados na saída: um arquivo para treino e outro para teste já em formato CSV (Figura 33). O arquivo fica na pasta raiz onde estão as duas pastas com os dados das imagens.

Figura 33- Formato CSV

filename	width	height	class	xmin	ymin	xmax	ymax
a1.jpg	1280	960	label	447	379	782	660
a10.jpg	960	1280	label	521	573	653	758
a10.jpg	960	1280	label	420	446	580	584
a11.jpg	960	1280	label	407	525	547	691
a13.jpg	1280	720	label	658	305	707	392
a13.jpg	1280	720	label	719	349	806	395
a14.jpg	1280	960	label	559	308	611	412

Fonte: Próprio autor (2020).

O formato de dado lido pelo *Tensorflow* é o *.record*, que são arquivos de armazenamento de dados binários destinados à aplicações em *Tensorflow*. Para conversão do arquivo CSV para *.record* é usado um outro algoritmo feito por Tanner (2019), o *generate_tfrecord.py* que obtém os arquivos de imagem e o arquivos das imagens em CSV e transforma em arquivo binário para *Tensorflow*. O usuário deve modificar o campo dentro da função *class_text_to_int* (Figura 34) colocar o nome do objeto que você vai detectar (manual no caso do projeto). O nome do objeto deve ser igual ao colocado no *LabelImg*.

Figura 34- Trecho de código *generatetfrecord.py*

```
def class_text_to_int(row_label):
    if row_label == 'manual':
        return 1

    else:
        None
```

Fonte: Tanner (2020).

Após a edição, deve ser executado o comando inserindo os caminhos da pasta, a localização do arquivo CSV e o nome do arquivo de saída. Após seguir

esses passos os dados já estão pronto e formatados para o treinamento da inteligência artificial.

Figura 35- Linha de comando para gerar arquivo .record

```
python generate_tfrecord.py --csv_input=images/train_labels.csv --image_dir=images/train --output_path=train.record
```

Fonte: Próprio autor (2020).

Após gerar o arquivo *.record* na pasta *objectdetection*, criamos uma pasta chamada *treino* e dentro dela criamos um arquivo com nome *labelmap.pbtxt* (Figura 36), descrevendo quais são os objetos a serem detectados, o valor de id dos objetos devem estar de acordo com o *generate_tfrecord.py*.

Figura 36- labelmap.pbtxt

```
item {  
  id: 1  
  name: 'manual'  
}
```

Fonte: Próprio autor (2020).

3.6 TREINANDO A INTELIGÊNCIA ARTIFICIAL

Para treinar uma inteligência artificial há duas opções: treinar uma inteligência do zero ou treinar em cima de um modelo que já foi testado e aprovado. Para acelerar o projeto, utilizamos um modelo pré-treinado para a detecção do manual. O *Tensorflow* oferece uma lista de modelos prontas com características como confiabilidade, tempo de detecção e tipo de saída se vai ser máscara ou *box*. O arquivo https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md está disponível em:

Para o projeto, foi escolhido dois modelos para detecção de manual: o *ssd_inception_v2_coco* e *faster_rcnn_inception_v2_coco*, onde ambos são rápidos e precisos para o projeto de detecção. Ao baixar o arquivo no site, extraímos na pasta do *Tensorflow API* “/research/objectdetection”.

Em seguida, foi necessário criar uma pasta de treino em *object detection*. Renomeamos a pasta com o nome do modelo a ser usado e colocamos na pasta de treino. Abrimos o arquivo *.config* (Figura 37) e editamos para que funcione corretamente o campo *num_classes* e a quantidade de objetos que serão detectados (devem estar de acordo com a quantidade do *labelmap.pbtxt*), o

fine_tune_checkpoint que designa onde o modelo baixado está localizado (deve colocar no final /model.ckpt). O *train_input_reader* se refere às imagens que serão treinadas e o *eval_input_reader* para amostras de testes que serão comparadas e no campo *label_map_path* deve colocar o caminho onde está localizado o arquivo *labelmap.pbtxt* e o *input_path* deve ser inserido o caminho do arquivo *.record*.

Figura 37- Trecho do arquivo *.config*

```
num_classes: 1
fine_tune_checkpoint: "C:/Users/Lucas/Documents/Programa\303\247\303\265es/Tensorflow/Object_detection/models/research/object_detection/ssd_inception_v2_coco_2018_01_28/model.ckpt"
train_input_reader {
  label_map_path: "C:/Users/Lucas/Documents/Programa\303\247\303\265es/Tensorflow/Object_detection/models/research/object_detection/trainingmwo/ssd/labelmap.pbtxt"
  tf_record_input_reader {
    input_path: "C:/Users/Lucas/Documents/Programa\303\247\303\265es/Tensorflow/Object_detection/models/research/object_detection/trainmwo/ssd/record"
  }
}
eval_input_reader {
  label_map_path: "C:/Users/Lucas/Documents/Programa\303\247\303\265es/Tensorflow/Object_detection/models/research/object_detection/trainingmwo/ssd/labelmap.pbtxt"
  shuffle: false
  num_readers: 1
  tf_record_input_reader {
    input_path: "C:/Users/Lucas/Documents/Programa\303\247\303\265es/Tensorflow/Object_detection/models/research/object_detection/testmwo/ssd/record"
  }
}
```

Fonte: Próprio autor (2020).

Na pasta *object_detection* há um arquivo chamado *train.py* que é responsável por fazer o treinamento da inteligência artificial. Como parâmetro, ele precisa saber a pasta de treinamento e a localização do arquivo *.config* (Figura 38). O algoritmo demora cerca de 30 a 60 segundos para começar a rodar caso não tenha erros.

Figura 38- Linha de comando para treinar a inteligência

```
python train.py --logtostderr --train_dir=diretorio paratreinamento --pipeline_config_path=arquivo.config
```

Fonte: Próprio autor (2020).

Ao treinar as imagens, há uma comparação entre todas de modo a fazer a inteligência achar partes comuns entre todas as imagens e entender as diferentes condições em que o objeto irá se submeter (e prever outras), mas ele não realiza este processo de forma instantânea. Ele realiza passos (*steps*) e a cada passo a inteligência entende cada vez mais sobre o objeto, porém deve tomar cuidado ao treinar demais, pois isso pode causar o *overfitting*, situação em que a inteligência deixa de “aprender” e apenas “decora”, o que pode dificultar na detecção do objeto em situações que não foram inseridas no banco de imagens.

Por sua vez, as avaliações podem começar a serem feitas quando o valor de *loss* (variável que determina o quão ruim foi a leitura de uma certa imagem durante o treinamento), começa a convergir. No algoritmo *train.py*, como saída, é mostrado a quantidade do *loss*, o número do *step* e a velocidade para processar um passo.

Figura 39- Saída train.py

```
I0419 00:44:43.966106 14164 learning.py:507] global step 10571: loss = 1.2681 (1.437 sec/step)
I0419 00:44:45.965842 14164 learning.py:507] global step 10572: loss = 1.9685 (2.000 sec/step)
I0419 00:44:47.981010 14164 learning.py:507] global step 10573: loss = 1.6106 (2.015 sec/step)
I0419 00:44:49.418184 14164 learning.py:507] global step 10574: loss = 1.3091 (1.406 sec/step)
I0419 00:44:51.433351 14164 learning.py:507] global step 10575: loss = 1.6616 (2.000 sec/step)
```

Fonte: Próprio autor (2020).

Para diferentes modelos pode haver uma mudança no início de *loss* que pode começar desde 96 até 128. Se seu banco for coerente e estiver corretamente indicado e com uma quantidade razoável (acima de 500 e em diferentes ambientes e condições) ele irá convergir. Neste projeto houve uma convergência em 0.6 de *loss* e houve mínimas variações após 16012 passos. O algoritmo *train.py* salva a inteligência a cada vinte minutos, ou seja, a cada vinte minutos temos a oportunidade de testar a inteligência para verificar se está no ponto ideal, no *underfitting* (situação em que o modelo não foi treinado o suficiente e suas predições tem uma baixa taxa de acerto) ou no *overfitting* (situação em que o modelo deixa de aprender o que deve ser detectado e começa a “decorar” as imagens o que prejudica na taxa de acerto do modelo). O arquivo de saída é salvo (Figura 40) dentro da pasta de treinamento indicado no parâmetro *trains_dir*.

Figura 40- Pasta de treinamento

checkpoint	06/01/2020 07:13	Arquivo	1 KB
events.out.tfevents.1578272630.LAPTOP-1...	06/01/2020 07:19	Arquivo LAPTOP-1...	193.122 KB
graph.pbtxt	05/01/2020 22:03	Arquivo PBTXT	19.150 KB
model.ckpt-10459.data-00000-of-00001	06/01/2020 04:03	Arquivo DATA-000...	208.192 KB
model.ckpt-10459.index	06/01/2020 04:03	Arquivo INDEX	53 KB
model.ckpt-10459.meta	06/01/2020 04:03	Arquivo META	10.684 KB
model.ckpt-10750.data-00000-of-00001	06/01/2020 04:13	Arquivo DATA-000...	208.192 KB
model.ckpt-10750.index	06/01/2020 04:13	Arquivo INDEX	53 KB
model.ckpt-10750.meta	06/01/2020 04:13	Arquivo META	10.684 KB
model.ckpt-14843.data-00000-of-00001	06/01/2020 06:33	Arquivo DATA-000...	208.192 KB
model.ckpt-14843.index	06/01/2020 06:33	Arquivo INDEX	53 KB
model.ckpt-14843.meta	06/01/2020 06:33	Arquivo META	10.684 KB
model.ckpt-15135.data-00000-of-00001	06/01/2020 06:43	Arquivo DATA-000...	208.192 KB
model.ckpt-15135.index	06/01/2020 06:43	Arquivo INDEX	53 KB
model.ckpt-15135.meta	06/01/2020 06:43	Arquivo META	10.684 KB
model.ckpt-15428.data-00000-of-00001	06/01/2020 06:53	Arquivo DATA-000...	208.192 KB
model.ckpt-15428.index	06/01/2020 06:53	Arquivo INDEX	53 KB
model.ckpt-15428.meta	06/01/2020 06:53	Arquivo META	10.684 KB
model.ckpt-15720.data-00000-of-00001	06/01/2020 07:03	Arquivo DATA-000...	208.192 KB
model.ckpt-15720.index	06/01/2020 07:03	Arquivo INDEX	53 KB
model.ckpt-15720.meta	06/01/2020 07:03	Arquivo META	10.684 KB
model.ckpt-16012.data-00000-of-00001	06/01/2020 07:13	Arquivo DATA-000...	208.192 KB
model.ckpt-16012.index	06/01/2020 07:13	Arquivo INDEX	53 KB
model.ckpt-16012.meta	06/01/2020 07:13	Arquivo META	10.684 KB
pipeline.config	05/01/2020 22:02	XML Configuration...	5 KB
ssd_inception_v2_coco.config	05/01/2020 21:56	XML Configuration...	5 KB

Fonte: Próprio autor (2020)

Conforme mostrado na Figura 40 a inteligência tem como formato de arquivo *model.ckpt*-(número do passo em que foi feito o *checkpoint*) e é dividido em três partes *.data*, *.index* e *.meta*.

3.7 TESTANDO A INTELIGÊNCIA ARTIFICIAL

Com os arquivos de inteligência gerados, deve-se usar um algoritmo que já vem na API do *Tensorflow* o *export_inference_graph.py*, para executar este algoritmo será necessário passar a localização do arquivo *.config*, o arquivo *model* que vai ser testado com o número do passo (*step*) que vai ser usado e o diretório de saída da inteligência.

Figura 41- Linha de comando para exportar inteligência

```
python export_inference_graph.py --input_type image_tensor --pipeline_config_path training/arquivo.config --trained_checkpoint_prefix training/model.ckpt-XXXX --output_directory saída
```

Fonte: Próprio autor (2020).

Os arquivos para teste são gerados dentro da pasta, que tem formato semelhante ao *models* baixados anteriormente, com a diferença de que somente o objeto que treinado é que vai ser identificado.

Figura 42- Arquivo gerado pronto para teste.

📁 saved_model	07/01/2020 01:33	Pasta de arquivos	
📄 checkpoint	07/01/2020 01:33	Arquivo	1 KB
📄 frozen_inference_graph.pb	07/01/2020 01:33	Arquivo PB	52.728 KB
📄 model.ckpt.data-00000-of-00001	07/01/2020 01:33	Arquivo DATA-000...	52.117 KB
📄 model.ckpt.index	07/01/2020 01:33	Arquivo INDEX	18 KB
📄 model.ckpt.meta	07/01/2020 01:33	Arquivo META	1.615 KB
📄 pipeline.config	07/01/2020 01:33	XML Configuration...	5 KB

Fonte: Próprio autor (2020).

Para realizar os testes, desenvolvemos um algoritmo que vai avaliar as imagens de teste para verificar se a detecção é viável para ser implantada na fábrica. Para realizar o teste, basta inserir no arquivo o nome do diretório a ser utilizado e onde está localizado a sequência de imagens. É de preferência do autor analisar imagem por imagem a detecção. Na saída do programa é mostrada a imagem detectada e o quadro de probabilidade que destaca a chance de ser um objeto de interesse.

Figura 43- Saída do algoritmo de avaliação



Fonte: Próprio autor (2020).

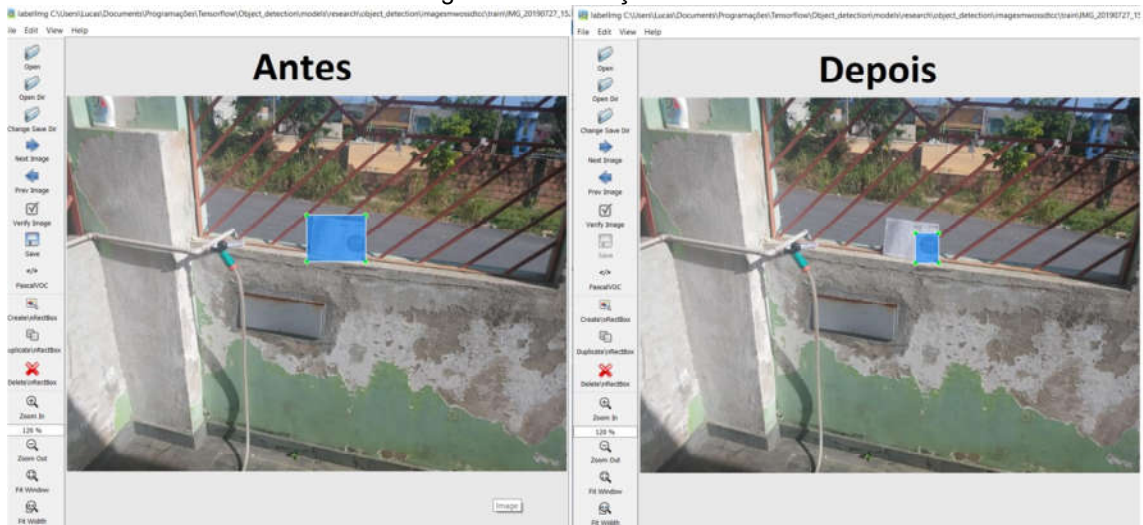
3.8 COMPARATIVO SSD VS FASTER RCNN

Foi feita a geração de dois modelos de detecção de manual, um baseado em SSD e outro em *Faster RCNN*, onde alguns parâmetros foram avaliados para a escolha. Ao realizar os testes, foram notados alguns problemas, como: falso positivo

e falso negativo, onde os falsos negativos se deram devido aos problemas de posicionamento do manual, já que ele estava embaixo do prato e completamente coberto não foi possível fazer a detecção e os falsos positivos aconteciam com frequência no modelo *Faster RCNN* em que etiquetas próximas do manual eram consideradas como manual. Outro problema causado principalmente pelo SSD é que todo o objeto ou quase todo deveria estar a mostra para possibilitar a detecção do sistema.

Como solução para detecção de objetos do SSD foi feita uma mudança no *ROI* do manual, assim parte do manual ficaria sempre amostra para a câmera. Para evitar problema de falso positivo foi implementado o novo objeto a ser detectado, as *labels*, o problema persistiu no *Faster RCNN* que ainda continha falsos positivos para qualquer papel em que houvesse texto, ou seja, a mudança do *ROI* serve como solução tanto para o *Faster RCNN* quanto para SSD. Para fazer essa mudança, todo processo teve de ser refeito com o novo *ROI* (Figura 44) que demarca a *logo* da empresa e a parte onde está escrito manual de usuário.

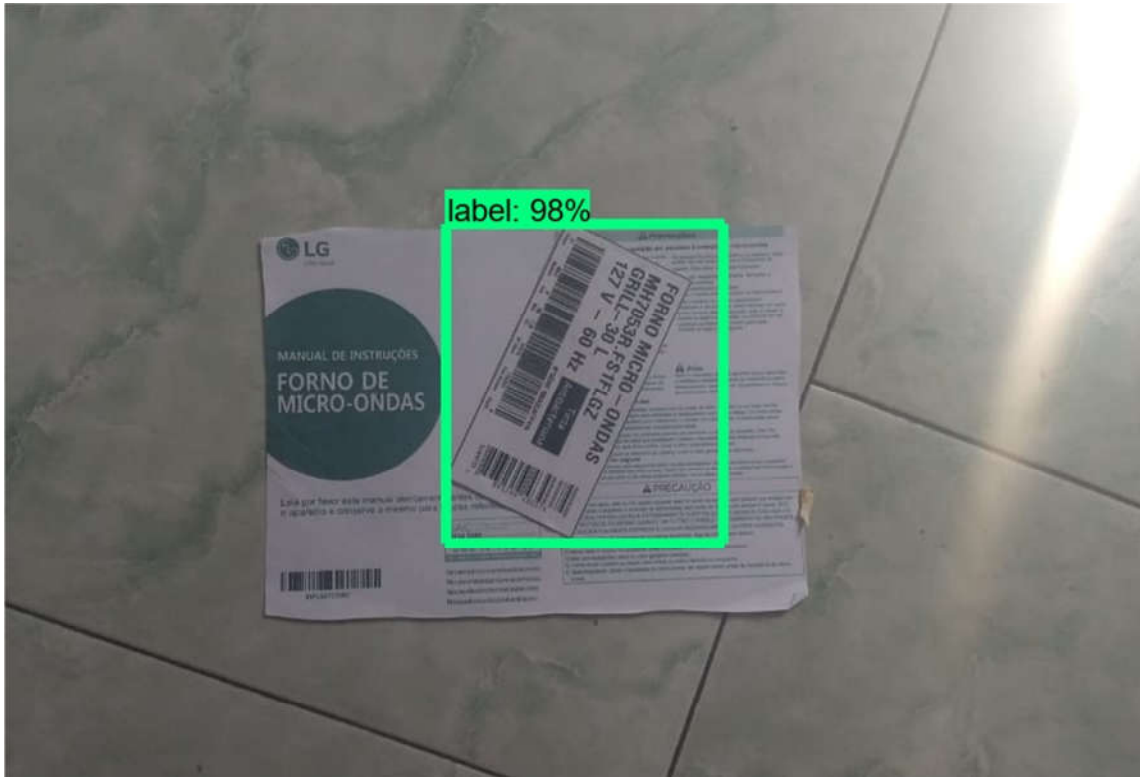
Figura 44- Mudança de *ROI*



Fonte: Próprio autor (2020).

Após refazer os processos e mudar as imagens de teste, houve melhora da *Faster RCNN* em questão de falsos positivos na parte de texto, porém há uma grande variedade de falsos positivos no *Faster RCNN* e falsos negativos com a *label* implementada (Figura 45).

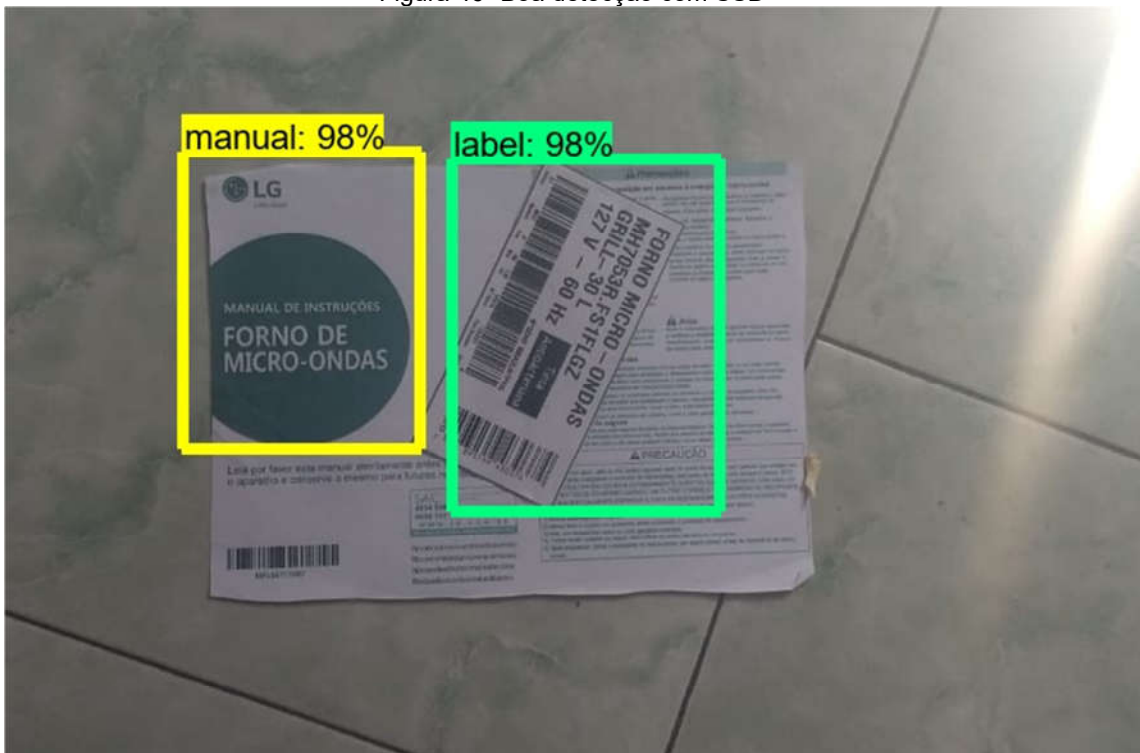
Figura 45- Falso negativo do manual em *Faster RCNN*



Fonte: Próprio autor (2020).

Houve uma grande melhoria por parte do SSD que agora consegue identificar de forma clara as *labels* e o manual (Figura 46).

Figura 46- Boa detecção com SSD



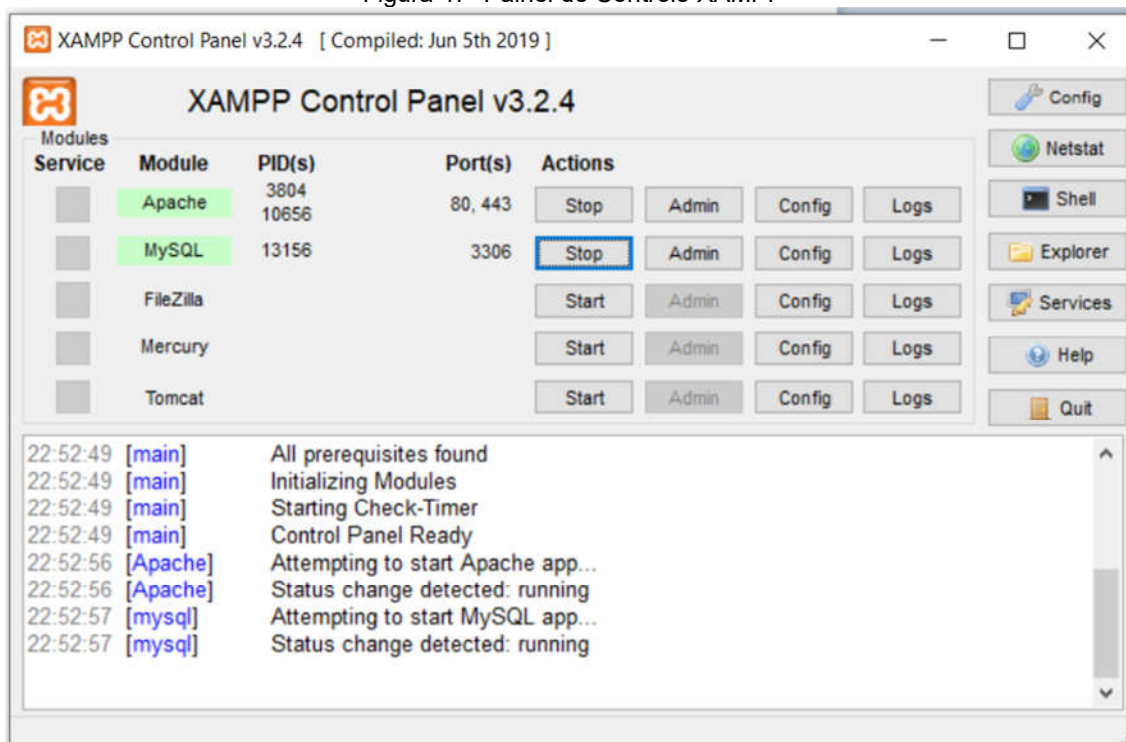
Fonte: Próprio autor (2020).

O sistema deve fazer a detecção em tempo real e fizemos um algoritmo que executa e captura as imagens em tempo real. Para o *Faster RCNN* o tempo para processamento de cada imagem demora cerca de 350 milissegundos e para *SSD* levou apenas 40 milissegundos de processamento por imagem, validando o *SSD* como o modelo selecionado para teste

3.9 BANCO DE DADOS

O banco de dados usado foi o *MariaDB* que é utilizado pelo XAMPP e disponível gratuitamente no site: https://www.apachefriends.org/pt_br/index.html. Ao baixar deve instalar através do executável. Ao finalizar, deve-se abrir o *software* que mostrará um painel de controle que mostra as opções de serviços *web*. Para ativar o banco de dados de selecionar *start* nas opções de *Apache* e *MySql*.

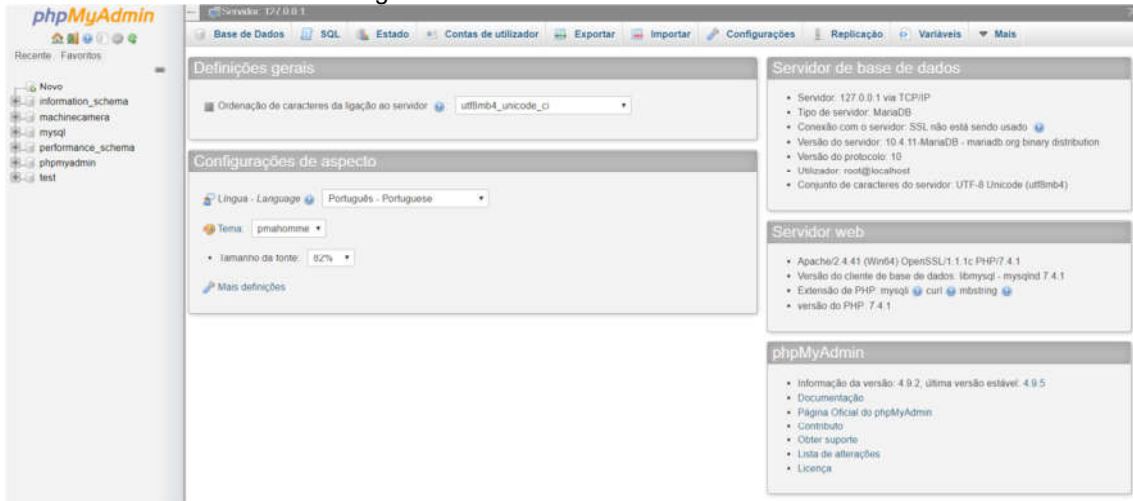
Figura 47- Painel de Controle XAMPP



Fonte: Próprio autor (2020).

Ao habilitar as opções o usuário deve em qualquer navegador acessar o *link*: <http://localhost/phpmyadmin/> que mostrará a interface (Figura 48) do banco de dados. Para adicionar uma nova base de dados basta escolher a opção '+ novo' localizado no canto superior direito.

Figura 48- Interface do banco de dados



Fonte: Próprio autor (2020).

O banco de dados do sistema foi nomeado como *machinecamera* e tem uma tabela de histórico com quatro colunas (Figura 49) com as seguintes funções:

- **Id:** Uma identificação única para cada dado é uma chave primária de identificação e ela faz autoincremento a cada registro
- **Código:** Código que foi lido. É uma variável do tipo *varchar* de 25 posições e é uma chave única.
- **Hora:** A hora que foi feita a leitura e detecção do objeto, tipo *datetime* que atualiza automaticamente quando o dado é recebido.
- **Status:** Determina se o objeto foi ou não detectado e é um *varchar* de duas posições que recebe “tr” ou “fa”.

Figura 49- Estrutura do banco de dados do projeto

#	Nome	Tipo	Agrupamento (Collation)	Atributos	Nulo	Predefinido	Comentários	Extra	Ações
1	id	int(11)			Não	Nenhum		AUTO_INCREMENT	Muda Elimina Mais
2	codigo	varchar(25)	utf8mb4_general_ci		Não	Nenhum			Muda Elimina Mais
3	hora	datetime			Não	current_timestamp()			Muda Elimina Mais
4	status	varchar(2)	utf8mb4_general_ci		Não	Nenhum			Muda Elimina Mais

Fonte: Próprio Autor (2020).

3.10 Modbus/TCP

Para comunicação *Modbus/TCP*, foi utilizado o *software* XG5000 que é responsável pela interface e programação do *CLP* da LSIS. Para acessar o *CLP* deve ligar o módulo XBM-DR16S (Figura 50) na tomada e conectar o cabo de rede no computador.

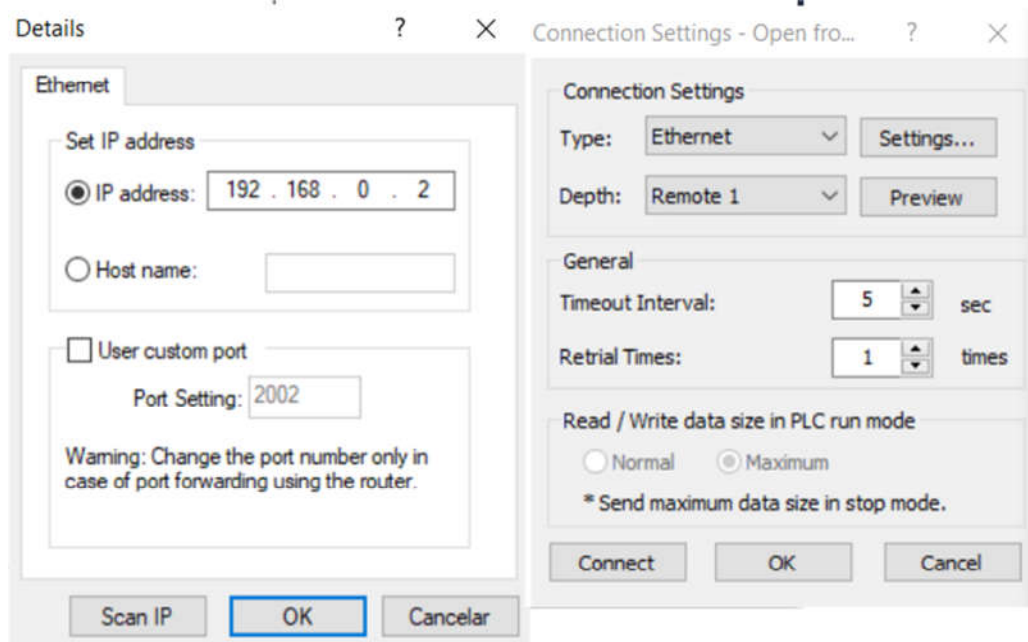
Figura 50- CLP LSIS



Fonte: Próprio autor (2020).

Ao conectar o CLP é necessário escolher a opção *Project->Open From CLP* (Figura 51) e irá aparecer a opção de configuração de conexão, onde na opção *Type* deve-se escolher o modo *Ethernet*, e inserir o IP configurado dentro do CLP.

Figura 51- Tela de configuração do CLP

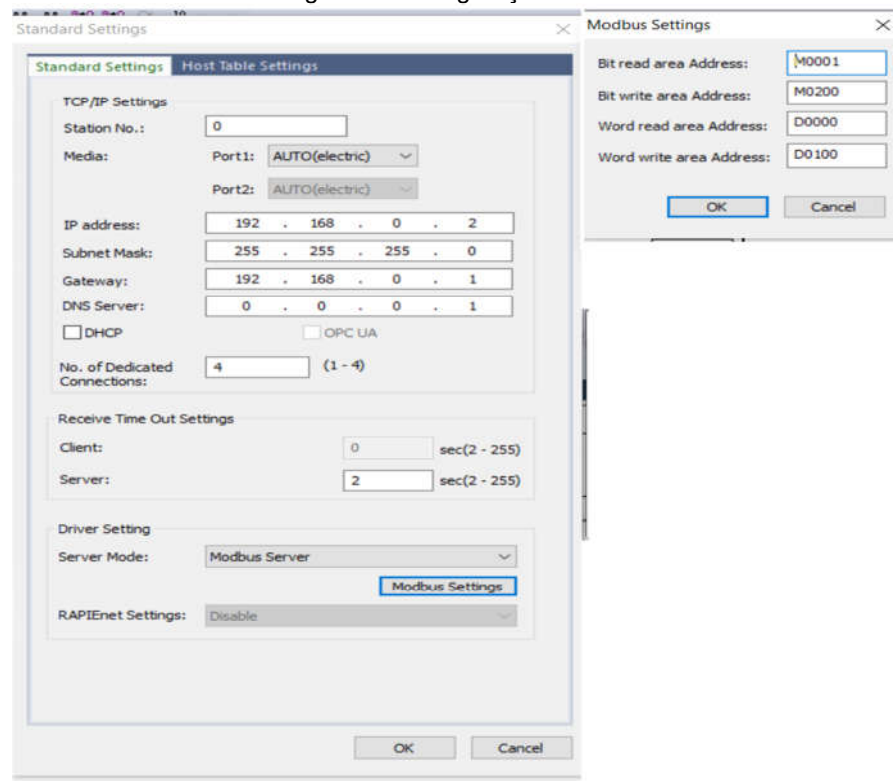


Fonte: Próprio autor (2020).

Ao conectar, é necessário habilitar a opção *Modbus* do módulo e para isso é necessário o usuário selecionar a opção XBL-EMTA, que é o módulo de conexão TCP. Ao selecionar, aparecerá as opções de IP, *gateway* e *subnet/mask*. Em *driver*

settings na opção *server mode* selecionar a opção *Modbus Server* e em *Modbus Settings* selecionar os endereços de entrada e saída de dados dos registros.

Figura 52-Configuração Modbus



Fonte: Próprio autor(2020).

O sistema foi programado em Ladder e no programa desenvolvido foi utilizado dois tipos de memória: a D que é aplicada à registro do *Modbus/TCP* e a memória tipo P que são as portas físicas do sistema P0 até P7 para entradas e P20 a P27 para saída. Para este projeto foi necessário o uso de:

- Um terminal de motor para avanço do atuador (P20).
- Um terminal de motor para recuo do atuador (P21).
- Uma terminal de saída do motor da linha (P22).
- Uma sensor fotoelétrico para detecção da caixa (P4).
- Um botão com trava para ligar a linha (P0).
- Um botão com trava para ativar e desativar o modo de inspeção (P1).
- Um botão com trava para acionar o modo de emergência (P4).

O CLP recebe estes comandos externos e o programa em *Ladder* escreverá as informações recebidas na memória D do *Modbus/TCP* para uso, interpretação e

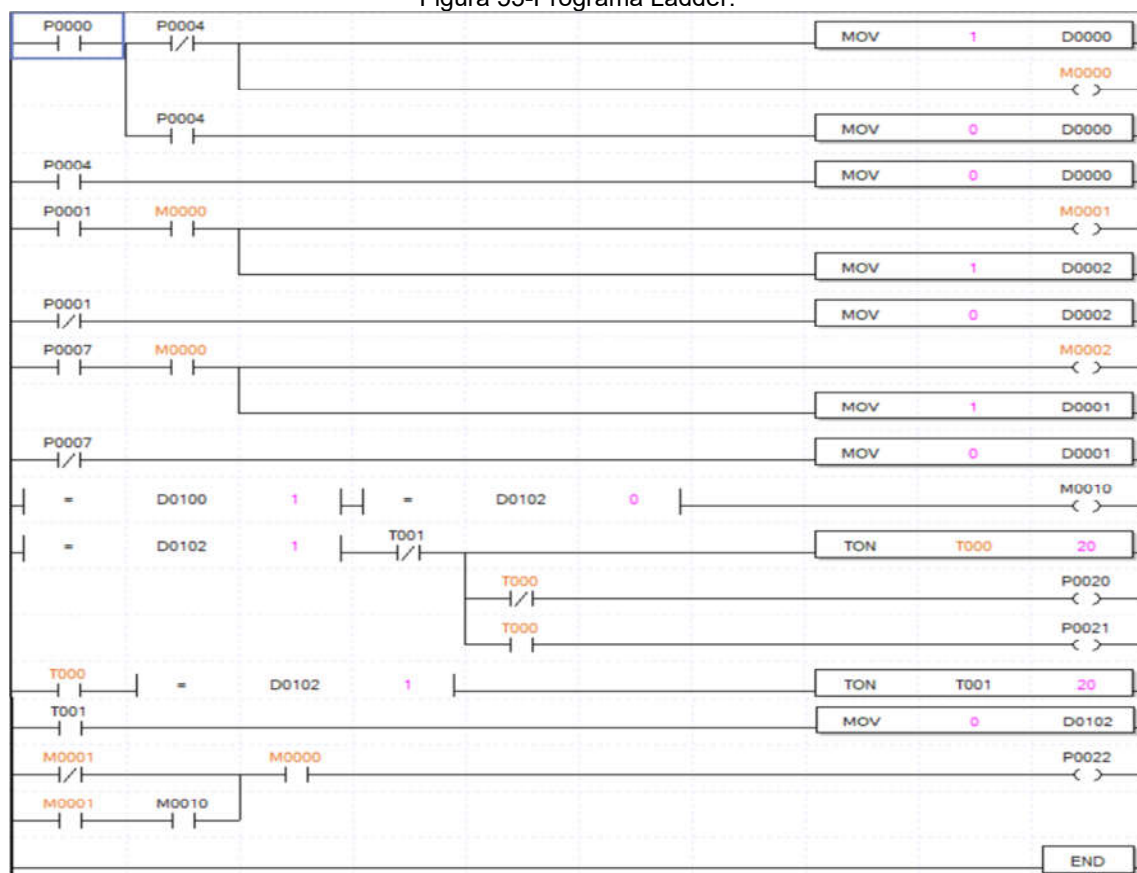
execução de comandos pelo sistema. Para entrada são utilizados os endereços de D0 até D99 e para saída a partir de D100 até D1000. A memória do *Modbus/TCP* utiliza os endereços:

- D0: Tem o objetivo de sinalizar ao programa caso a linha esteja ligada(1) ou desligada(0).
- D1: Verifica se o modo de inspeção está ativado ou desativado.
- D2: Verifica o estado atual do sensor
- D100: Para ativar e desativar o motor da linha
- D102: Para acionar o atuador e retirar a caixa fora de conformidade da linha.

O programa desenvolvido em Ladder (

Figura 53) tem dois modos de inspeção ou sem inspeção. O sem inspeção desativa o sistema de visão e o teste não acontece. Caso o endereço D0 e D1 estejam habilitados o sistema esperará a chegada da caixa que será sinalizada pelo endereço D2; caso o D2 detecte uma caixa o motor recebe um comando de desativação no D100, espera a avaliação e caso não esteja conforme é ativado o avanço do atuador e o recuo para tirar a caixa da linha, caso esteja nos conformes D102 não é ativado. Ao final D100 é ativado e o ciclo se inicia novamente.

Figura 53-Programa Ladder.



Fonte: Próprio autor(2020).

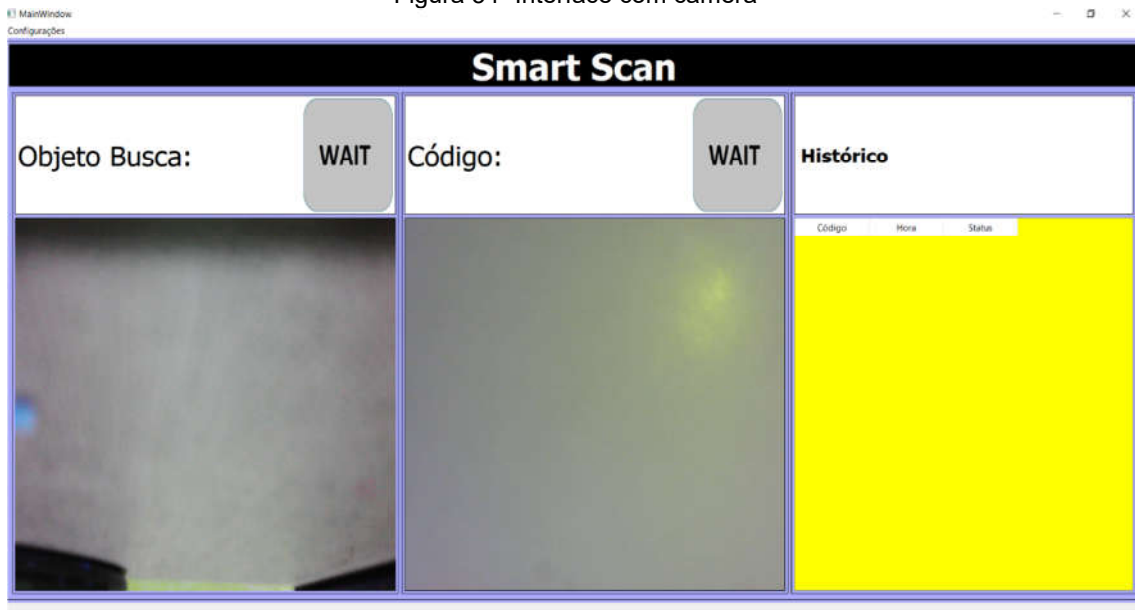
4 TESTES E RESULTADOS OBTIDOS

Ao finalizar as programações e testes de inteligência e interface, todos os recursos foram inseridos em um só algoritmo. Como estratégia de programação, o

código foi dividido em três classes maiores: a primeira compreende à parte gráfica do sistema, a segunda envolve a configuração das câmeras, leitura de código e inteligência artificial, e a última é a parte da comunicação com CLP através de *Modbus/TCP*.

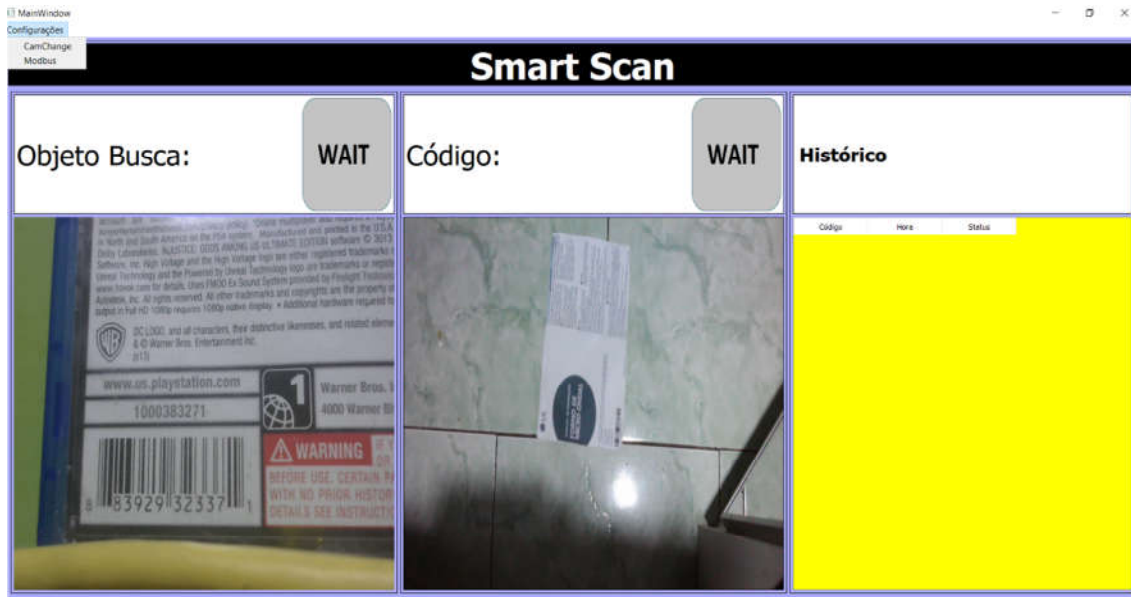
Para testes foi usado o algoritmo *TCC.py* (Figura 54), que irá carregar todas as bibliotecas instaladas e mostrar parte gráfica com o sistema já carregado e caso tudo esteja dentro dos conformes já carregará as imagens das duas câmeras em tempo real.

Figura 54- Interface com câmera



O sistema tem uma aba de configuração (Figura 55), onde uma das opções é a *CamChange* que caso a câmera esteja sendo usada de forma incorreta ao selecionar, ele irá fazer a correção e deixará nas caixas de imagens apropriadas.

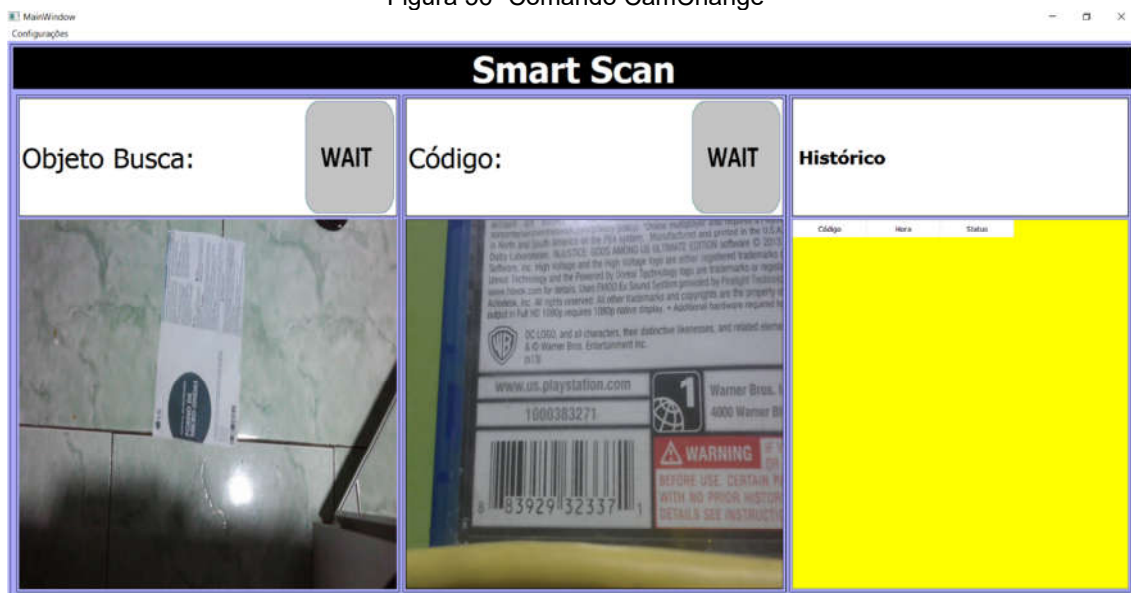
Figura 55- Aba de configuração



Fonte: Próprio autor (2020).

Após a seleção, o sistema irá fazer a correção (Figura 56) e deixará nas caixas de imagens apropriadas, já que cada caixa de imagem tem funções diferentes que faria o sistema não funcionar corretamente caso estivesse invertido.

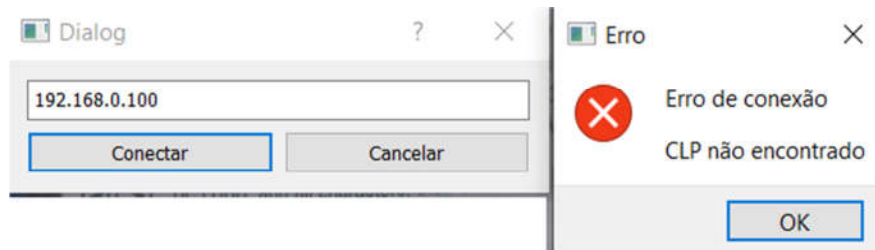
Figura 56- Comando CamChange



Fonte: Próprio autor (2020).

Outra opção selecionada foi a ModBus (Figura 57) ao selecionar uma nova janela abre em que o usuário deve inserir o IP do CLP. Caso não seja detectado, uma janela de erro é apresentada, solicitando uma nova tentativa.

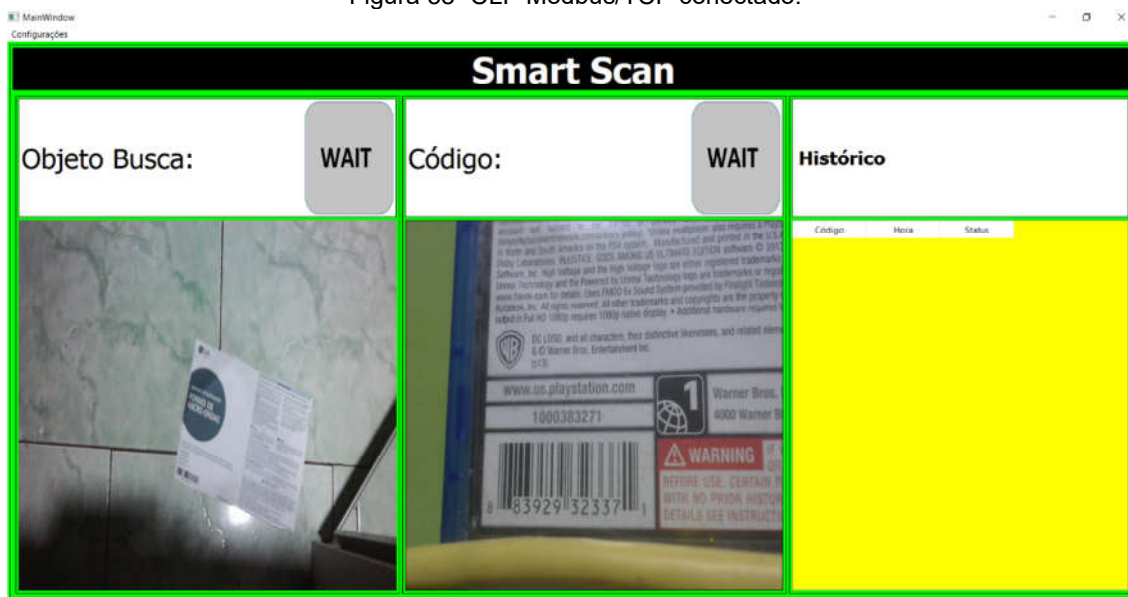
Figura 57- Tela de conexão e erro do Modbus/TCP.



Fonte: Próprio autor (2020).

Caso o CLP for identificado (Figura 58) o fundo do programa ficará verde, sinalizando que os dados vindos do CLP por *Modbus* estão sendo recebidos. No caso do programa o CLP foi sinalizado com o IP: 192.168.0.2.

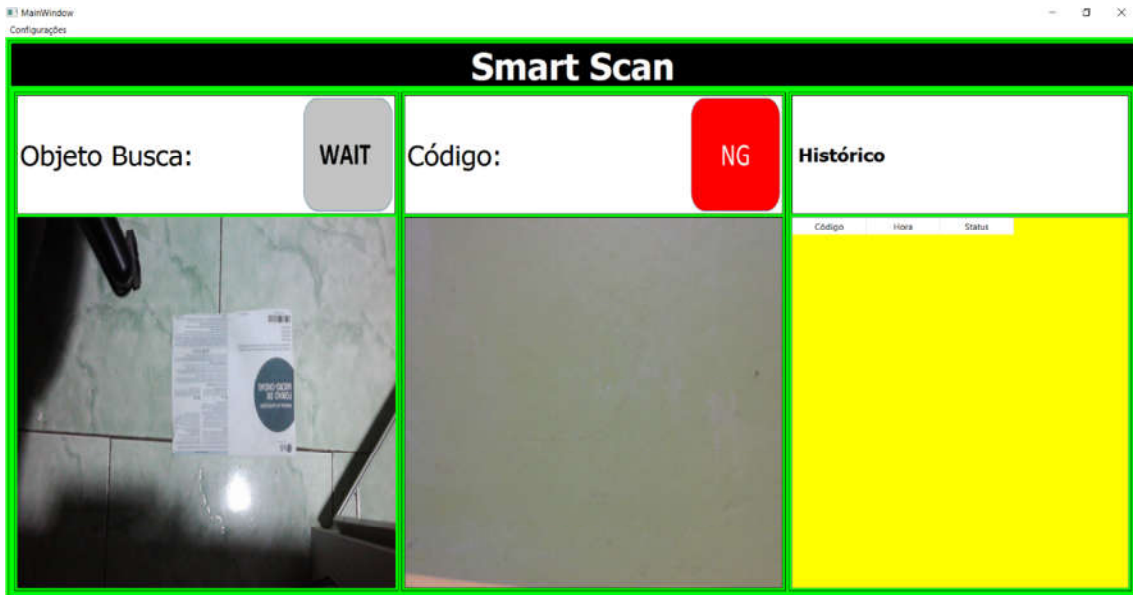
Figura 58- CLP Modbus/TCP conectado.



Fonte: Próprio autor (2020).

Caso o D0 e D1 estejam habilitados o programa esperará pelo sinal vindo do sensor na entrada D2. Caso detecte algum objeto, o programa primeiro irá procurar o código e enquanto isso a inteligência artificial ficará em espera. Se o código não for detectado, aparecerá um estado “NG” (Figura 59). Essa condição não registra no banco de dados e nem na tabela de histórico.

Figura 59- Código não detectado.

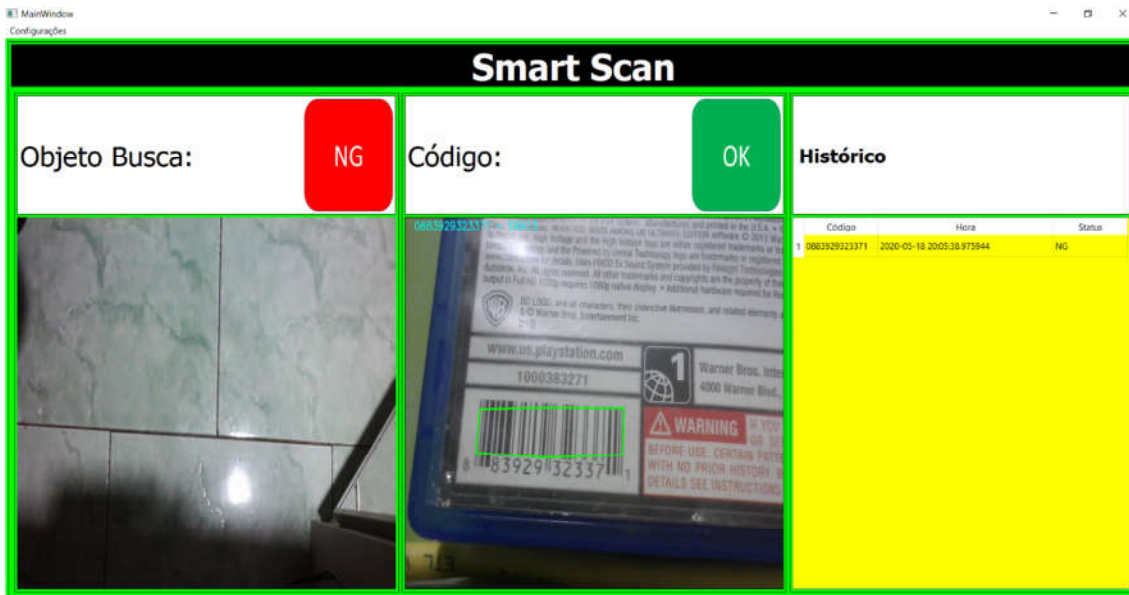


Fonte: Próprio autor (2020).

Com o código não detectado a inteligência artificial não é invocada. Para melhorar o rastreamento dos objetos a melhor opção é amarrar a detecção a algum código. Essa é a primeira condição para retirar o objeto da linha e ativar o atuador para remoção do objeto.

Caso o código seja detectado, o mesmo já é registrado no banco de dados e na área de histórico sinalizando a área de código como “OK” e a inteligência artificial será invocada. Caso o objeto seja detectado no campo de objeto será sinalizado “OK” e se não for detectado será “NG”. Com a inteligência sendo utilizada, caso seja detectada no tempo abaixo de 3 s deixa “OK” de forma permanente e caso passe 5 s sem detectar o atuador de retirada de objeto é acionado (Figura 60).

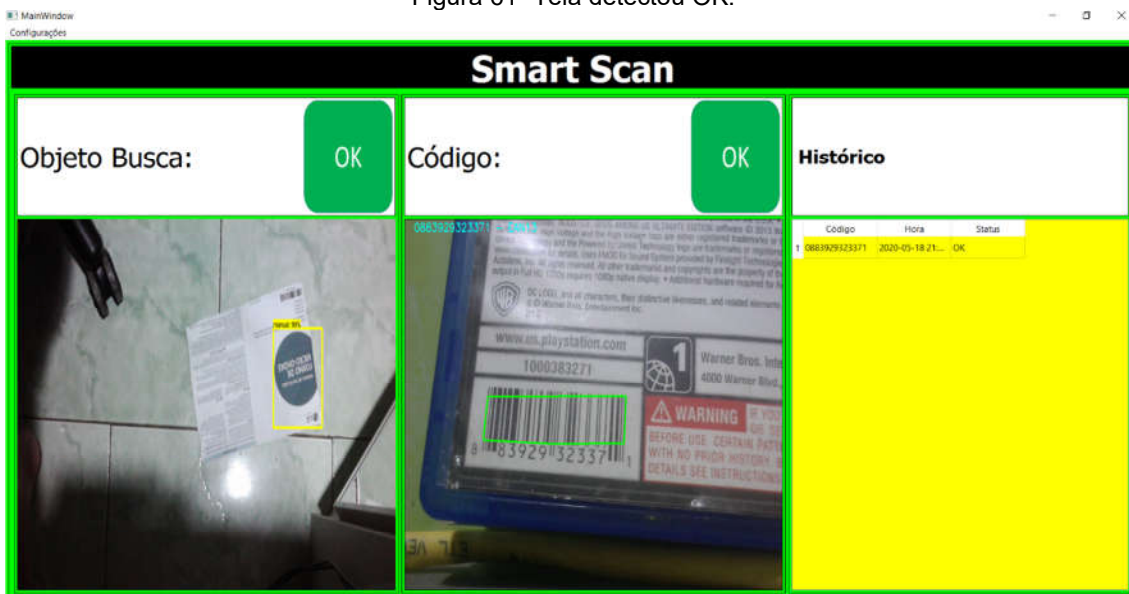
Figura 60- Objeto não detectado



Fonte: Próprio autor (2020).

Devido ao movimento da linha, há a possibilidade do objeto estar embaçado na imagem o que dificulta a detecção do sistema, mas no momento que ele parar a imagem e o objeto ficarão mais nítidos. Por isso, o tempo de 3 segundos esperando o “OK” (Figura 61), mas em testes realizados a detecção é feita em um momento bem menor que o programado.

Figura 61- Tela detectou OK.



Fonte: Próprio autor (2020).

Após a detecção, todos os dados são enviados para o histórico (Figura 62) que registrará tudo e poderá ser consultado por qualquer operador ou responsável para fins de pesquisa e supervisão.

Figura 62- Histórico do sistema

id	codigo	hora	status
6	0883929323371	2020-05-18 20:09:53	Tr
7	0711719506119	2020-05-17 23:30:07	Tr

Fonte: Próprio autor (2020).

5 CONCLUSÃO

Neste trabalho, verificamos que o projeto de criação do sistema de detecção de manuais e seu protótipo foram resultados de um processo extenso de pesquisa e desenvolvimento, exigindo grande dedicação para sua elaboração. O sistema foi desenvolvido para manter uma interface para ser interpretada com facilidade por qualquer operador na linha de produção e gerar um histórico com rastreamento através da leitura do código de barras de todas as caixas que passam pelo posto. Para a comunicação com a linha foi utilizada Modbus TCP, necessitando apenas alterar a programação do CLP. A utilização de um sistema que detecte um manual automaticamente torna as trocas de informações mais fáceis e ágeis dentro do ambiente fabril para o qual o protótipo foi desenvolvido, sendo a contribuição deste trabalho, que utilizou visão computacional e inteligência artificial. De modo a conseguir compreender a fundo os métodos empregados neste sistema, foi necessário procurar em literaturas que são bases da Engenharia de Controle e Automação, o que resultou em fundamentos ao decorrer de todo o estudo.

Este projeto foi instalado com um tempo de inspeção de 3 segundos, onde foi executado em uma empresa e já detectou faltas de manuais por diversos motivos. O

projeto foi apresentado para o presidente na reunião dedicada à implementação de automações de toda a planta e recebeu destaque devido ao baixo custo comparado aos outros projetos.

Como trabalhos futuros, pretendemos desenvolver outros sistemas para detecção de itens cruciais que refletem diretamente no campo já usando este software como base apenas trocando o modelo a ser detectado. Cumprindo as metas da fábrica em relação à qualidade e automação que são exigidas a cada ano.

6 REFERÊNCIAS

ANDRADE, Fabrício. **Tudo sobre o protocolo MODBUS**. Brasil, 6 fev. 2019. Disponível em: <<https://automacaoecartoons.com/2018/11/23/protocolo-modbus/>>. Acesso em janeiro de. 2020.

BIRGONHA, Carolina. **Panorama Setorial da Internet: Inteligência Artificial em perspectiva**. Inteligência Artificial e ética, Brasil, p. 1-15, 1 out. 2018. Disponível em < https://nic.br/media/docs/publicacoes/1/Panorama_outubro_2018_online.pdf >. Acessado em fevereiro de 2020

CARDOSO, Danilo. **ESTUDO DA APLICABILIDADE DE FERRAMENTAS DA INDÚSTRIA 4.0 EM UMA PLANTA DE GERAÇÃO DE ENERGIA A PARTIR DA REFORMA DO BIOGÁS**. UFF, Brasil, p. 15-27, 1 fev. 2018. Disponível em: <<https://app.uff.br/riuff/bitstream/1/8080/1/TCC%20-%20Danilo-lam-Maria%20Julia.pdf>>. Acessado em janeiro de 2020.

CARVALHO, Eduardo dos Santos de Sá; DUARTE FILHO, Nemésio Freitas. **Proposta de um sistema de aprendizagem móvel com foco nas características e aplicações práticas da indústria 4.0**. RISTI, Porto , n. 27, p. 36-51, jun. 2018 . Disponível em <http://www.scielo.mec.pt/scielo.php?script=sci_arttext&pid=S1646-98952018000200004&lng=pt&nrm=iso>. acessos em 14 jun. 2020. <http://dx.doi.org/10.17013/risti.27.36-51>.

DOS REIS, Fábio. **Instalação do PyCharm, IDE para programação em Python.** [S. l.], 9 set. 2016. Disponível em: <http://www.bosontreinamentos.com.br/programacao-em-python/instalacao-do-pycharm-ide-para-programacao-em-python/>. Acesso em: 12 jan. 2020.

FAUSTINO, Bruno. **Seis princípios básicos da Indústria 4.0 para os CIOs.** 2. ed. Brasil, 2 maio 2016. Disponível em: <<https://cio.com.br/seis-principios-basicos-da-industria-4-0-para-os-cios/>>. Acessado em fevereiro de. 2020.

FELICIANO, Flavio. **VISÃO COMPUTACIONAL APLICADA À METROLOGIA DIMENSIONAL AUTOMATIZADA: CONSIDERAÇÕES SOBRE SUA EXATIDÃO.** Engevista, UFF, v. 7, n. 2, p. 38-50, 13 jul. 2005. Disponível em <<https://periodicos.uff.br/engevista/article/view/8789> >. Acessado em dezembro de 2020

FERST, Matheus Kowalczyk. **IMPLEMENTAÇÃO DE COMUNICAÇÃO SEGURA COM MODBUS E TLS.** 2018. TCC (Graduação) - UTFPR, [S. l.], 2018. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/11017/1/PB_COENC_2018_1_07.pdf>. Acessado em janeiro de. 2020.

FILHO, Wil. **Conhecendo o coração e as artérias do PyQt5.** [S. l.], 24 nov. 2014. Disponível em: <https://medium.com/@wilfilho/conhecendo-o-coracao-e-as-artérias-do-pyqt5-22ba4187531>. Acesso em: 22 jan. 2020.

FREITAS, Carlos. **Protocolo Modbus: Fundamentos e Aplicações.** Brasil, 7 abr. 2014. Disponível em: <<https://app.uff.br/riuff/bitstream/1/8080/1/TCC%20-%20Danilo-lam-Maria%20Julia.pdf>>. Acessado em janeiro de 2020.

GEITGEY, Adam. **Face Recognition.** Brasil, 26 jan. 2012. Disponível em: <<https://transformacaodigital.com/o-que-e-machine-learning-e-como-funciona/>>. Acessado em: dezembro de. 2019.

GOMES, Dennis. **Inteligência Artificial: Conceitos e Aplicações.** Olhar Científico, Brasil, v. 1, n. 2, p. 1-13, 1 dez. 2010. Disponível em <academia.edu/28005380/Inteligência_Artificial_Conceitos_e_Aplicações>. Acessado em fevereiro de 2020.

INSTRUMENTS, National. **O protocolo Modbus em detalhes.** [S. l.], 17 set. 2019. Disponível em: <<https://www.ni.com/pt-br/innovations/white-papers/14/the-modbus-protocol-in-depth.html>>. Acessado em dezembro de. 2020.

JÚNIOR, Geraldo Tessarini. **IMPACTOS DA INDÚSTRIA 4.0 NA ORGANIZAÇÃO DO TRABALHO: UMA REVISÃO SISTEMÁTICA DA LITERATURA.** ABEPRO, [s. l.], v. 18, p. 743-769, 2018. Disponível em <<https://producaoonline.org.br/rpo/article/viewFile/2967/1678> >. Acessado em janeiro de 2020

MAGNUS, Tiago. **O que é machine learning e como funciona?.** Brasil, 29 jan. 2018. Disponível em: <<https://transformacaodigital.com/o-que-e-machine-learning-e-como-funciona/>>. Acessado em janeiro de. 2020.

MONTANARI, Raphael. **Detecção e classificação de objetos em imagens para rastreamento de veículos**. 2015. Tese (Mestrado em Ciência de computação e Matemática Computacional) - USP, USP-São Carlos, 2015. p. 58-62. Disponível em < <https://www.teses.usp.br/teses/disponiveis/55/55134/tde-08012016-113715/pt-br.php> >. Acessado em janeiro de 2020

SEVERANCE, Charles. **Python para Todos**. 1. ed. [S. l.]: Shroff Publishers, 2017. 248 p. v. 1.

SILVA, Igor. **LINGUAGEM DE PROGRAMAÇÃO PYTHON**. **Revista Tecnologias em Projeção**, [s. l.], v. 10, n. 1, p. 55-56, 2019. Disponível em < <http://revista.faculdadeprojecao.edu.br/index.php/Projecao4/article/download/1359/1064> >. Acessado em novembro de 2019.

TANNER, Gilbert. **Creating your own object detector**. [S. l.], 6 fev. 2019. Disponível em: <<https://towardsdatascience.com/creating-your-own-object-detector-ad69dda69c85>>. Acessado em novembro de 2019.

VAZ, Welton. **Python**: Como surgiu a linguagem e o seu cenário atual. [S. l.], 1 fev. 2018. Disponível em: <https://www.eusoudev.com.br/python-como-surgiu/>. Acesso em: 8 jan. 2020.

VICTORIANO, Erick Rodrigues. **RECONHECIMENTO DE IMAGEM POR MEIO DE DA VISÃO DE MÁQUINA**. AEDB, [s. l.], 21 abr. 2014. Disponível em < <https://www.aedb.br/wp-content/uploads/2015/04/214130.pdf> > Acessado em dezembro de 2020

YEGULALP, Serdar. **What is TensorFlow? The machine learning library explained**. [S. l.], 18 jun. 2019. Disponível em: <https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html>. Acessado em dezembro de 2019.