CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA DA ETEC DA ZONA LESTE

Mtec Desenvolvimento de Sistemas AMS

Murillo Castro De Jesus

Rafael Ferreira Lopes

Vinicius Rafael Rodrigues

Vitor Daisuke Iwamoto

EYE TRUCK: Sistema de Prevenção de Colisão com Viadutos e Pontes

São Paulo

2025

Murillo Castro De Jesus

Rafael Ferreira Lopes

Vinicius Rafael Rodrigues

Vitor Daisuke Iwamoto

EYE TRUCK: Sistema de Prevenção de Colisão com Viadutos e Pontes

Trabalho de Conclusão de Curso apresentado ao Curso Ensino Médio com Habilitação Profissional de Técnico em Desenvolvimento de Sistemas AMS da Etec Zona Leste, orientado pelo Professor Jeferson Roberto de Lima, como requisito parcial para obtenção do título de técnico em Desenvolvimento de Sistemas.

São Paulo

2025

Resumo

O projeto propõe desenvolver um dispositivo IoT com a finalidade de evitar colisões de veículos de transporte de cargas em viadutos. O sistema é formado por um dispositivo com hardware e uma aplicação web. O software será utilizado para o usuário informar a altura do caminhão. O hardware compara a altura do caminhão com a altura máxima permitida, indicada na placa do viaduto captado por uma câmera. Com essa comparação será possível emitir sinais alertando o condutor do caminhão sobre a altura do veículo em relação à altura do viaduto, podendo evitar uma possível colisão por falta de atenção do motorista.

Palavras-chave: Internet das Coisas (IoT); Altura máxima permitida; Reconhecimento de placas; Prevenção de colisões; Veículos de carga.

Abstract

The project proposes to develop an IoT device to prevent collisions between cargo vehicles on overpasses. The system consists of a hardware device and a web application. The software will be used to inform the user of the height of the truck. The hardware compares the height of the truck with the maximum permitted height, indicated on the overpass sign captured by a camera. With this comparison, it will be possible to emit signals alerting the truck driver about the height of the vehicle in relation to the height of the overpass, thus avoiding a possible collision due to the driver's lack of attention.

Keywords: Internet of Things (IoT); Maximum allowed height; Sign recognition; Collision prevention; Cargo vehicles.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

Internet of Things (IoT)

Deep Learning (DL)

Universal Serial Bus (USB)

Gigahertz (GHz)

Integrated Development Environment (IDE)

Kotlin Multiplataforma (KMM)

Limited Inquiry Access Code (LIAC)

Media Access Control (MAC)

Optical Character Recognition (OCR)

Package Installer for Python (PIP)

Rede Neural Artificial (RNA)

Unified Modeling Language (UML)

Open Source Computer Vision Library (OpenCV)

You Only Look Once (YOLO)

SUMÁRIO

1 INTRODUÇÃO

Objeto (temática) – O Eyetruck consiste em é um dispositivo.....

Objetivo geral - O objetivo geral deste dispositivo é prevenir colisões contra estruturas rodoviárias, de forma.....

Problemática - O principal problema é .....

Hipóteses - ausência de padronização no tamanho das estruturas e sinalização. Incialmente, ...

Justificativa - O presente estudo justifica-se pela importância de prevenção de acidentes e custos desnecessários, causados pela não existência de recursos que sejam capazes de alertar ou informar os condutores de veículos pesados acerca das estruturas....

Delimitação – São Paulo, capital, região metropolitana. Dada a grande quantidade veículos de grande porte que circulam e causam acidentes ....

Principais autores - Para falar sobre acidentes de trânsito utilizou-se XXXXXXXX (00000), para falar sobre IOT baseando-nos principalmente em se XXXXXXXX (00000), sobre a tecnologia YOLO.

2 REFERENCIAL TEÓRICO

Neste capítulo serão apresentados os principais conceitos, tecnologias e uma visão geral da problemática, tendo em vista melhor entendimento do contexto em que este trabalho foi desenvolvido.

2.1 Acidentes envolvendo caminhões e viadutos

Segundo notícia publicada pela Folha de S. Paulo (2019), ao menos 15 caminhões por mês ficam bloqueados por ultrapassarem a altura permitida em pontes e viadutos. Entre janeiro de 2017 e outubro de 2018, foram registrados 213 casos desse tipo de ocorrência, que geraram riscos a outros condutores de veículos e danos às infraestruturas de transportes.

2.2 Tecnologias Utilizadas

A seguir, serão explicadas as tecnologias utilizadas no desenvolvimento do projeto Eye Truck, abordando linguagens de programação, componentes eletrônicos e conceitos utilizados.

2.2.1 Internet of Things (IoT)

De acordo com Santos (2019), a Internet das Coisas é uma conexão de dispositivos que se comunicam entre si com a finalidade de realizar tarefas de forma autônoma, sem a necessidade de interação humana.

Carrion e Quaresma (2019) destacam que o cientista da computação e inventor John Romkey criou uma torradeira que podia ser ligada e desligada pela internet, sendo considerada o primeiro de muitos dispositivos IoT.

Diante disso, percebe-se que a Internet das Coisas é muito mais do que apenas conectar um dispositivo à internet. Seu objetivo é tornar as “coisas” inteligentes, sendo capazes de aproveitar a conexão e utilizar os dados coletados do ambiente ou das redes às quais estão conectadas (OLIVEIRA, 2017).

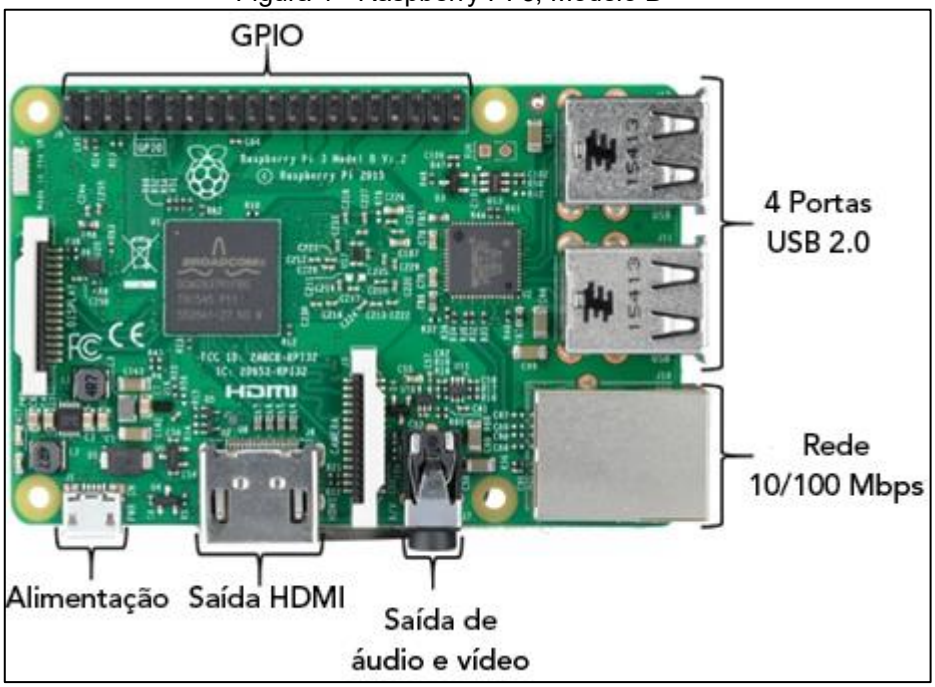
2.2.2 Raspberry PI

Em 2006, Eben Upton, Rob Mullins, Jack Lang e Alan Mycroft criaram um computador portátil e acessível para crianças na Universidade de Cambridge, na Inglaterra. A iniciativa surgiu após a percepção de que os estudantes que se inscreviam para participar do laboratório de Ciências da Computação apresentavam pouco conhecimento prévio sobre linguagens de programação e arquitetura dos computadores quando comparado à dos alunos de 1990. Assim, o projeto teve como objetivo ajudar jovens a adquirirem conhecimentos de *software* e *hardware*. (EBERMAM et al. 2017).

Em 2012, a empresa Raspberry PI, fundada por Eben e seus colegas, criou um microcomputador de baixo custo com o mesmo nome. Esse pequeno dispositivo é um ótimo meio para a introdução aos conceitos de Eletrônica, podendo ser utilizado tanto por novatos na área da tecnologia quanto por especialistas nas áreas de Computação. (OLIVEIRA; NABARRO; ZANETTI, 2018).

Conforme Silva, Dias e Escudero (2022), o Raspberry PI possui diferentes modelos disponíveis. A principal diferença entre eles está na sua capacidade computacional. Alguns exemplos para melhor entendimento seriam a potência e o tipo do processador, quantidade de entradas USB (*Universal Serial Bus*), tecnologias implementadas, entre outros.

Figura 1 - Raspberry PI 3, Modelo B



Fonte: Oliveira, Nabarro, Zanetti, 2018

2.2.3 Bluetooth

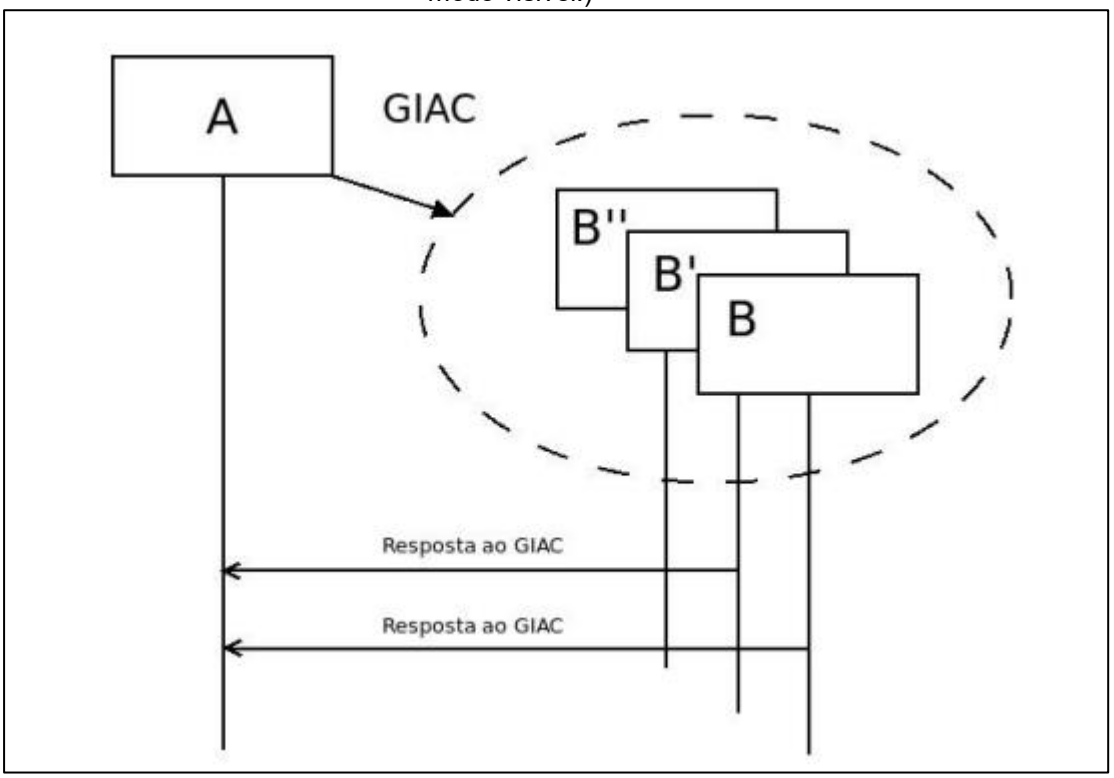
Em conformidade com Siqueira (2006), o Bluetooth é uma ferramenta de comunicação que elimina a necessidade de cabos ou fios, criada para comunicação de curta distância. Ele se destaca pelo seu valor acessível e pelo baixo consumo de energia. Ele facilita a conexão entre dispositivos como celulares, laptops, câmeras digitais, impressoras, entre outros, permitindo a transmissão de informações de forma segura e sem a dependência de um cabo.

A ideia inicial da tecnologia era fazer com que um dispositivo conseguisse se conectar a outro de forma rápida com um alcance de 1 a 100 metros. Uma de suas vantagens é a capacidade de estabelecer conexões automáticas entre dispositivos que entram em proximidade, formando uma rede sem configuração prévia (SIQUEIRA, 2006).

A arquitetura Bluetooth tem como um dos seus componentes o transceptor que é responsável pela comunicação por rádio e a pilha de protocolos, que oferece tipos de funcionalidades e serviços para estabelecer conexões e troca de dados. A operação ocorre na faixa de frequência ISM (industrial, científica e médica), entre 2,4 GHz (Gigahertz) e 2,485 GHz, faixa essa de uso livre em diversas regiões. Para não existir interferências ou problemas com sinal, é utilizado uma técnica de salto de frequência (*frequency hopping*), com até 1600 mudanças de canal por segundo (SIQUEIRA, 2006).

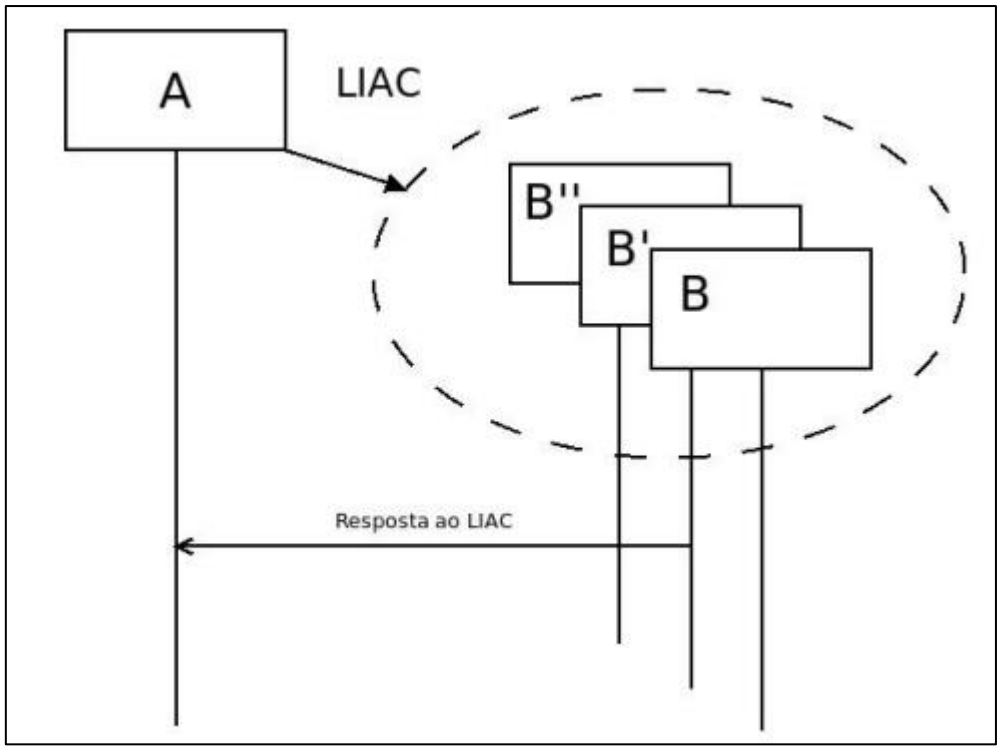
Atoji (2010) complementa ao descrever o processo de descoberta de dispositivos Bluetooth, que pode ser iniciado por qualquer equipamento com o objetivo de localizar outros dispositivos presentes na mesma área de cobertura. Essa descoberta é feita por meio do envio de uma mensagem de difusão (broadcast), utilizando um dos dois códigos de acesso disponíveis: GIAC ou LIAC. O GIAC (*General Inquiry Access Code*) é utilizado para detectar dispositivos que ficam sempre visíveis, enquanto o LIAC (*Limited Inquiry Access Code*) é utilizado para os que têm a sua visibilidade temporária. A diferença entre os dois códigos é importante em locais com grande concentração de dispositivos, por isso é possível separar os dispositivos sem confusões. As respostas obtidas incluem o endereço físico de rede Bluetooth (*MAC address*), permitindo que o dispositivo solicitante inicie o processo de conexão com o outro (ATOJI, 2010).

Figura 2 - Descoberta geral (GIAC), onde B'' está oculto, B' está visível por tempo limitado, e B em modo visível.)



Fonte: Atoji, 2010, p. 6.

Figura 3 - Descoberta limitada (LIAC), onde B está oculto, B' está visível por tempo limitado e B em modo visível.



Fonte: Atoji, 2010, p. 6.

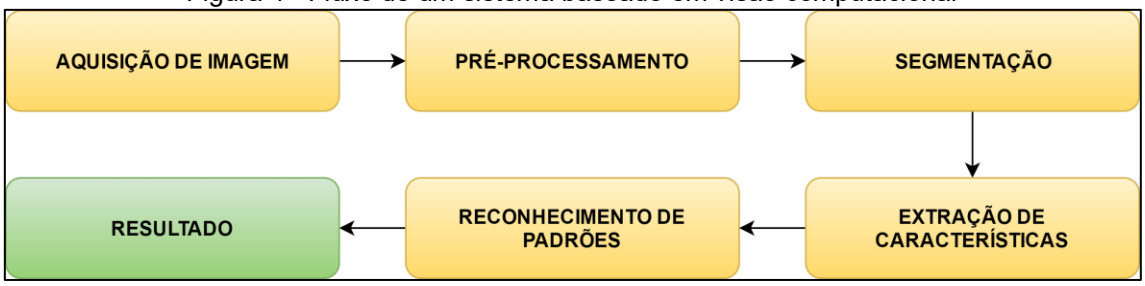
2.2.4 Visão Computacional

Segundo Barelli (2018) Visão Computacional é o estudo de máquinas que possam reconhecer elementos de um ambiente, através de imagens capturadas, sensores e outras tecnologias, essas informações podem ser extraídas e manipuladas pelo desenvolvedor conforme seus objetivos.

A visão computacional é um tema muito recente no mundo tecnológico, com sua primeira aparição em 1955, com Selfridge. Na década de 70 começaram as pesquisas de visão computacional com a inteligência artificial, os cientistas acreditavam que em breve teriam o sentido completo da visão das máquinas, atrasando-se em decorrência da complexidade que o tema possuía na época para a ciência, surgindo pela falta de informações biológicas de como nosso cérebro consegue interpretar um ambiente ou objeto (MILANO; HONORATO, 2014).

De acordo com Barelli (2018) os sistemas de visão computacional, por mais que pareçam complexos, seguem um fluxo em comum, como mostrado no exemplo a seguir.

Figura 4 - Fluxo de um sistema baseado em visão computacional



Fonte: Barelli, 2018, p. 18.

Visão Computacional não é somente limitado ao controle de qualidade, esta tecnologia pode ser usada em diversas áreas do mundo, desde agricultura até sistemas de reconhecimento autônomo, um desses exemplos é o "Alvinn" um robô produzido por Dean Pormeleau, um carro que podia adquirir conhecimento através da experiencia, ou seja, ele reconhecia as ruas e "memorizava" elas. (MILANO; HONORATO, 2014).

2.2.5 OpenCV (Open Source Computer Vision Library)

OpenCV é uma biblioteca multiplataforma feita pela Intel que divulgou sua criação por volta dos anos 2000, desenvolvida especialmente para o carregamento de imagens e estruturação de dados em aplicações, considerado um grande aliado da visão computacional o OpenCV, por ser de código aberto (Open Source), ou seja, é disponível para uso do público. (BARELLI, 2018).

Consoante a Marengoni e Stringhini (2010), o OpenCV é uma biblioteca gigante com diversas funções disponíveis de acesso gratuito, é dividido em cinco principais grupos, que são: Manipulação e tratamento de imagens, análise da estrutura de cenas, monitoramento de movimento e rastreamento de objetos, identificação de padrões e ajuste de câmera com reconstrução em 3D. A ideia do projeto foi criar um projeto que facilitasse o acesso dos desenvolvedores a tecnologias de visão computacional.

Agora iremos exibir um exemplo que consiga identificar, capturar e salvar uma imagem de um rosto.

Figura 5 - Trecho do código que realiza a captura da face

```
cam = cv2.VideoCapture(1)
cam.set(3, 640)  # set video width
cam.set(4, 480)  # set video height

face_detector = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
```

Fonte: Sette, Carvalho, Basile, 2021, p. 5.

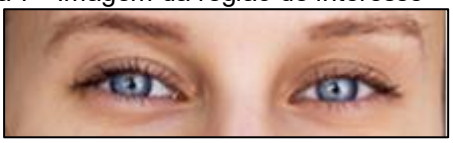
No exemplo do código, dois métodos principais da biblioteca OpenCV são usados para executar o reconhecimento facial. O primeiro, cv2.VideoCapture(), ativa a câmera do dispositivo, permitindo a captura de imagens em tempo real. O segundo, cv2.CascadeClassifier, aplica modelos de detecção baseados em classificadores em cascata já treinados, para identificar os rostos nas imagens capturadas. Aí também estão algumas outras funções, como cv2.cvtColor, para converter as imagens em tons de cinza, e cv2.imwrite, que salva os arquivos gerados, que também são frequentes ao longo do código. (SETTE; CARVALHO; BASILE, 2021)

Figura 6 - Imagem da face alinhada com marcação nos olhos

Fonte: Sette, Carvalho, Basile, 2021, p. 6.

Nesta imagem o OpenCV está destacando o ponto de interesse, ou seja, ele descarta o que não é relevante para o desenvolvedor e para o sistema para facilitar a extração de dados e informações úteis com mais facilidade.

Figura 7 - Imagem da região de interesse



Fonte: Sette, Carvalho, Basile, 2021, p. 6.

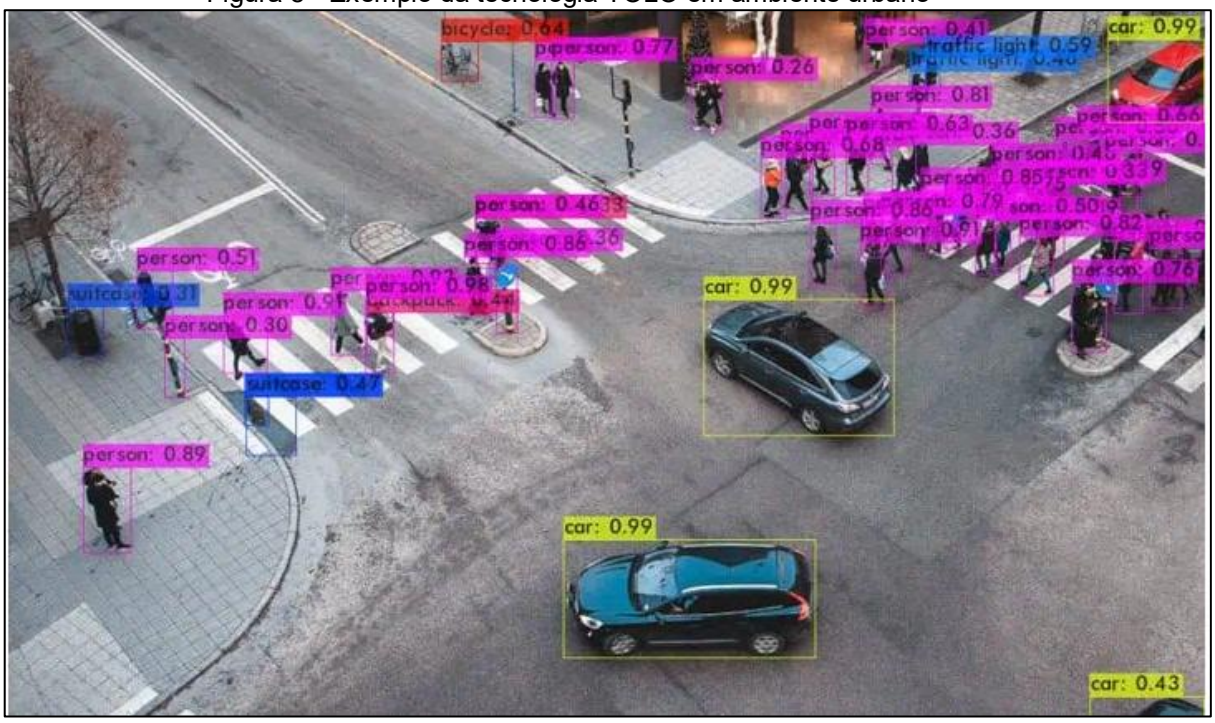
Por fim ele separou a região de interesse do resto da imagem utilizando as bibliotecas do OpenCV para o desenvolvedor poder manipulá-la da forma que quiser, como por exemplo, reconhecimento por íris ou photoshop. (SETTE; CARVALHO; BASILE, 2021).

2.2.6 YOLO (You Only Look Once)

YOLO (You Only Look Once) é conhecido por suas práticas com redes neurais, é um modelo de detecção de objetos criado por Joseph Readmon e Ali Farhadi na universidade de Washington em 2015. Seu diferencial dentre as outras tecnologias de detecção de objetos é sua velocidade para a detecção, porém pelo YOLO olhar somente uma vez para a imagem acontece uma perda de acurácia de acordo com Araújo (2022).

Conforme Silva (2018, apud HUANG, 2024), YOLO cria (grids) que são divisões na imagem, para que dentro de cada grid exista uma caixa delimitadora (bounding box) para armazenar a probabilidade de existir um objeto naquela região.

Figura 8 - Exemplo da tecnologia YOLO em ambiente urbano



Fonte: Ultralytics, 2023

Cada Bounding box tem um ponto de confiança que vai indicar se existe ou não a chance de existir um objeto na região, caso a probabilidade seja muito baixa a bounding box vai desaparecer consoante ao Leocádio et al. (2021).

Concomitante ao Pacheco (2018), o YOLO vem de uma Rede Neural Artificial (RNA), o RNA surgiu por meio de um modelo de matemática do neurônio biológico realizado por MCulloch e Pitts na data de 1943. No RNA é feito diversas camadas que agem de forma analógica a um neurônio realizando um processamento de uma parte da informação total, essas camadas são: camada de entrada, camadas intermediárias ou ocultas, e camada de saída.

A camada de entrada irá inicialmente receber os primeiros dados da rede, logo depois na camada intermediária irá acontecer o maior número de processamento e por último, dentro da camada de saída, acontecerá a conclusão que é exibida de acordo com Kovács (2023).

Segundo Pereira (2017), Deep Learning (DL) também chamado de Aprendizado Profundo envolve RNA, o DL foi criado como exemplo para analisar e assimilar arquiteturas que têm quantidades maiores de dados, com isso ele é capaz

de criar exemplos parciais que são assimilados durante o treinamento, assim ocupando a necessidade do pré-processamento e extração de recursos manualmente designados.

2.2.7 Python

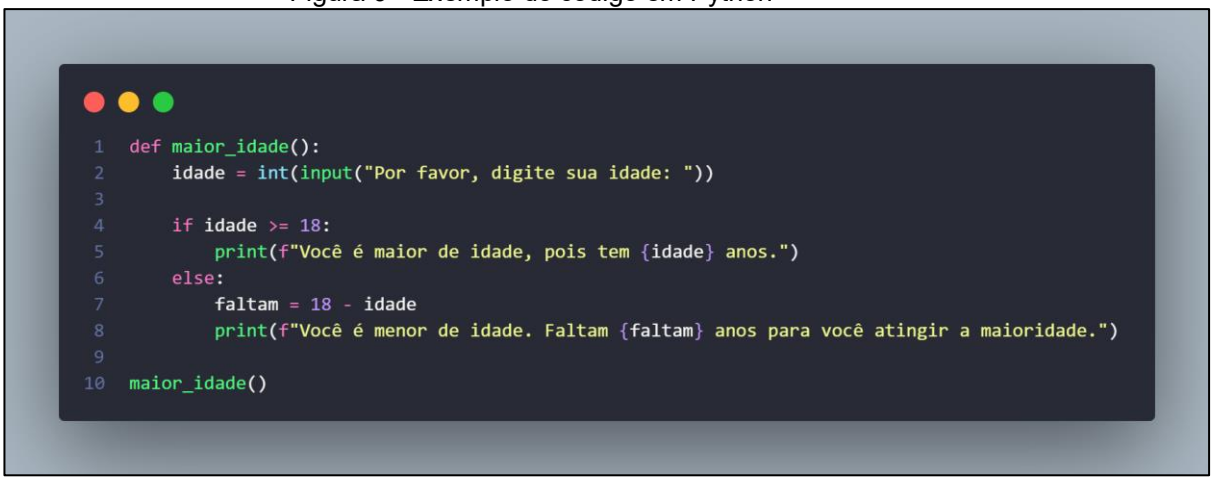
A linguagem de programação Python foi desenvolvida por Guido van Rossum em 1990, no Instituto (CWI) da Holanda, inicialmente prezando automatizar processos de cálculos matemáticos para engenheiros e físicos, além de substituir sua linguagem predecessora, a ABC, de acordo com Borges (2014).

Segundo Matthes (2016), Python tem uma grande performance diante de sua simplicidade, com uma sintaxe objetiva e legível, grande versatilidade em uso geral, fácil manutenção e alta concentração de pacotes para o auxílio de desenvolvedores, sendo capaz de realizar desde pequenos códigos até grandes aplicações.

> Nos últimos 20 anos, ele passou de uma linguagem de computação científica inovadora [...] para uma das mais importantes linguagens para ciência de dados, machine learning e desenvolvimento de softwares em geral no ambiente acadêmico e no setor empresarial. (McKinney, 2023, p.18)

Python consolidou seu espaço na indústria global, sendo aplicada em grandes empresas, como Google, Yahoo, Microsoft, Nokia e Disney para aplicações web, desenvolvimento de softwares para celulares e animações 3D, como destaca Borges (2014).

Figura 9 - Exemplo do código em Python



Fonte: Autoria Própria, 2025

O código acima é um exemplo de uma função básica em Python que realiza uma verificação para saber se a idade digitada pelo usuário indica se ele é maior de idade ou não. A seguir, uma breve explicação do código:

Linha 1: Cria uma função denominada maior_idade.

Linha 2: Utiliza a função input() para o usuário digitar sua idade e retornar como string. Através da função int(), ele transforma esse valor em um número inteiro, que será armazenado dentro da variável idade.

Linha 4: Usa o comando de decisão if para verificar se o valor armazenado na variável idade é maior ou igual a 18 por meio do operador de comparação “maior ou igual a” (>=) .

Linha 5: Aplica a função print() para aparecer a mensagem no console. O F fora das aspas aponta que é uma f-string, que possibilita adicionar variáveis no texto, como a variável idade adicionada na mensagem.

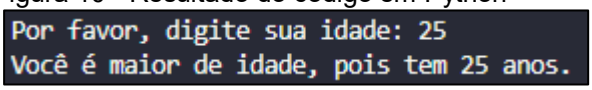
Linha 6: Caso a condição idade >= 18 for falsa, emprega-se outro comando de decisão, o else, que executará outro código.

Linha 7: Calcula quantos anos faltam para o usuário completar 18 anos por meio de uma subtração, 18 – idade, que é armazenada na variável faltam.

Linha 8: Exibe uma mensagem no console informando que o usuário ainda é menor de idade e mostrando quantos anos faltam para atingir a maior idade, através da variável faltam.

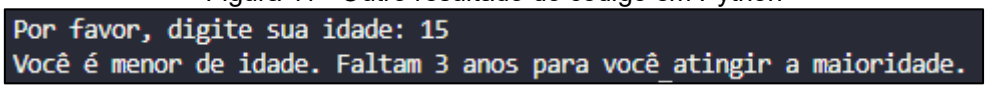
Linha 10: Chama a função maior_idade para que o código seja executado.

Figura 10 - Resultado do código em Python



Fonte: Autoria Própria, 2025

Figura 11 - Outro resultado do código em Python



Fonte: Autoria Própria

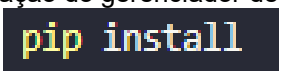
Acima, há uma demonstração de dois exemplos do retorno do código em Python, nos quais foram necessários fornecer a idade do usuário. Foi possível fazer uma verificação por meio de um comando de decisão e do uso de um operador de comparação para indicar se ele era maior de idade ou não.

2.2.7.1 Package Installer for Python (PIP)

O Package Installer for Python (PIP) é o gerenciador de pacotes do Python. Ele gerencia as bibliotecas que você for instalar, atualizar e desinstalar. É uma ferramenta crucial para o gerenciamento de pacotes no Python, segundo Matthes (2016).

Logo abaixo, há uma figura exibindo o comando para a instalação da biblioteca Selenium em Python:

Figura 12 - Instalação do gerenciador de pacotes do python



Fonte: Autoria Própria, 2025

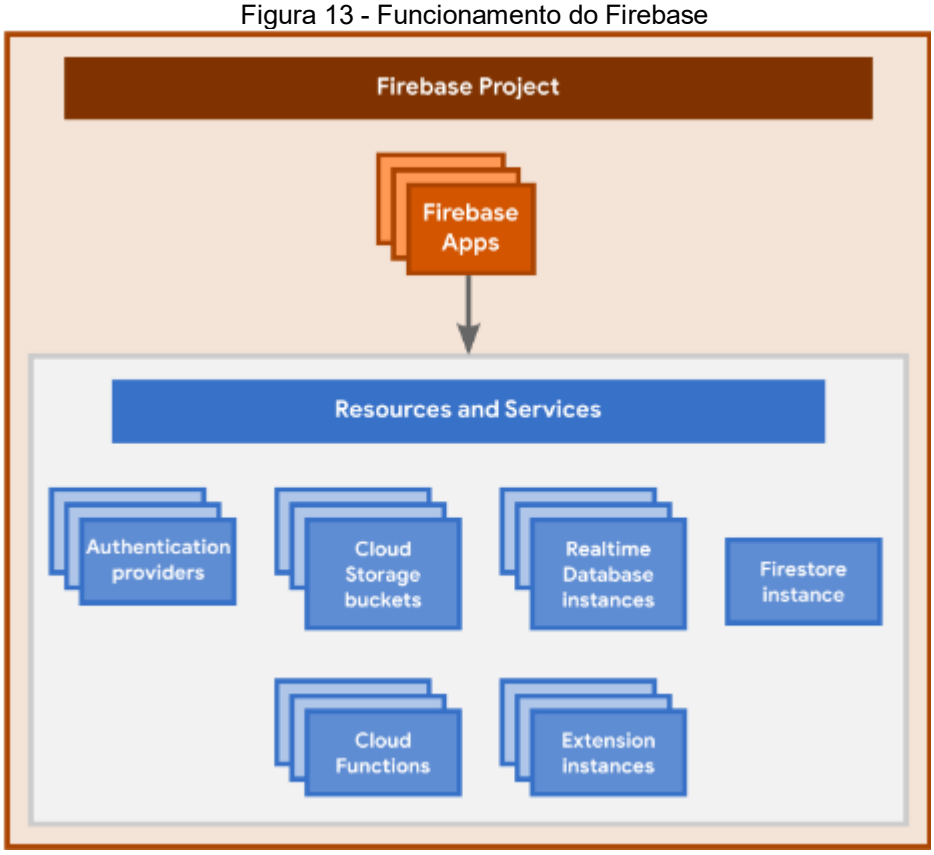
2.2.8 Banco de Dados

Gonçalves (2014) propõe que bancos de dados por sua vez são um conjunto de informações relacionadas que podem ser agrupadas em um local no computador e nos tempos de hoje na nuvem, esses dados são organizados da forma de linhas e colunas e são guardadas em tabelas que são criadas dentro do banco, e podem ser manipuladas com o CRUD (Create, Read, Update e Delete).

2.2.9 Firebase

De acordo com Google (2025), o Firebase é um serviço gratuito disponibilizado pela empresa que permite o usuário criar e desenvolver projetos através de diversos produtos dentro do próprio Firebase, sendo elas Analytics, Cloud Firestore, Performance Monitoring ou Remote Config.

Figura 13 - Funcionamento do Firebase



Fonte: Google, 2025

Dentro de um projeto Firebase é possível criar um ou mais aplicativos e todos eles com os mesmos acessos e recursos, ou seja, os aplicativos registrados no seu projeto são ligados à mesma propriedade de acordo com (GOOGLE, 2025).

Em conformidade com Google (2025), para o usuário ter privilégios na plataforma e em projetos é disponibilizado dois planos pagos, como o plano Spark e Blaze com diferentes números de limitações de criação de projetos.

2.2.9.1 Cloud Firestore

Cloud Firestore é um produto disponibilizado pela empresa Google com Firebase, sendo um banco de dados flexível com foco em dispositivos móveis, plataformas na internet e servidores de acordo com (GOOGLE, 2025).

Conforme a Google (2025), o Cloud Firestore é hospedado na nuvem, assim podendo ser acessado juntamente com os aplicativos da Apple, Android e da Web com um conjunto de ferramentas disponibilizado pela empresa chamado SDKs, trabalhando com o NoSQL do Cloud Firestore que armazena dados em blocos de documentos, assim mapeando valores para ajudar o usuário a se organizar e desenvolver consultas.

Figura 14 - Explicação de um Dado

Fonte: GOOGLE (2025).

De acordo com Google (2025), o Cloud Firestore é rápido e funcional para recuperação de dados específicos sem a necessidade de trabalhos desnecessários e mantendo os dados atualizados.

2.2.10 Tesseract OCR

Em conformidade com Lima (2024), o Tesseract é uma biblioteca responsável pelo reconhecimento óptico de caracteres, amplamente empregada a extração de textos presentes em vídeos ou imagens.

Essa ferramenta demonstrou notável evolução em seus resultados durante os anos e é considerado um dos principais motores de OCR mais utilizados na atualidade, de acordo com Zorn, Andriollo e Borba (2024 apud SMITH, 2007).

Segundo Neto (2023 apud CABRAL; MACHADO, 2014), o Tesseract foi originalmente desenvolvido pela Hewlett-Packard entre 1984 e 1995, mas somente no ano de 2005 foi disponibilizado em código aberto.

2.2.11 Pytesseract

Pytesseract conforme Pereira *et al.* (2023), é uma biblioteca em Python que fornece uma base de dados já treinada para reconhecimento de caracteres, utilizando o módulo de código aberto da tecnologia OCR Tesseract. Por meio dessa tecnologia, é possível realizar o reconhecimento de caracteres a partir de imagens em Python.

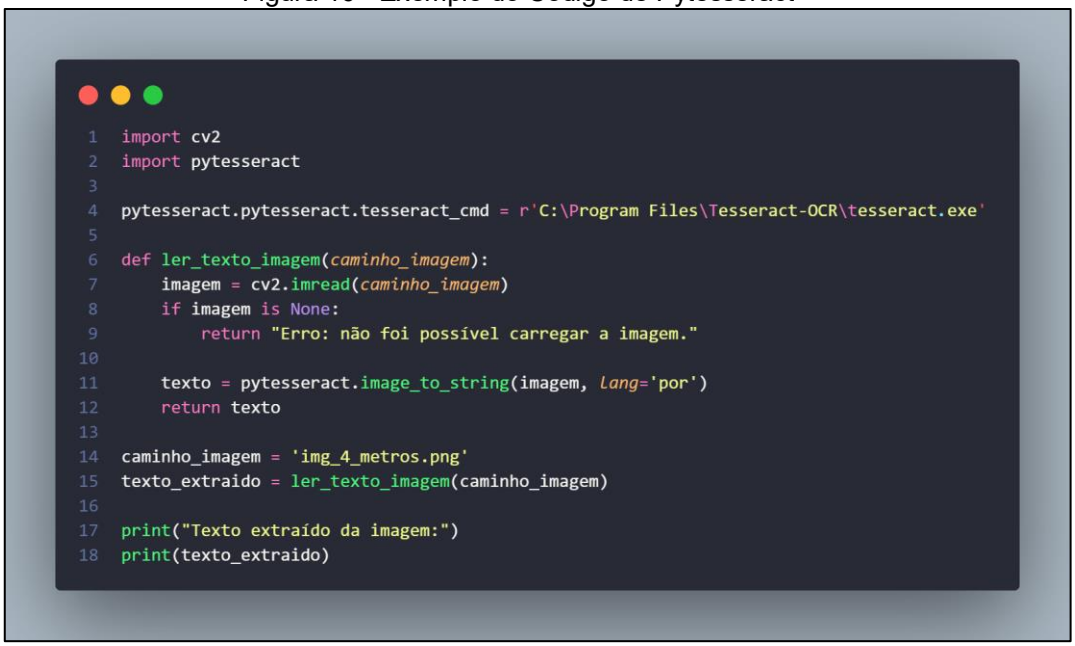
Figura 15 - Instalação do Pytesseract

```
pip install pytesseract
```

Fonte: Autoria Própria, 2025

A instrução acima indica como realizar a instalação da biblioteca Pytesseract e suas dependências para a linguagem de programação Python.

Figura 16 - Exemplo do Código de Pytesseract



```python
import cv2
import pytesseract

pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

def ler_texto_imagem(caminho_imagem):
    imagem = cv2.imread(caminho_imagem)
    if imagem is None:
        return "Erro: não foi possível carregar a imagem."

    texto = pytesseract.image_to_string(imagem, lang='por')
    return texto

caminho_imagem = 'img_4_metros.png'
texto_extraido = ler_texto_imagem(caminho_imagem)

print("Texto extraído da imagem:")
print(texto_extraido)
```

Fonte: Autoria Própria, 2025

O código apresentado posteriormente é uma pequena demonstração sobre o uso de Pytesseract junto com OpenCV para o reconhecimento de caracteres de uma imagem pré-definida pelo desenvolvedor. Logo abaixo uma pequena explicação sobre o código:

Linha 1: Importação do módulo do OpenCV, usado para processar vídeos e imagens.

Linha 2: Código pertencente a biblioteca que conecta o Tesseract OCR ao Python, usado para reconhecer os caracteres presentes em imagens.

Linha 4: Indica a biblioteca Pytesseract onde encontrar o executável do Tesseract instalado no sistema.

Linha 6: Cria uma função chamada “ler_texto_imagem” que recebe o “caminho_imagem” como parâmetro.

Linha 7: Usa a tecnologia OpenCV para ler a imagem.

Linha 8 e 9: Verifica se há alguma complicação ao carregar a imagem, caso tenha, será retornada uma mensagem de erro.

Linha 11: Usa o comando do Pytesseract para fazer a extração dos caracteres encontrado na imagem e é salvo na variável “texto”. O parâmetro “lang=’por’” indica ao Tesseract para usar o modelo de idioma português.

Linha 12: retorna o texto extraído da imagem para quem chamou a função.

Linha 14: define a variável “caminho_imagem” com o caminho da imagem que será processada.

Linha 15: chama a função “ler_texto_imagem” com o caminho da imagem e guarda o texto extraído na variável “texto_extraido”.

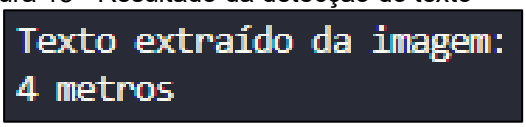
Linha 17 e 18: Exibe uma mensagem no console e adiciona o texto extraído da imagem.

Figura 17 - Imagem para teste

Fonte: Autoria Própria, 2025

Acima há uma imagem com um texto que será analisado através da ferramenta apresentada.

Figura 18 - Resultado da detecção do texto



Fonte: Autoria Própria, 2025

Acima o resultado do texto extraído da imagem através do Pytesseract.

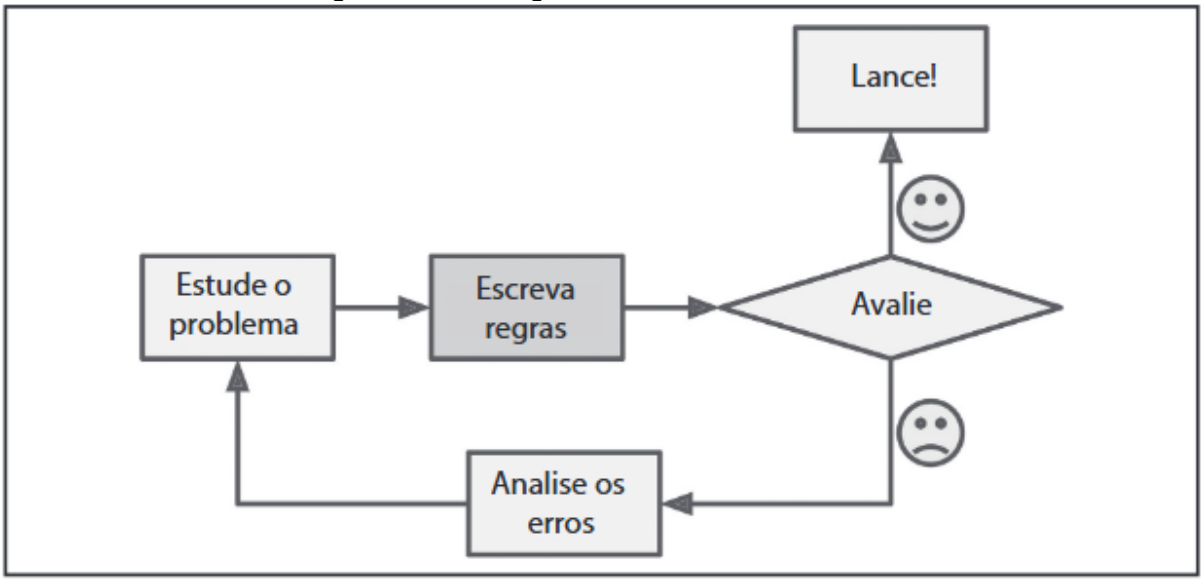
2.2.12 Aprendizado de Máquina

Segundo Almeida (2019), quando pensamos no tópico “aprendizado de máquina” pensamos em um robô que irá dominar o mundo ou irá fazer todas as nossas tarefas do dia a dia, porém isso já existe desde 1990 quando foi criado o “filtro de spam”, se pararmos para notar os pequenos detalhes como “recomendados” entendemos que o aprendizado de máquina já existe em diversas tecnologias. Para

resumir em poucas palavras, o aprendizado de máquina é uma arte ou ciência que permite uma máquina ser treinada e possa aprender com dados.

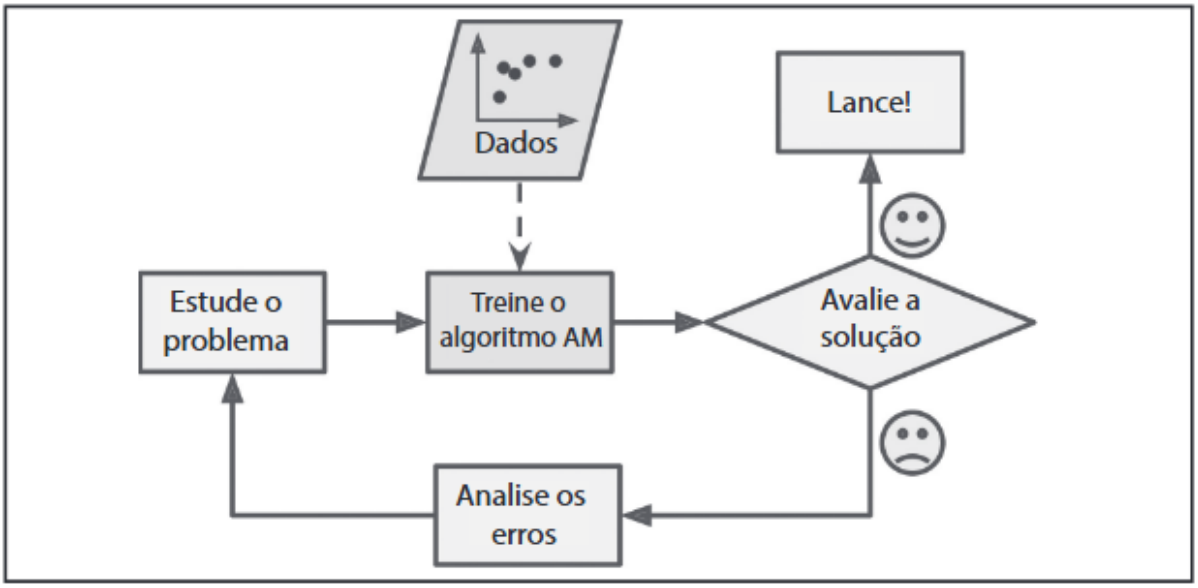
Uma das principais vantagens de se utilizar o Aprendizado de Máquina é sua tecnologia que permite se adaptar, por exemplo, vamos pensar que você esteja criando um programa de “filtro de spam”, caso você fosse criar em uma abordagem comum (sem o Aprendizado de Máquina), você teria que criar diversas regras de forma que quando chegasse um e-mail “Oferta imperdível” ele fosse para a caixa de spam pelas palavras, até o momento seu programa funcionaria, porém se o e-mail passasse a chegar como “Esse desconto só para você”, seria inviável conforme Santos (2019).

Figura 19 - Abordagem tradicional



Fonte: Géron, 2020, p. 4.

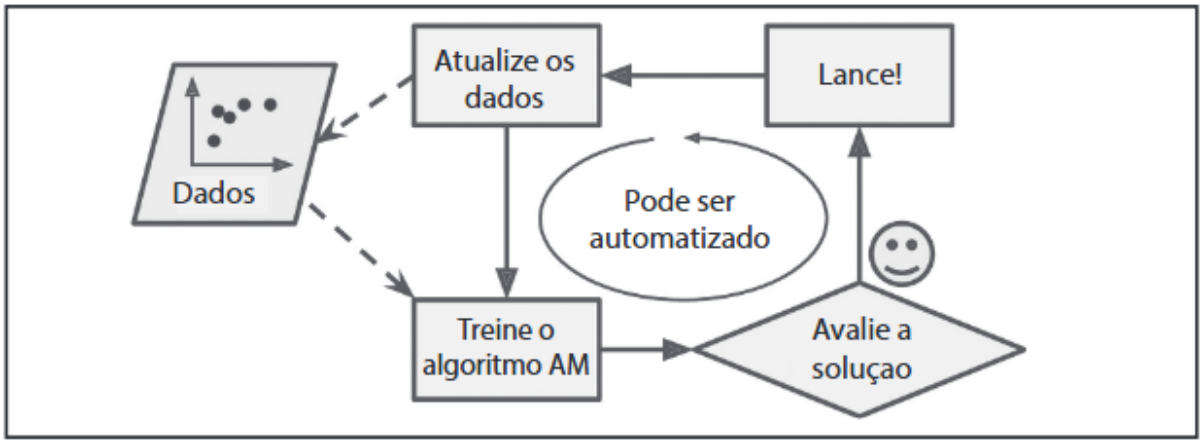
Figura 20 - Abordagem do aprendizado de máquina



Fonte: Géron, 2020, p. 5.

Consoante ao Géron (2020), notamos a diferença entre os tipos de abordagem, na (Figura 2) o caso do spam seria diferente, caso o aprendizado de máquina percebesse que “Esse desconto só para você” se tornou um email frequente na caixa de spam dos outros usuários, automaticamente ele começa a redirecionar todos os próximos e-mails para a caixa de spam. Aqui está um exemplo de como funcionaria essa atualização de dados para que fosse funcional:

Figura 21 - Adaptando-se automaticamente à mudança



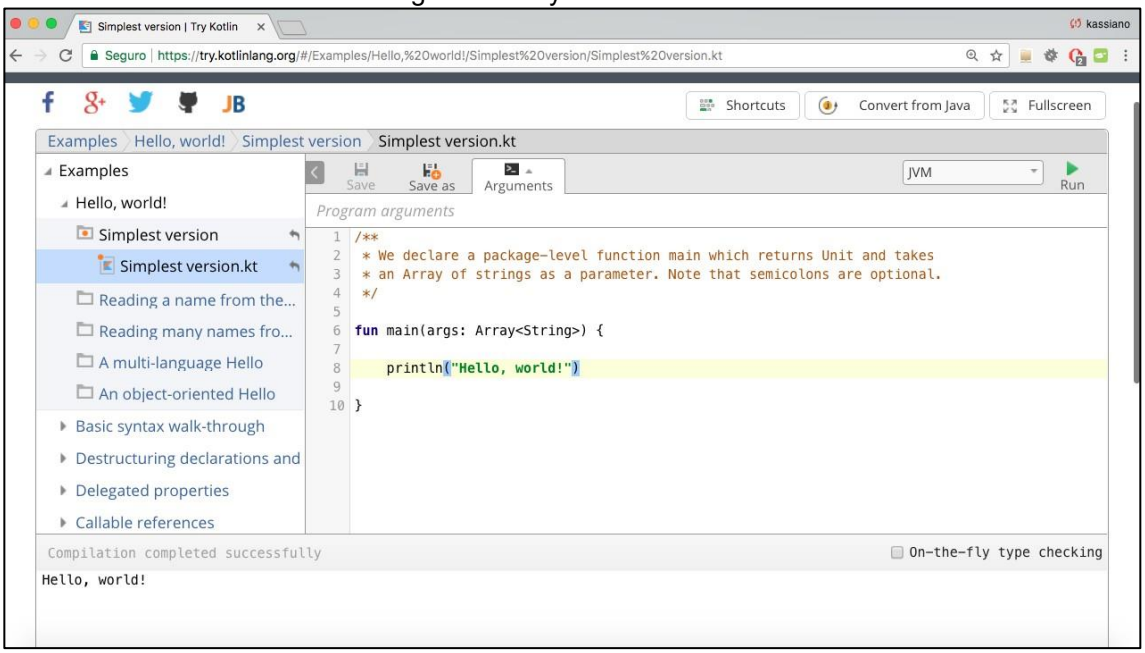
Fonte: Géron, 2020, p. 5.

2.2.13 Kotlin

Segundo Resende (2018) Kotlin é uma linguagem de programação moderna criada para ser eficiente e ter uma boa interoperabilidade com o Java, criada pela JetBrains a mesma criadora da principal IDE do Kotlin (Android Studio) sua ideia era uma opção ao Java para desenvolvimento Android.

O principal objetivo da criação do Kotlin foi ter uma opção mais segura e produtiva do Java, que possa ser utilizada nos mesmos contextos e ambientes, na maioria desses lugares usar o Kotlin pode fornecer códigos menores que executem o mesmo comando que o Java. Entretanto o Kotlin pode ser usado para outras finalidades, como desenvolvimento web com Javascript, pode também ser utilizado em desenvolvimento IOS com o uso da tecnologia KMM (Kotlin Multiplataforma), essa linguagem não é limitada somente a Android e aplicativos móveis, ela tem variedades de campos e integração com bibliotecas que ajudam os desenvolvedores a terem mais código e produtividade com menos tempo (JAMEROV; ISAKOVA, 2017).

Figura 22 - Try Kotlin



Fonte: Resende, 2018, p. 22.

Isto é um site chamado Try Kotlin, onde o usuário pode fazer códigos simples para testes ou aprendizado. Ao entrar na aplicação já é feito automaticamente um código básico imprimindo a frase "Hello World! ", mas vamos explicar melhor tudo que está sendo feito nesta tela.

Veja a primeira linha:

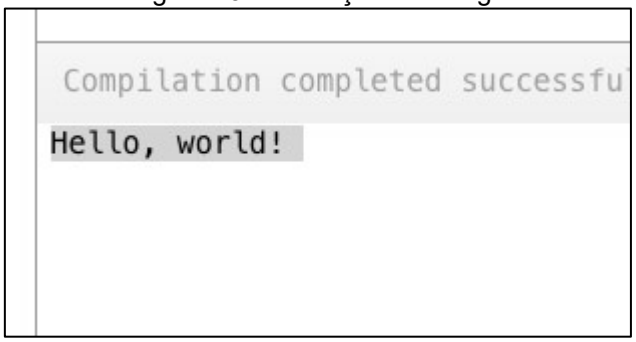
fun main(args: Array<String>)

Esta linha de código começa criando uma função principal do Kotlin chamada "main ", esta função será a principal durante todo o desenvolvimento de qualquer projeto, outras funções também podem ser criadas com outros nomes e outras bibliotecas e valores, logo após esta função recebe um array de string chamado argumentos, isso significa que o programa pode receber dados de fora chamado "args " que traduzindo seria "argumentos ". (RESENDE, 2018).

println("Hello, world!")

Consoante aquela linha, dentro da função "main" imprime o que está dentro das aspas, ou seja, "Hello, world!", resumindo o comando "println" neste código pertence a função "main" que quando compilada exibe no console a frase "Hello, world!", desta forma. (RESENDE, 2018).

Figura 23 - Execução do código



Fonte: Resende, 2018, p. 25.

Da mesma forma que o Java, O Kotlin é uma linguagem "estaticamente tipada" isso significa que o tipo de toda declaração em um programa é reconhecido durante a compilação, e a IDE é capaz de verificar se os métodos e campos que o desenvolvedor está usando existem nos objetos, mas diferente do Java, o Kotlin não requer que o desenvolvedor indique os tipos das variáveis, o Kotlin determina automaticamente dependendo do contexto, podendo até omitir sua declaração. (JAMEROV; ISAKOVA, 2017).

2.2.14 UML (Unified Modeling Language)

Segundo Guedes (2018), a UML é uma linguagem visual de modelagem que auxilia na compreensão do sistema, definindo os requisitos fundamentais para seu funcionamento e manutenção, sendo adotada como norma-padrão internacionalmente.

Conforme Booch (2006), a UML, assim como todas as outras linguagens, exerce a função de estabelecer um vocabulário e regras para a comunicação entre os envolvidos no desenvolvimento do projeto, possibilitando a representação conceitual e física dos elementos que compõem o sistema.

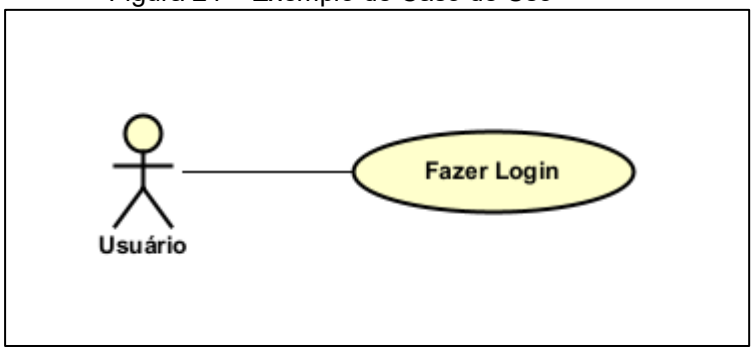
Fowler (2005) destaca que, mesmo a UML não possuindo o mesmo rigor dos métodos matemáticos, sua aplicabilidade a torna valiosa no cotidiano dos programadores. Ela é relevante mesmo sem ser totalmente formal ou detalhada.

Uma casa não se constrói com apenas uma planta, ela é dividida em várias plantas que mostram diversas partes da casa para uma melhor visualização. Devido a isso, existem 13 diagramas no total que podem ser utilizados para descrever o funcionamento tanto estrutural quanto lógico de um sistema (GUEDES, 2018).

Com isso em mente, utilizaremos o Diagrama de Casos de Uso, Diagrama de Atividade, Diagrama de Sequência e, por fim, Diagrama de Máquina de estados.

O diagrama de casos de uso nos mostra uma visão geral dos requisitos do sistema, sem se aprofundar nas especificações das funcionalidades. Através dele podemos entender as funcionalidades do sistema e quais funcionalidades o usuário tem acesso (GUEDES, 2018).

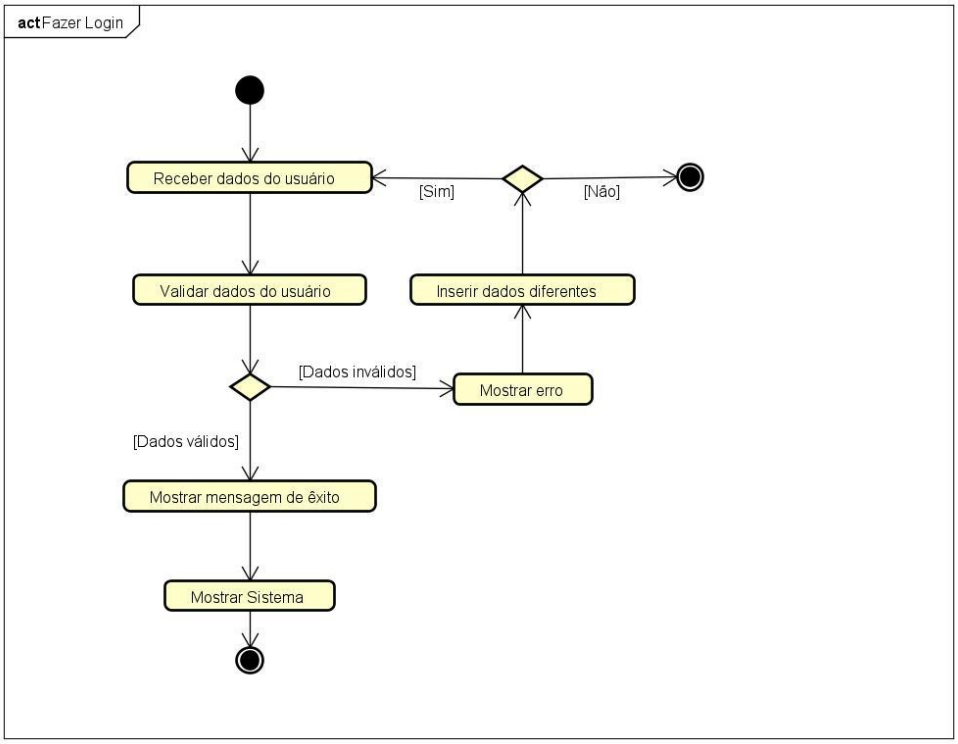
Figura 24 – Exemplo de Caso de Uso



Fonte: Autoria Própria, 2025

O diagrama de atividade demonstra o fluxo de ações e decisões, representando a execução das atividades de forma clara e visual, desde o início até a conclusão do processo. (BOOCH, 2006).

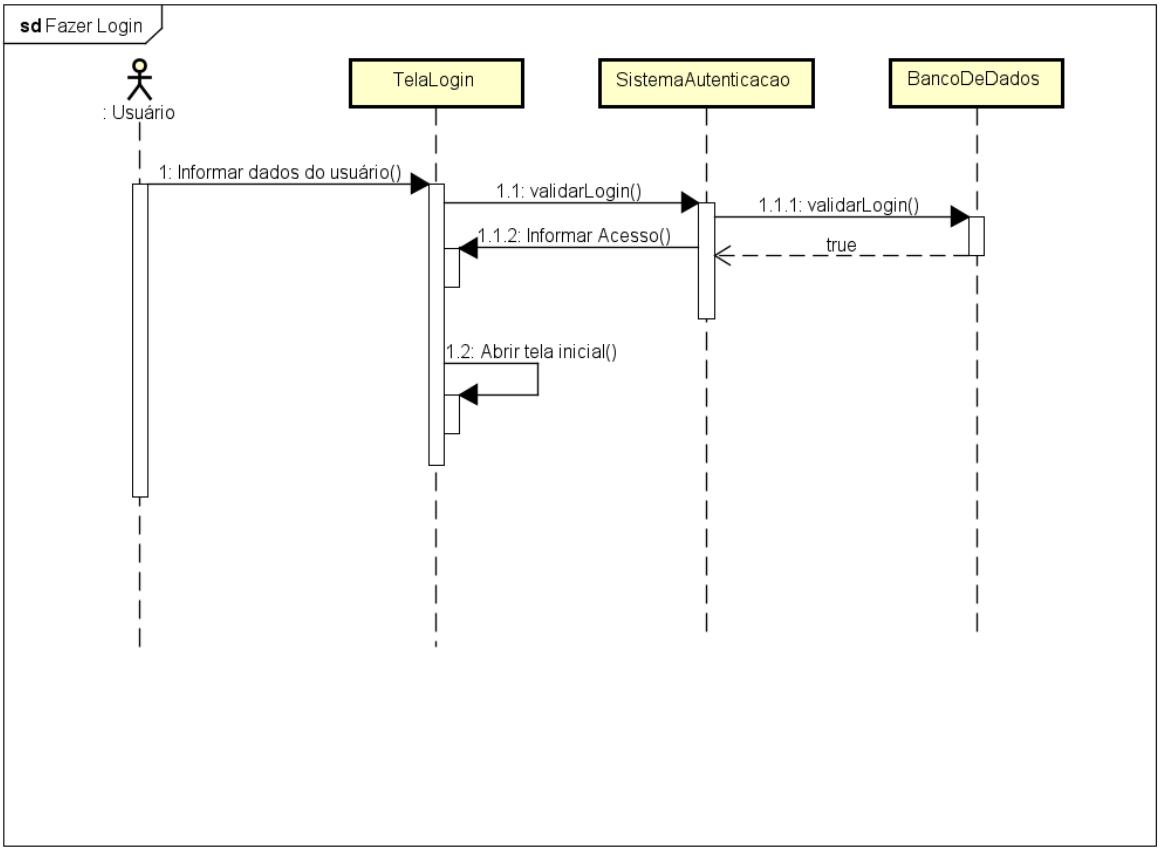
Figura 25 - Diagrama de atividade Fazer Login



Fonte: Autoria Própria, 2025

O diagrama de sequência é uma representação das interações de um objeto durante a execução de um processo, deixando visível as mensagens trocadas entre os objetos, a ordem em que ocorrem e quais métodos são chamados. Para ser construído, ele utiliza casos de uso e o diagrama de classes (GUEDES, 2018).

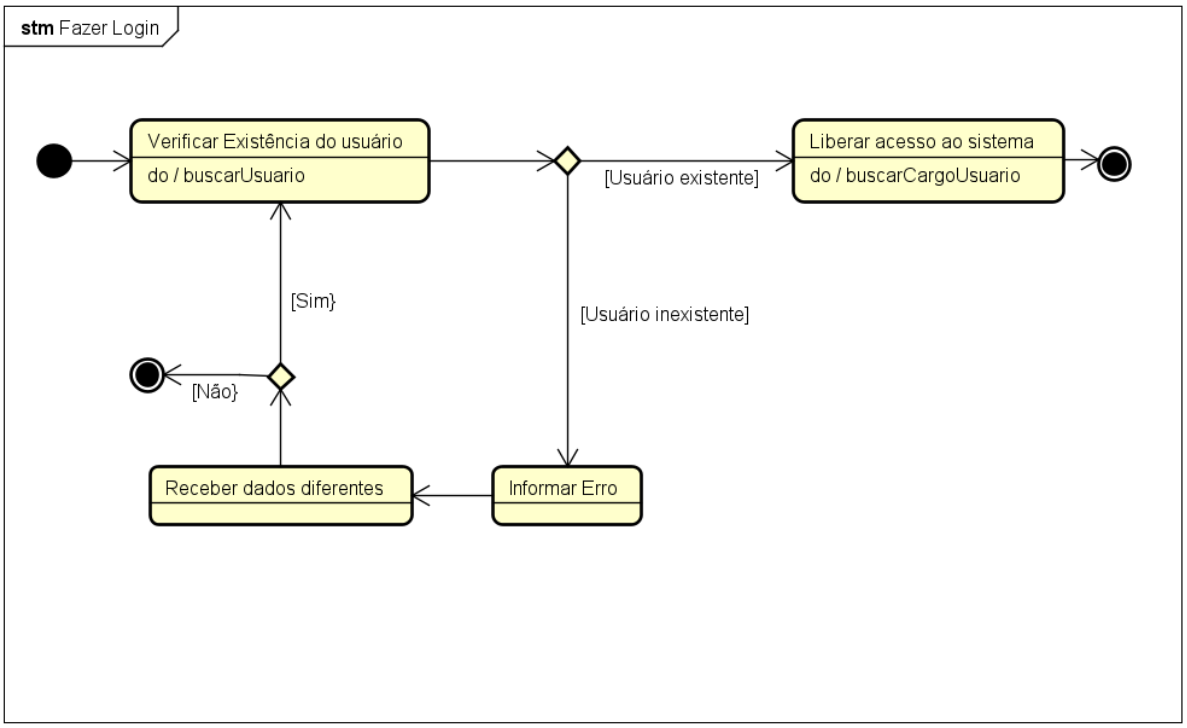
Figura 26 – Diagrama de sequência Fazer Login



Fonte: Autoria Própria, 2025

O diagrama de máquina de estado é utilizado para representar o comportamento de um objeto ou sistema em resposta a eventos, mostrando os estados pelos quais um objeto pode passar ao longo de sua vida (FOWLER, 2005).

Figura 27 - Diagrama de máquina de estado Fazer Login



Fonte: Autoria Própria, 2025

2.2.15 Wireframe

Como aponta Texeira (2015), wireframe é a forma de ajudar a equipe e os clientes a visualizarem as estruturas das páginas, a ordem e os elementos que o compõe, com o intuito de enxergar a ideia e enviar para os desenvolvedores.

Os wireframes são criados para elaborar protótipos da estrutura e da finalidade da aplicação. Eles auxiliam na usabilidade, mostrando os elementos essenciais, a interação do usuário com o aplicativo e seus supostos comportamentos e respostas, conforme Júnior (2021).

2.2.16 Figma

Segundo Rosa (2023), o Figma é uma das principais ferramentas líderes e eficientes no mercado de trabalho, pela sua alta performance e sua aptidão para uso simultâneo por usuários, oferecendo uma ampla variedade de recursos e ferramentas de prototipação de interfaces, facilitando o desenvolvimento do layout de telas.

Conforme a publicação da PM3 (2024), o objetivo do Figma é facilitar o desenvolvedor de software com a criação de uma interface digital, oferecendo serviços

como, criação de componentes, protótipos interativos, realizações de testes e colaboração do time em tempo real de forma remota e simultânea.

3 CONCLUSÃO

REFERÊNCIAS

ALMEIDA, F. C.; SOUSA, A. L. de; PEREIRA, M. G. A. **Avaliação de técnicas de aprendizado de máquina para classificação de dados meteorológicos em sistemas agrícolas**. Revista Brasileira de Ciência do Solo, v. 43, p. 1-10, 2019. Disponível em: https://apct.sede.embrapa.br/cct/article/view/26381/14242. Acesso em: 26 maio 2025.

ARAÚJO, Aline Moura. **Detecção e Destaque em Vídeo de Objetos utilizando YOLO.** 2022. 69 p. Dissertação (Programa de Pós-Graduação em Informática) - Universidade Federal da Paraíba, João Pessoa, 2022.

ATOJI, Rodolpho Iemini. ***Bluetooth e NFC: estudo de caso***. 2010. 58 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2010. Disponível em: https://www.ime.usp.br/~cef/mac499-10/monografias/rodolpho/pdf/mac499-monografia.pdf. Acesso em: 23 maio 2025.

BARELLI, Felipe. **Introdução à Visão Computacional**: Uma abordagem prática com Python e OpenCV. 1. São Paulo: Casa do Código, 2019.

BOOCH, Grady. **UML: Guia do Usuário**. 1. São Paulo: Campus, 2006.

BORGES, Luiz Eduardo. **Python para Desenvolvedores:** Aborda Python 3.3. 1. ed. São Paulo: Novatec Editora Ltda, 2014.

CARRION, Patricia; QUARESMA, Manuela. Internet da Coisas (IoT): Definições e aplicabilidade aos usuários finais. **Human Factors in Design**, Florianópolis, v. 8, n. 15, p. 049–066, 2019. DOI: 10.5965/2316796308152019049. Disponível em: https://www.revistas.udesc.br/index.php/hfd/article/view/2316796308152019049. Acesso em: 24 maio. 2025.

DE MILANO, Danilo; HONORATO, Luciano Barrozo. **Visão Computacional**. UNICAMP Universidade Estadual de Campinas FT Faculdade de Tecnologia, 2014.

EBERMAM, Elivelto; PESENTE, Guilherme Moraes; RIOS, Renan Osório; PULINI, Igor Carlos. **Programação para leigos com Raspberry Pi**. Vitória: Edifes; João Pessoa: Editora IFPB, 2017.

FIGMA. **O que é o Figma?**. Disponível em https://help.figma.com/hc/pt-br. Acesso em: 09 ago. 2025.

FOWLER, Martin. **UML Essencial**: Um Breve Guia para a Linguagem-padrão de Modelagem de Objetos. 3. Porto Alegre: Bookman, 2005.

GÉRON, Aurélien. **Mãos à obra**: aprendizado de máquina com Scikit-Learn, Keras e TensorFlow: conceitos, ferramentas e técnicas para a construção de sistemas inteligentes. 2. ed. Rio de Janeiro: Alta Books, 2020.

GONÇALVES, Eduardo. **SQL:** Uma abordagem para bancos de dados Oracle. 1. ed. São Paulo: Casa do Código, 2014.

GOOGLE. **Firebase**. Disponível em: https://firebase.google.com/. Acesso em: 11 ago. 2025.

GUEDES, Gilleanes. **UML 2: Uma Abordagem Prática**. 3. São Paulo: Novatec Editora Ltda, 2018.

JEMEROV, Dmitry; ISAKOVA, Svetlana. **Kotlin em ação.** 1. São Paulo: Novatec, 2017.

JÚNIOR, Abrão Osório. **Interface gráfica para busca de profissionais criativos**. 2020. Trabalho de Conclusão de Curso (Graduação em Design) – Universidade Federal de Uberlândia, Uberlândia, 2021.

KOVÁCS, Zsolt. **Redes Neurais Artificiais**: Fundamentos e Aplicações. 4. ed. São Paulo: Editora Livraria da Física, 2023.

LEOCÁDIO, Rodolfo R. V. et al. **Detecção de Abelhas Nativas em Colmeias em Campo Utilizando Visão Computacional**. In: WORKSHOP DE COMPUTAÇÃO APLICADA À GESTÃO DO MEIO AMBIENTE E RECURSOS NATURAIS (WCAMA), 12., 2021, Porto Alegre: Sociedade Brasileira de Computação, 2021. p. 59-68.

LIMA, Leandro de Souto. **Reconhecimento de Datas Manuscritas em Prescrições Médicas:** Comparativo entre OCRs com Licença de Software Livre e uma Solução Comercial. 2024. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Universidade Federal de Campina Grande, Campina Grande, 2024.

MAGRANI, Eduardo. **A internet das coisas**. Rio de Janeiro: FGV Editora, 2018.

MARENGONI, M.; STRINGHINI, S. Tutorial: Introdução à Visão Computacional usando OpenCV. **Revista de Informática Teórica e Aplicada**, *[S. l.]*, v. 16, n. 1, p. 125–160, 2010. DOI: 10.22456/2175-2745.11477. Disponível em: https://seer.ufrgs.br/index.php/rita/article/view/rita_v16_n1_p125. Acesso em: 27 maio. 2025.

MATTHES, Eric. **Curso Intensivo de Python:** uma Introdução Prática e Baseada em Projetos à Programação. São Paulo: Novatec Editora Ltda, 2016.

MCKINNEY, Wes. **Python para Análise de Dados**: Tratamento de dados com pandas, NumPy e Jupyter. 3. ed. São Paulo: Novatec Editora Ltda, 2023.

NETO, Aroldo de Sá Vargas. **Reconhecimento Automático de Placas Veiculares do Mercosul Utilizando o Tesseract OCR.** 2023. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Instituto Federal do Espírito Santo, Campus Serra, Serra, 2023.

PACHECO, César Augusto Rodrigues. **Deep Learning**: Conceitos e Utilização nas Diversas Áreas do Conhecimento. 2018. 32 f. Trabalho de Conclusão de Curso (Engenharia de Computação) – Centro Universitário de Anápolis, 2018.

PEREIRA, Gabriel Gutierrez et al. **Automatização de Reconhecimento de Notas Fiscais para Gastos Públicos utilizando OCR e Aprendizado de Máquina**. 2023. Trabalho de Conclusão de Curso (Ciência da Computação) – Universidade Presbiteriana Mackenzie, São Paulo, 2023.

PEREIRA, Matheus de Matos. **Aprendizado Profundo**: Redes LSTM. 2017. 40 f. Trabalho de Conclusão de Curso (Sistemas de Informação) – Universidade Federal da Grande Dourados, 2017.

RESENDE, Kassiano. **Kotlin com Android**: Crie aplicativos de maneira fácil e divertida. 1. ed. São Paulo: Casa do Código, 2017.

RODRIGUES, Arthur**. Quinze caminhões ficam presos em viadutos todo mês em SP e abalam estruturas**. Folha de S.Paulo, 2019. Disponível em:

https://www.folha.uol.com.br/cotidiano/2019/02/quinze-caminhoes-ficam-presos-em-viadutos-todo-mes-em-sp-e-abalam-estruturas. Acesso em: 07 abr. 2025.

ROSA, João Vitor de Moura. **Desenvolvimento de uma aplicação web para simulados preparatórios de vestibular com enfoque no ENEM**. 2023. Trabalho de Conclusão de Curso (Sistemas de Informação) – Universidade Federal de Ouro Preto, João Monlevade, 2023.

SANTOS, José A. dos; SILVA, Felipe L. da. **Um estudo sobre a aplicação de algoritmos de aprendizado de máquina para a detecção de fraudes em sistemas bancários**. Simpósio Brasileiro de Sistemas de Informação (SBSI), v. 1, p. 100-110, 2019. Disponível em: https://sol.sbc.org.br/index.php/sbsi/article/view/5818/5716. Acesso em: 26 maio 2025.

SANTOS, Sandro. **Introdução à IoT**: desvendando a Internet das Coisas. Santa Catarina: Clube de Autores, 2019.

SETTE, Guilherme; CARVALHO, Luiz; BASILE, Antonio. ***Reconhecimento facial pela região dos olhos utilizando OpenCV***. 2021. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Faculdade de Computação e Informática, Universidade Presbiteriana Mackenzie, São Paulo, 2021.

SIQUEIRA, Thiago Senador de. **Bluetooth – Características, protocolos e funcionamento**. 2006. 17 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Universidade Estadual de Campinas, 2006. Disponível em: https://ic.unicamp.br/~ducatte/mo401/1s2006/T2/057642-T.pdf. Acesso em: 23 maio 2025.

TEXEIRA, Fabricio. **INTRODUÇÃO E BOAS PRÁTICAS EM UX DESIGN**. São Paulo: Casa do Código, 2015

ZORN, Erick Nogueira; ANDRIOLLO, Julio Duvique; BORBA, Anderson Adaime de. **Gestão e Automatização na Confecção de Documentos Digitais**. 2024. Trabalho acadêmico – Universidade Presbiteriana Mackenzie, Faculdade de Computação e Informática, São Paulo, 2024.