UNIVERSIDADE DO ESTADO DO AMAZONAS
ESCOLA SUPERIOR DE TECNOLOGIA

CLEMILTON BARROSO DE SOUZA FURTADO

DESENVOLVIMENTO DE UMA SOLUÇÃO DE VISÃO COMPUTACIONAL PARA CONTAGEM DE GARRAFAS TAMPADAS EM UMA LINHA DE PRODUÇÃO FABRIL

Manaus
2024

CLEMILTON BARROSO DE SOUZA FURTADO

DESENVOLVIMENTO DE UMA SOLUÇÃO DE VISÃO COMPUTACIONAL PARA CONTAGEM DE GARRAFAS EM UMA LINHA DE PRODUÇÃO FABRIL

Projeto de pesquisa desenvolvido durante a disciplina de Trabalho de Conclusão de Curso II e apresentada à banca avaliadora do Curso de Engenharia Elétrica da Escola Superior de Tecnologia da Universidade do Estado do Amazonas, como pré-requisito para obtenção do título de Engenheiro Eletricista.

Orientador: Rubens de Andrade Fernandes, Dr.

Manaus

2024

Universidade do Estado do Amazonas – UEA
Escola Superior de Tecnologia - EST
Reitor:
André Luiz Nunes Zogahib
Vice-Reitor:
Kátia do Nascimento Couceiro
Diretora da Escola Superior de Tecnologia:
Jucimar Maia Júnior
Coordenador do Curso de Engenharia Elétrica:
Jozias Parente De Oliveira
Banca Avaliadora composta por: Data da defesa: *11/12/2024*.
Prof. Rubens de Andrade Fernandes (Orientador)
Prof. Jozias Parente De Oliveira
Prof. Wheidima Carneiro de Melo

CIP – Catalogação na Publicação

Furtado, Clemilton

Desenvolvimento de uma solução de visão computacional para contagem de garrafas tampadas em uma linha de produção fabril Z *Clemilton Barroso de Souza Furtado*; orientado por **Rubens de Andrade Fernandes**. – Manaus: 2024.
48 p.: il.

Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica). Universidade do Estado do Amazonas, 2024.

1. Visão Computacional. 2. Garrafas. 3. Contagem.
I. Fernandes, Rubens de Andrade.

CLEMILTONBARROSODESOUZAFURTADO

DESENVOLVIMENTODEUMASOLUC¸ ÃODEVIS ÃO COMPUTACIONALPARACONTAGEMDEGARRAFASEMUMA LINHADEPRODUC¸ ÃOFABRIL

Pesquisadesenvolvidaduranteadisciplina deTrabalhodeConclus˜aodeCursoIIe apresentada`abancaavaliadoradoCursode EngenhariaEl´etricadaEscolaSuperiorde TecnologiadaUniversidadedoEstadodo Amazonas,comopr´e-requisitoparaobten¸c˜ao dot´ıtulodeEngenheiroEletricista.

Notaobtida: 9,7 (Nove vírgula sete)

Aprovadaem11/12/2024

Áreadeconcentra¸c˜ao:ProcessamentoDigitaldeImagenseVis˜aoComputacional

BANCAEXAMINADORA

Rubens de A. Fernandes

Orientador:RubensdeAndradeFernandes,Dr.

Jozias P. de Oliveira

Avaliador:JoziasParentedeOliveira,Dr.

Wheidima Carneiro de Melo

Avaliador:WheidimaCarneirodeMelo,Dr.

Manaus

2024

Dedicatória

Dedico este trabalho ao meu querido avô, Clemilton Souza, cuja sabedoria, dedicação e amor sempre foram uma fonte inesgotável de inspiração para mim. A ele, que com seu exemplo de integridade, me ensinou o valor do esforço e da perseverança. Que, com suas histórias e conselhos, plantou em mim o desejo de aprender e de buscar sempre mais. Agradeço por todo o apoio, pelas palavras de encorajamento e por estar presente em cada etapa da minha jornada.

AGRADECIMENTO

Primeiramente, agradeço a Deus pela força, coragem e sabedoria que me sustentaram ao longo de toda esta jornada. Sua presença foi meu alicerce em cada desafio, me guiando com amor e iluminando o caminho que me trouxe até aqui. Sem Ele, nada disso seria possível.

Aos meus avós, Clemilton Souza, Emir Souza e Simoni Castro, expresso minha eterna gratidão por serem uma fonte inesgotável de inspiração, sabedoria e carinho. Suas histórias, ensinamentos e exemplos de vida foram meu guia em cada passo desta caminhada, e seu amor sempre me deu forças para seguir em frente.

Aos meus pais, Clemir Souza e Orlando Furtado Sobrinho, minha mais profunda gratidão pela educação, pelo apoio incondicional e pelos inúmeros sacrifícios feitos ao longo da minha vida. A confiança, o amor e o incentivo de vocês foram a base que me sustentou e a força que me impulsionou a chegar até aqui.

Aos meus irmãos, Carlos Furtado e Orlando Furtado, agradeço de coração por serem meu porto seguro e por estarem ao meu lado em todos os momentos. O apoio incondicional de vocês e a força que me transmitem são fundamentais na minha caminhada, motivando-me constantemente a ser uma pessoa melhor.

À minha namorada, Camila Melo, minha mais profunda gratidão pelo carinho, paciência e apoio constante que demonstrou em todos os momentos dessa jornada, especialmente nos períodos mais desafiadores. Sua compreensão, palavras de incentivo e presença amorosa foram minha fortaleza e minha inspiração, permitindo-me superar obstáculos e seguir em frente com confiança.

Aos meus familiares, expresso minha mais sincera gratidão por estarem sempre ao meu lado de diversas formas ao longo dessa jornada. Cada gesto de apoio, palavra de incentivo e demonstração de carinho foi fundamental para o meu crescimento pessoal e acadêmico. Sou profundamente grato por fazer parte de uma família tão unida e amorosa, que me fortalece e inspira diariamente.

Ao meu orientador, Rubens Fernandes, agradeço por ter me apresentado ao fascinante mundo da Inteligência Artificial e por ter sido uma fonte constante de incentivo e motivação ao longo dessa jornada. Sua paciência, confiança e entusiasmo não apenas ampliaram meus horizontes acadêmicos, mas também despertaram em mim a paixão por uma área que transformou minha visão de futuro. Foi sob sua orientação que encontrei o caminho para trilhar uma carreira que me inspira a cada dia, e sou imensamente grato por sua dedicação e por acreditar no meu potencial.

Ao meu professor, Manuel Cardoso, manifesto minha mais profunda gratidão por todo o conhecimento compartilhado e pelo apoio constante que me ofereceu ao longo dessa

jornada. Você foi mais do que um professor; foi um verdadeiro mentor, que com dedicação e paciência ajudou a moldar minha trajetória acadêmica e pessoal. O compromisso com o ensino é verdadeiramente inspirador e deixou um impacto profundo em minha formação. Cada ensinamento, orientação e palavra de incentivo que recebi serviu como alicerce para as conquistas que hoje celebro, e por isso serei eternamente grato.

Aos amigos do curso, pela amizade, colaboração e pelos momentos de descontração que tornaram essa caminhada mais leve e divertida. A jornada seria muito mais difícil sem vocês ao meu lado.

Aos amigos do trabalho, pelo incentivo e compreensão ao longo desse processo. A convivência e o apoio de vocês foram fundamentais para que eu pudesse equilibrar meus compromissos acadêmicos e profissionais.

À Universidade do Estado do Amazonas, minha gratidão por ter sido o alicerce de minha formação acadêmica, proporcionando um ambiente de aprendizado, pesquisa e desenvolvimento.

A todos que, de alguma forma, contribuíram para a realização deste trabalho, meu sincero agradecimento.

RESUMO

Este trabalho de conclusão de curso apresenta o desenvolvimento de uma solução de visão computacional para a contagem automatizada de garrafas tampadas em uma linha de produção industrial de bebidas. A proposta busca substituir métodos tradicionais, como aqueles baseados em sensores ópticos, promovendo maior eficiência e automatização no processo de contagem. Nessa lógica, o sistema proposto é composto por uma webcam Logitech C920s e um NVIDIA Jetson Nano, juntamente com técnicas de segmentação e limiarização de cor para processar imagens em tempo real, com uma região de interesse otimizada para focar exclusivamente nos objetos relevantes. Validado em ambiente fabril, o protótipo demonstrou desempenho eficiente, alcançando precisão e *recall* elevados com um F1-score de 99,8%. A principal contribuição do trabalho está na demonstração da viabilidade de integrar tecnologias acessíveis e escaláveis de visão computacional em ambientes fabris, melhorando o controle de qualidade e a eficiência operacional, ao mesmo tempo em que reduz erros e desperdícios. Essa solução oferece uma abordagem para a automação industrial e abre caminho para futuras aplicações e melhorias em diferentes áreas da manufatura.

Palavras chave: Visão computacional, Automação industrial, Controle de qualidade, Contagem de garrafas, Processamento digital de imagem, Limiarização, Segmentação.

ABSTRACT

This final project presents the development of a computer vision solution for automated counting of capped bottles in an industrial beverage production line. The proposal aims to replace traditional methods, such as those based on optical sensors, promoting greater efficiency and automation in the counting process. In this context, the proposed system consists of a Logitech C920s webcam and an NVIDIA Jetson Nano, combined with color segmentation and thresholding techniques to process images in real-time, with an optimized region of interest to focus exclusively on relevant objects. Validated in a factory environment, the prototype demonstrated efficient performance, achieving high precision and recall, with an F1-score of 99.8%. The main contribution of this project lies in demonstrating the feasibility of integrating accessible and scalable computer vision technologies into manufacturing environments, improving quality control and operational efficiency while reducing errors and waste. This solution provides an approach to industrial automation and paves the way for future applications and improvements in various manufacturing sectors. .

Keywords: Computer Vision, Industrial Automation, Quality Control, Bottle Counting, Digital Image Processing, Thresholding, Segmentation.

Lista de Figuras

SUMÁRIO

INTRODUÇÃO

A qualidade do produto é uma importante vantagem competitiva para a sobrevivência no mercado, se tal qualidade estiver abaixo do esperado, pode-se afetar a imagem da empresa, bem como colocar em risco a saúde de seu público (SILVA, 2023). Na indústria de produção de bebidas, a precisão na contagem de produtos é essencial para a eficácia operacional e a satisfação do cliente. Contudo, frequentemente surgem discrepâncias entre os dados registrados por sensores tradicionais, como por exemplo o sensor óptico, e a quantidade real de garrafas produzidas nas linhas de produção. Essas inconsistências podem resultar de várias fontes, como falhas mecânicas, erros de configuração dos sensores ou inadequações no tratamento de dados. Tais erros não apenas comprometem a precisão do inventário e o planejamento de produção, mas impactam diretamente nos custos operacionais e na reputação da empresa, ao permitir que disparidades nos números produzidos conduzam a problemas de sobra ou falta de estoque no mercado. Assim, garantir a precisão na contagem das garrafas torna-se uma questão prioritária, pois influencia diretamente a eficiência da produção, a gestão de custos e a confiabilidade das entregas aos clientes finais.

Nessa lógica, é possível desenvolver um sistema de visão computacional que emprega um hardware composto por uma webcam e um NVIDIA Jetson Nano, com recursos de processamento digital de imagem, para realizar a contagem de garrafas que estão tampadas em uma linha de produção fabril de envase de líquidos.

Sendo assim, este trabalho de conclusão de curso teve como objetivo geral o desenvolvimento e validação de uma solução de visão computacional para a contagem automática de garrafas tampadas em linhas de produção fabril, com o intuito de proporcionar maior controle sobre a produção diária, através de uma comparação direta entre os dados do sensor da linha com o sistema proposto. Ademais, com relação aos objetivos específicos, este trabalho visa o explorar conceitos fundamentais de visão computacional e técnicas de processamento digital de imagens, analisar o funcionamento de linha de produção em fábrica de envase de líquidos, criação e implementação em campo de um sistema para contabilizar garrafas tampadas que saem da máquina envasadora, avaliação do sistema através de métricas de desempenho e a identificação das limitações do sistema.

Esta monografica está organizada nas seguintes partes:

Capítulo 1: Referencial Teórico. Nesta seção apresenta o material teórico utilizados durante a pesquisa.

Capítulo 2: Metodologia. Esta seção está contida em detalhes os materiais e métodos utilizando durante a pesquisa.

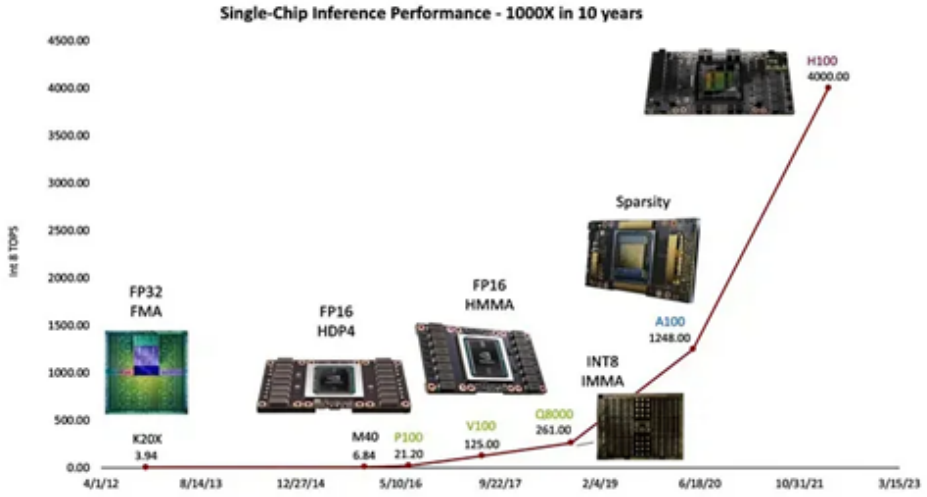
Capítulo 3: Resultados e Discussões. Neste capítulo estão os dados e parâmetros obtidos a partir do protótipo e análises dos resultados.

Conclusão: Relata as dificuldades encontradas no decorrer do trabalho. Aborda os resultados obtidos pelo protótipo desenvolvido no projeto. Finalizando com sugestões para trabalhos futuros com o objetivo do aperfeiçoamento da proposta

1 REFERENCIAL TEÓRICO

Tendo identificado um problema específico na contagem inconsistente de garrafas em uma linha de produção de refrigerantes, será crucial explorar as ferramentas tecnológicas necessárias para implementar uma solução eficaz. Apesar da existência de soluções avançadas que utilizam inteligência artificial, como YOLO e MobileNet, para a detecção de garrafas, muitas dessas tecnologias requerem o uso de GPUs para otimizar a qualidade e a velocidade do processamento. Como evidenciado na Figura 1, a evolução das GPUs tem sido significativa, apresentando avanços notáveis nos chips, o que reforça a importância de considerar alternativas tecnológicas mais acessíveis e eficientes para atender a diversas necessidades industriais. (NVIDIA, 2023)

Figura 1 – Evolução das GPUs apresentadas como evoluções nos Chips



Fonte: (HUANG, 2023)

Contrariamente a essas abordagens, a hipótese central deste trabalho será desenvolver um sistema de visão computacional que utilizará técnicas tradicionais de processamento digital de imagem para a contagem de garrafas tampadas, eliminando a necessidade de hardware especializado como GPUs. Este enfoque não apenas simplificará a implementação tecnológica, mas também reduzirá os custos operacionais, tornando a solução acessível e viável para ambientes de produção com limitações de infraestrutura tecnológica.

Além de entender a aplicabilidade da visão computacional, será importante revisar as tecnologias de captura de imagem adequadas para o contexto industrial, especialmente aquelas que se alinham com as necessidades de simplicidade e custo-benefício do projeto. Neste trabalho, ao invés de utilizar câmeras de alta velocidade, que são comumente empregadas em sistemas avançados de visão computacional, optar-se-á pelo uso de uma webcam Logitech c920s, conforme a Figura 2. Esta escolha visa demonstrar a viabilidade de utilizar equipamentos de baixo custo e amplamente disponíveis no mercado para realizar a

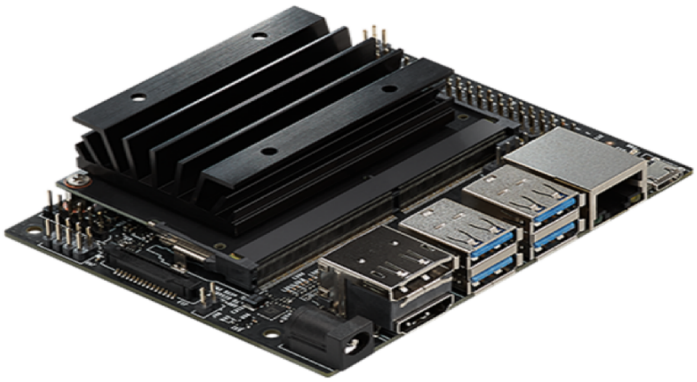
tarefa de contagem de garrafas em uma linha de produção. A webcam será integrada ao sistema de visão computacional, onde técnicas de processamento digital de imagens serão aplicadas para analisar e quantificar as garrafas em movimento, garantindo eficácia sem a necessidade de investimentos significativos em hardware especializado.

Figura 2 – Logitech C920s

Fonte: (ARDILA, 2021)

Para a construção física e integração do sistema de visão computacional, serão escolhidos componentes que permitam uma montagem eficiente e econômica, adequados às necessidades do projeto. O sistema será composto por uma webcam, que servirá como o dispositivo de captura de imagem, montada em um tripé para estabilização e posicionamento adequado. O processamento das imagens será realizado por um Nvidia Jetson Nano, conforme a Figura 3, um computador compacto e eficiente. Embora o Jetson Nano esteja equipado com uma GPU capaz de realizar processamento gráfico avançado, neste projeto, optar-se-á por não utilizar essa capacidade. A decisão de não usar a GPU visa demonstrar que mesmo sem recorrer à aceleração gráfica, será possível alcançar resultados satisfatórios na contagem de garrafas tampadas utilizando apenas o processamento centralizado na CPU. Além disso, um monitor, teclado e mouse completarão a configuração, proporcionando uma interface de usuário simples para operação e monitoramento do sistema.

Figura 3 – Nvidia Jetson Nano



Fonte: (BARBA-GUAMÁN; NARANJO; ORTIZ, 2020)

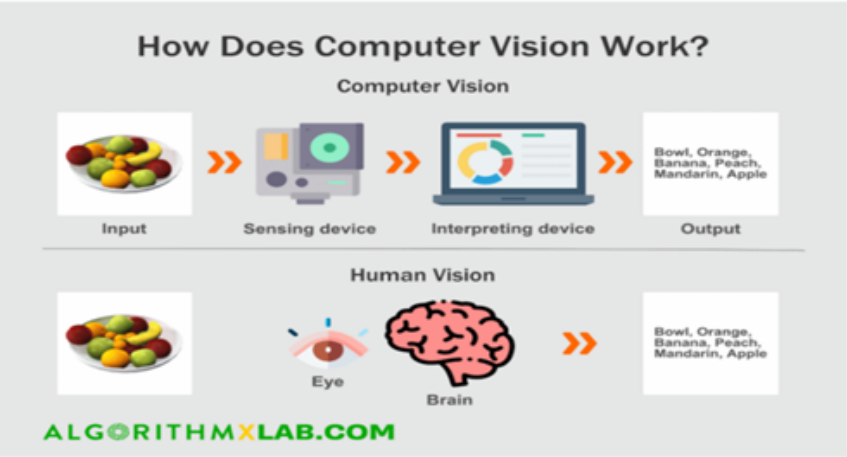
No que diz respeito ao software, será fundamental uma revisão abrangente da literatura envolvendo técnicas de detecção e contagem de objetos, com um foco especial na identificação de garrafas. Esta revisão teórica ajudará a fundamentar e orientar o desenvolvimento dos algoritmos que serão empregados no sistema de visão computacional. Ao explorar estudos prévios e metodologias estabelecidas, buscar-se-á adaptar e otimizar essas técnicas para o contexto específico de linhas de produção de refrigerantes, onde a precisão e a velocidade são cruciais. Além da implementação dessas técnicas, o sistema será configurado para exibir os resultados de contagem em tempo real em um monitor. Esta funcionalidade não apenas facilitará o monitoramento contínuo da produção por parte dos operadores, mas também permitirá a visualização instantânea das detecções realizadas pela webcam, proporcionando um feedback imediato e visível sobre o desempenho do sistema.

1.1 VISÃO COMPUTACIONAL

A capacidade de capturar e processar imagens é uma importante habilidade dos seres vivos, pois as imagens fornecem uma grande quantidade de informações à respeito do ambiente. Porém, a extração de informações úteis de imagem não é uma tarefa trivial, pois estima-se que cerca de 70% do processamento cerebral seja destinado ao tratamento das imagens capturadas pelos olhos. Por causa da extensa aplicação de análise de imagem em diversas áreas, o processamento de imagem tem crescido bastante e se tornando extremamente importante nas últimas décadas. (ARTERO, 2009)

Podemos definir visão computacional como a área de estudo que tenta repassar para máquinas a incrível capacidade da visão, conforme a Figura 4. A visão consiste em captar imagens, melhorá-las, separar as regiões de interesse ou objetos de interesse de uma cena, extrair várias informações dependendo da imagem analisada, como, por exemplo, forma, cor e textura, e, finalmente, relacionar as imagens com outras vistas previamente. (BACKES; JUNIOR, 2016)

Figura 4 – Funcionamento da visão computacional

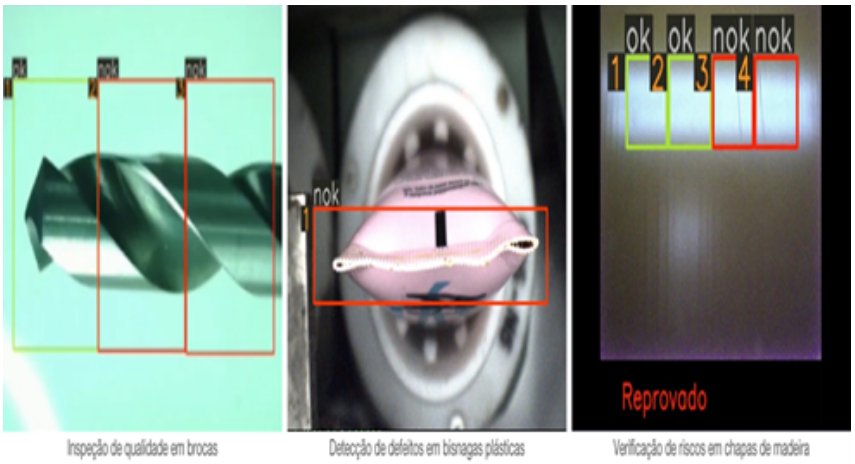


Fonte: (Prime Control, 2023)

Disponíveis em diversas configurações e aplicáveis em diferentes situações industriais, os sistemas de inspeção por computador tem como principal objetivo o auxilio ou até mesmo a substituição da visão humana no ambiente industrial. As vantagens de sistemas como estes, quando comparados à inspeção convencional, evidenciam-se ainda mais quando a linha de produção deve ser monitorada em alta velocidade e com precisão. (STIVANELLO, 2004)

Como referenciado na Figura 5, os sistemas de visão computacional são implementados em linhas de montagem para detectar defeitos em produtos, assegurando a manutenção de padrões de qualidade rigorosos e minimizando custos associados a falhas e retrabalhos. Essa tecnologia é especialmente valiosa em ambientes onde a precisão e a consistência são críticas.

Figura 5 – Visão computacional em fábricas



Fonte: (WEG, 2024)

Na contagem de garrafas tampadas, a visão computacional permite uma monitoração

eficiente e precisa, superando os métodos tradicionais que frequentemente dependem de contagens manuais ou mecânicas. Utilizando câmeras e software especializado, o sistema pode identificar e contar garrafas tampadas em alta velocidade à medida que passam na linha de produção. Isso não apenas aumenta a acurácia do inventário, mas também otimiza o uso dos recursos ao reduzir o desperdício e garantir que a quantidade exata de produto seja processada e embalada. Adicionalmente, a capacidade de integrar dados de contagem com sistemas de gerenciamento de produção permite ajustes em tempo real, melhorando a resposta a demandas de produção e potencializando a eficácia operacional global.

1.2 PROCESSAMENTO DIGITAL DE IMAGEM

O processamento digital de imagens (PDI) é uma área de suma importância dentro da computação gráfica e visão computacional, dedicada à manipulação de imagens digitais por meio de computadores. Uma imagem pode ser definida como uma função bidimensional, f(x, y), em que x e y são coordenadas espaciais (plano), e a amplitude de f em qualquer par de coordenadas (x, y) é chamada de intensidade ou nível de cinza da imagem nesse ponto. Quando x, y e os valores de intensidade de f são quantidades finitas e discretas, chamamos de imagem digital. (GONZALEZ, 2010)

O objetivo principal do PDI é melhorar a qualidade das imagens ou extrair informações úteis e dados a partir delas. Este campo interdisciplinar combina técnicas de matemática, estatísticas, física e programação, e é amplamente utilizado em diversas aplicações que vão desde a medicina até a segurança, passando por entretenimento, monitoramento ambiental e controle industrial.

1.2.1 Fundamentos de PDI

O PDI começa com a aquisição de imagem, uma fase fundamental onde a imagem analógica é capturada por dispositivos como câmeras ou scanners e transformada em uma imagem digital por meio dos processos de amostragem e quantização. Durante a amostragem, a imagem contínua é convertida em uma representação discreta, onde os valores de intensidade de luz são medidos em intervalos regulares. Segue-se a quantização, como mostrado na Figura 6, onde esses valores amostrados são mapeados para os níveis de intensidade mais próximos disponíveis, geralmente representados em uma escala de bits finita. Essa transformação da informação visual em dados digitais é crucial, pois a precisão na amostragem e na quantização influencia diretamente a qualidade da imagem digital resultante e afeta todas as operações de processamento subsequentes. Uma quantização e amostragem inadequadas podem levar à perda de detalhes importantes, limitando a eficácia das técnicas de processamento de imagem aplicadas posteriormente.

Figura 6 – Quantização de imagem

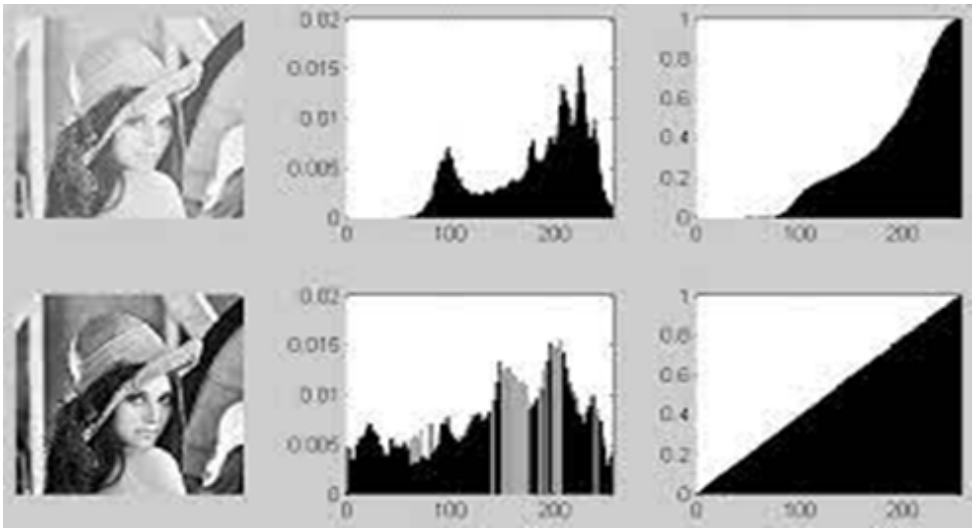
Fonte: (ALENCAR, 2012)

Após a aquisição, a filtragem e o realce de imagem são geralmente os próximos passos. A filtragem, como mostrado na Figura 7, é utilizada para remover ruídos ou realçar características dentro da imagem. Existem diversos filtros, como os de suavização, que ajudam a reduzir a variação de intensidade entre os pixels, e os filtros de aguçamento, que realçam as bordas dos objetos na imagem. O realce de imagem, como mostrado na Figura 8, por sua vez, é focado em melhorar a aparência visual da imagem ou destacar certos detalhes que são importantes para análises específicas. Técnicas como ajuste de contraste, manipulação de histograma e correção de cores são comumente aplicadas para aumentar a clareza visual e a definição das imagens.

Figura 7 – Filtragem de imagem

Fonte: (LUQUE, 2016)

Figura 8 – Realce de imagem por equalização do histograma



Fonte: (GONZALEZ, 2010)

A restauração de imagens é outro componente vital do PDI, que visa corrigir distorções ou degradações que ocorreram durante o processo de aquisição. Diferente da filtragem, que também pode remover detalhes importantes junto com o ruído, a restauração procura modelar o processo de degradação e reverter os efeitos adversos para recuperar a imagem original. Isso é particularmente útil em cenários onde as imagens são sujeitas a condições adversas de captura, como movimento da câmera ou iluminação insuficiente.

Além destes, outros processos avançados de PDI incluem a segmentação de imagem, conforme a Figura 9, que divide a imagem em regiões ou objetos significativos baseados em critérios como cor, intensidade ou textura. Isso facilita a análise mais detalhada de partes específicas da imagem e é amplamente utilizado em aplicações como diagnósticos médicos por imagem, onde diferentes tecidos precisam ser claramente identificados e analisados.

Figura 9 – Segmentação de imagens

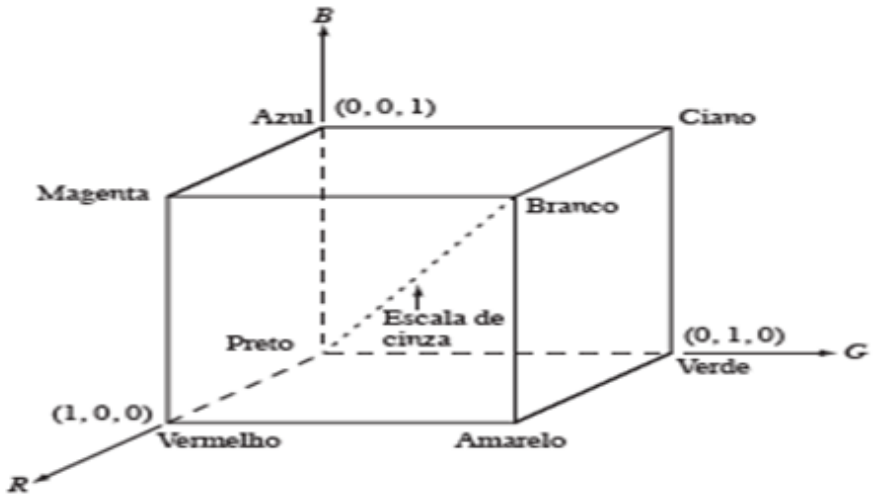
Fonte: (IMOBILIS, 2020)

1.2.2 Modelo de cor RGB

Este modelo é um subespaço do espaço euclidiano composto pelo cubo unitário mostrado na Figura 10. As cores aparecem com seus componentes primários de vermelho, verde e azul. Os valores de R, G e B estão ao longo de três eixos. Ou seja, no eixo vermelho, no eixo verde e no eixo azul estão as intensidades de cada cor. O ciano está localizado no vértice do cubo onde as cores verde e azul têm valor máximo e o valor do vermelho é zero; as coordenadas são (R, G, B) = (0, 1, 1). Da mesma forma, magenta, que é a combinação de vermelho e azul, está localizado nas coordenadas (R, G, B) = (1, 0, 1); e amarelo (mistura de verde com vermelho) está localizado em (R, G, B) = (1, 1, 0). O preto está posicionado na origem do sistema e o branco no vértice oposto à origem. A escala de cinza está localizada na diagonal que vai do preto ao branco, as demais cores encontram-se dentro do cubo. (PÉREZ, 2009)

Em imagens digitais de 8 bits por canal, cada componente pode assumir valores entre 0 e 255, o que permite gerar 16.777.216 cores, sendo a combinação máxima das três componentes (255, 255, 255) resulta na cor branca, enquanto a ausência de todas as componentes (0, 0, 0) resulta na cor preta.

Figura 10 – Esquema do cubo de cores RGB



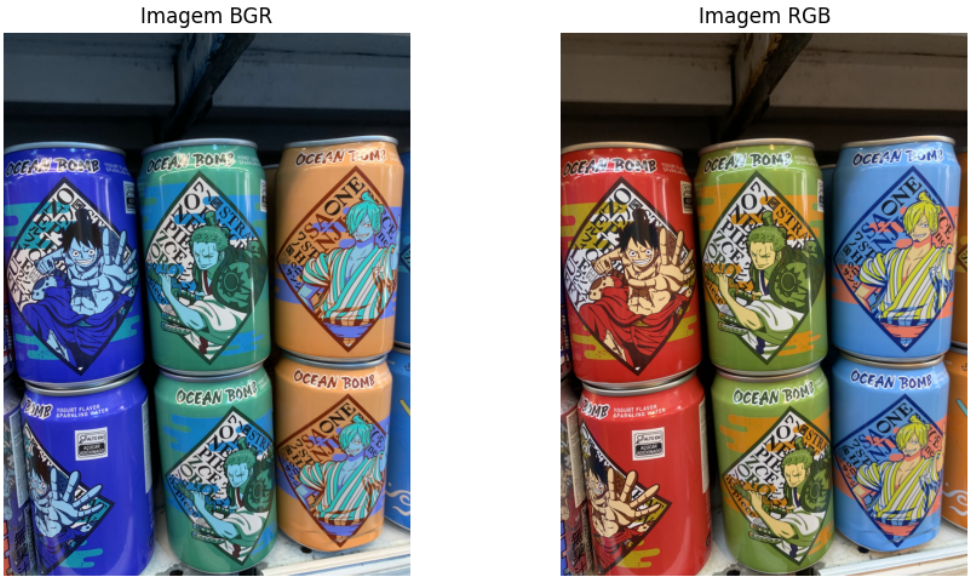
Fonte: (GONZALEZ, 2010)

1.2.3 Modelo de cor BGR

O modelo de cor BGR (Blue, Green, Red) é amplamente utilizado em sistemas computacionais para representar cores em imagens digitais. Ele é similar ao modelo RGB, mas organiza os canais de forma diferente: primeiro o azul (Blue), seguido pelo verde (Green) e, por último, o vermelho (Red). Esse modelo é especialmente adotado em bibliotecas como o OpenCV, que utiliza BGR como padrão devido à sua estrutura interna de armazenamento de dados, invertendo a ordem usual do RGB. (SINGH; INDU, 2021)

A diferença do modelo de cor pode ser visto na Figura 11.

Figura 11 – Modelo de cor BGR e RGB



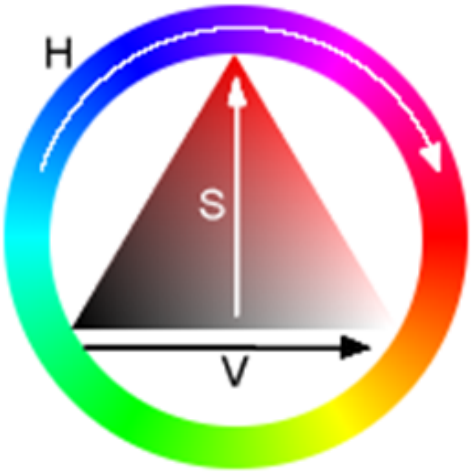
Fonte: Autoria própria

1.2.4 Modelo de cor HSV

O modelo de cor HSV, conforme a Figura 12, separa a cor em três componentes: Matiz (*Hue*), Saturação (*Saturation*) e Valor (*Value*).

- **Matiz**: Responsável por detectar as partes da imagem que parecem ser semelhantes às cores percebidas, ou seja, Azul, Verde, Vermelho ou uma combinação delas (SINGH; INDU, 2021). Medido em graus, variando de 0 a 360, o matiz é definido pelo comprimento de onda dominante na mistura espectral e é disposto em um círculo de cores, onde 0° corresponde ao vermelho, 120° ao verde e 240° ao azul.
- **Saturação**: A "coloração de um impulso em relação ao seu próprio brilho" (SINGH; INDU, 2021). Indica a pureza da cor, variando de 0 a 1. Uma saturação de 0 significa uma cor completamente acinzentada, enquanto uma saturação de 1 indica uma cor pura, sem mistura de branco.
- **Valor**: Responsável por determinar onde a imagem será mais clara ou mais escura (SINGH; INDU, 2021). Também conhecido como brilho, refere-se à intensidade da cor, variando de 0 a 1. Um valor de 0 corresponde a preto, enquanto um valor de 1 representa o brilho máximo da cor.

Figura 12 – Sistema de cores HSV



Fonte: (WIKIPÉDIA, 2023)

1.2.5 Transformação de BGR para HSV

A transformação de uma imagem no formato BGR para HSV envolve a conversão dos valores de cada pixel de suas componentes Azul, Verde e Vermelha para as componentes de Matiz, Saturação e Valor. Este processo é realizado por meio de uma série de cálculos que mapeiam o cubo de cores BGR para o espaço de cores HSV.

Em primeiro plano, os valores BGR são normalizados para o intervalo [0, 1], dividindo cada componente por 255, caso seja uma imagem de 8 bits, conforma a equação 1.

$$B' = \frac{B}{255}, \quad G' = \frac{G}{255}, \quad R' = \frac{R}{255} \tag{1}$$

Em seguida, encontra-se o valor máximo e mínimo das componentes normalizadas, conforme as equações 2 e 3. Posteriormente, conforme a equação 4, é possível calcular a variação das componentes normalizadas.

$$C_{\max} = \max(B', G', R') \tag{2}$$

$$C_{\min} = \min(B', G', R') \tag{3}$$

$$\Delta = C_{\max} - C_{\min} \tag{4}$$

Logo, a componente **Valor** é determinada como o valor máximo das componentes normalizadas, conforme a equação 5.

$$V = C_{\max} \tag{5}$$

Em segundo plano, a **Saturação** é calculada com base na equação 6. Nessa lógica, se o valor da componente máxima é zero, a saturação é zero, caso contrário, é a razão entre o delta (4) pela componente máxima (2).

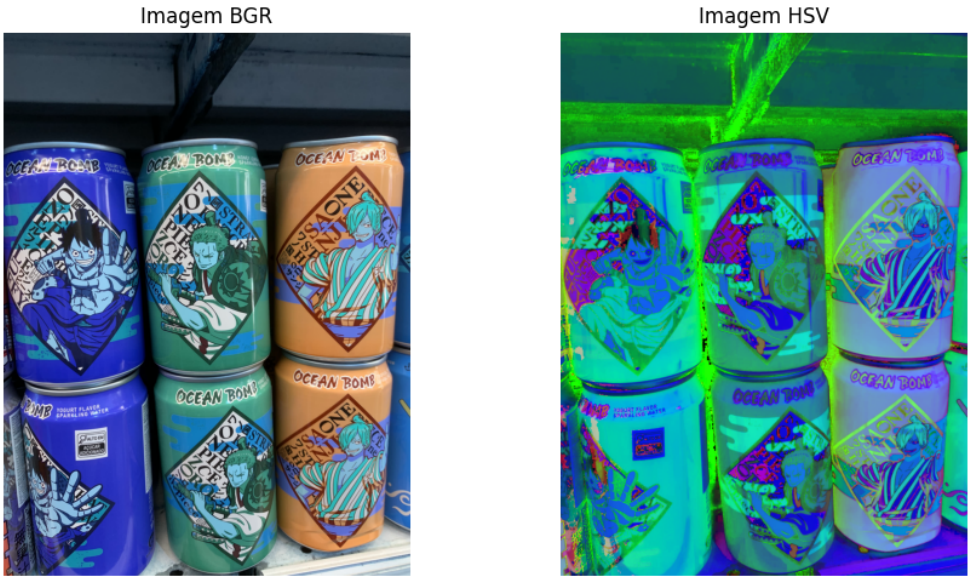
$$S = \begin{cases} 0, & C_{\max} = 0 \\ \frac{\Delta}{C_{\max}}, & C_{\max} \neq 0 \end{cases} \tag{6}$$

Por fim, o **Matiz** é calculado conforme a equação 7, considerando a componente de cor com o valor máximo ($C_{\max}$) e aplicando diferentes fórmulas para cada caso. Quando a componente máxima é a vermelha ($C_{\max} = R'$), o matiz é determinado pela diferença entre as componentes verde (G') e azul (B'), normalizada pelo intervalo de cor e ajustada por um fator que o converte para graus, posicionando-o no segmento correspondente ao vermelho no círculo de cores. Caso a componente máxima seja a verde ($C_{\max} = G'$), a diferença entre as componentes azul (B') e vermelha (R') é utilizada, com um deslocamento adicional de $+2$ para alinhar o matiz ao segmento verde do espectro. Quando a componente máxima é a azul ($C_{\max} = B'$), a diferença entre as componentes vermelha (R') e verde (G') é usada, com um deslocamento de $+4$, posicionando o matiz no segmento azul. O fator $60°$ multiplica cada cálculo para escalá-lo ao intervalo de $0°$ a $360°$, cobrindo todo o espectro visível, enquanto o uso de $\mod 6$ garante que o matiz permaneça dentro do intervalo esperado, assegurando a continuidade e a consistência na representação das cores no espaço HSV.

$$H = \begin{cases} 60° \times \frac{G'-B'}{\Delta} \mod 6, & C_{\max} = R' \\ 60° \times \left(\frac{B'-R'}{\Delta} + 2\right), & C_{\max} = G' \\ 60° \times \left(\frac{R'-G'}{\Delta} + 4\right), & C_{\max} = B' \end{cases} \tag{7}$$

A Figura 13 apresenta a conversão de BGR para HSV, ilustrando claramente como cada componente é transformada para determinar o matiz e os demais valores do modelo HSV.

Figura 13 – Conversão de imagem BGR para HSV usando Python



Fonte: Autoria própria

1.2.6 Segmentação e limiarização

A segmentação é uma etapa crítica no processamento digital de imagens, sendo responsável pela divisão de uma imagem em regiões que satisfaçam alguns critérios de similaridade (semelhança nos níveis de cinza) ou descontinuidades (mudanças bruscas nos níveis de cinza) pré-definidos. Essa técnica é amplamente utilizada em diversas aplicações, como reconhecimento de padrões, detecção de objetos, análise de imagens médicas, entre outras.

A limiarização (Thresholding) é uma técnica fundamental no processamento digital de imagens, desempenhando um papel central nas aplicações de segmentação de imagem. Devido às suas propriedades intuitivas, simplicidade de implementação e velocidade computacional, a limiarização é amplamente utilizada.

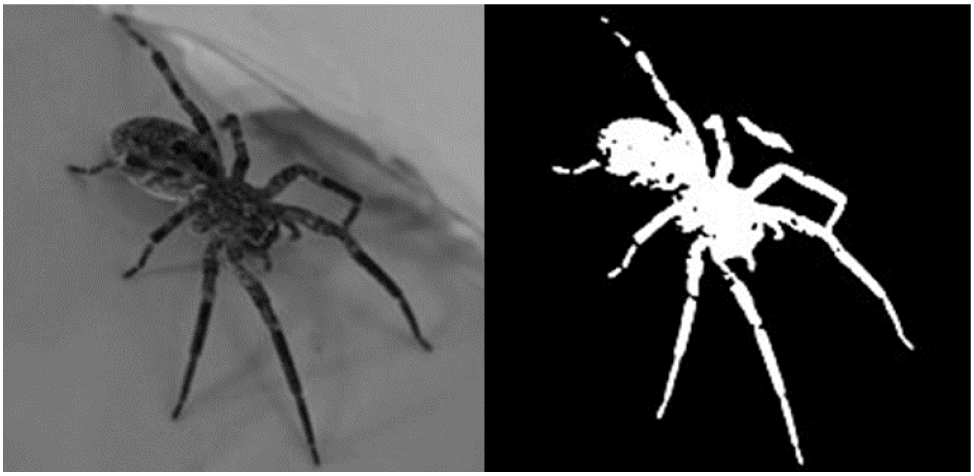
A limiarização de intensidade é um processo que divide uma imagem em regiões com base nos valores de intensidade dos pixels, conforma a Figura 14. A ideia básica é selecionar um limiar T que separa os pixels em duas classes: pixels de objeto (valores de intensidade maiores que T) e pixels de fundo (valores de intensidade menores ou iguais a T).

Esta operação é descrita pela equação:

$$g(x,y) = \begin{cases} 1, & \text{se } f(x,y) > T \\ 0, & \text{se } f(x,y) \leq T \end{cases} \tag{8}$$

onde f(x,y) é a imagem original e g(x,y) é a imagem binarizada. (GONZALEZ, 2010)

Figura 14 – Limiarização de imagens



Fonte: (WIKIPÉDIA, 2024)

1.2.7 Segmentação por limiarização baseado no modelo de cor HSV

A segmentação por limiarização baseada em cor no espaço HSV é uma técnica eficaz no processamento digital de imagens, especialmente em situações onde a iluminação varia ou quando a cor é um critério significativo para a detecção e segmentação de objetos. Tal segmentação envolve a definição de intervalos específicos para as componentes de matiz, saturação e valor que correspondem à cor do objeto de interesse. O processo pode ser descrito pelos seguintes passos:

1. **Definição dos Intervalos de Cor:** Com base no conhecimento prévio da cor do objeto, são definidos intervalos de valores para H, S e V. Por exemplo, para segmentar objetos vermelhos, pode-se definir um intervalo de matiz que capture as variações de vermelho.
2. **Aplicação da Limiarização:** A imagem é percorrida pixel a pixel, e cada pixel é verificado se suas componentes de HSV estão dentro dos intervalos definidos:

$$g(x,y) = \begin{cases} 1, & \text{se } H_1 \leq H(x,y) \leq H_2 \text{ e } S_1 \leq S(x,y) \leq S_2 \\ & \text{e } V_1 \leq V(x,y) \leq V_2 \\ 0, & \text{caso contrário} \end{cases} \tag{9}$$

   Onde H_1 e H_2 são os limites do intervalo de matiz, S_1 e S_2 são os limites do intervalo de saturação, e V_1 e V_2 são os limites do intervalo de valor.

3. **Geração da Máscara Binária:** Como resultado da limiarização, é gerada uma máscara binária onde os pixels que caem dentro dos intervalos especificados são

marcados como 1 (branco), representando a presença do objeto, e os demais pixels são marcados como 0 (preto).

2 METODOLOGIA

O desenvolvimento da solução de visão computacional para contagem de garrafas tampadas em uma linha de produção de fábrica de refrigerante foi baseado em uma abordagem prática e estruturada em etapas. A pesquisa aplicada combina revisão bibliográfica, experimentação e desenvolvimento de um sistema prototípico. O objetivo principal desta metodologia é garantir que o sistema seja capaz de detectar e contar garrafas tampadas com alta precisão em um ambiente industrial.

A construção do projeto foi fundamentada no uso de tecnologias avançadas de visão computacional para processamento de imagens em tempo real. O fluxo de trabalho está organizado em quatro fases principais: preparação do hardware e software, desenvolvimento do algoritmo de detecção de garrafas, implementação do sistema de contagem e a aplicação prática no ambiente de fábrica.

Este capítulo detalha os procedimentos, ferramentas e métodos que serão adotados em cada uma dessas fases, com o intuito de garantir uma solução robusta e eficiente que atenda aos requisitos operacionais da indústria de bebidas. O método hipotético-dedutivo será empregado para validar a eficácia das técnicas de visão computacional e aprimorar o sistema durante as fases de experimentação e testes.

2.1 MATERIAIS UTILIZADOS

Para o desenvolvimento do sistema de visão computacional voltado à contagem de garrafas em uma linha de produção, foi necessário utilizar um conjunto específico de materiais que garantisse a captura de imagens em alta resolução e processamento eficiente. A escolha dos componentes foi feita com base em suas capacidades técnicas, compatibilidade com as bibliotecas de visão computacional e pela disponibilidade dos materiais no momento. Esses fatores asseguraram que o sistema pudesse funcionar de maneira eficaz no ambiente real de uma linha de produção industrial.

2.1.1 Nvidia Jetson Nano Developer Kit

A plataforma selecionada para o desenvolvimento deste projeto foi o NVIDIA Jetson Nano Developer Kit, uma solução robusta e amplamente reconhecida por sua capacidade de realizar tarefas intensivas de processamento de imagens e aprendizado de máquina em tempo real. Equipado com a arquitetura CUDA (Compute Unified Device Architecture), o Jetson Nano permite a aceleração gráfica por meio do processamento paralelo das GPUs, o que o torna altamente eficiente para aplicações de visão computacional.

Contudo, neste trabalho, a funcionalidade CUDA não foi utilizada, uma vez que o foco principal do projeto está voltado para o PDI, sem a necessidade de aceleração gráfica oferecida pelo CUDA. Mesmo sem o uso dessa tecnologia, o Jetson Nano Developer Kit

demonstrou-se plenamente capaz de atender às exigências do projeto, proporcionando um desempenho adequado para o desenvolvimento e execução dos algoritmos propostos, além de garantir uma excelente relação entre eficiência energética e capacidade de processamento.

A escolha do Jetson Nano se deu pela sua facilidade de acesso no laboratório. O dispositivo conta com uma GPU de 128 núcleos Maxwell, uma CPU ARM quad-core e 4 GB de memória LPDDR4, além de suporte para câmeras MIPI CSI.

As especificações completas do Jetson Nano Developer Kit podem ser vistas na Tabela 1, que detalha suas capacidades e configurações, reforçando sua adequação para aplicações em ambientes industriais com visão computacional.

Tabela 1 – Especificações do Jetson Nano Developer Kit

Especificação	**Detalhes**
GPU	128-core NVIDIA Maxwell™ GPU
CPU	Quad-core ARM® Cortex®-A57 MPCore Processor
Memória	4 GB 64-bit LPDDR4
Armazenamento	microSD (não incluído)
Vídeo	HDMI 2.0 and DisplayPort 1.2
Câmera	MIPI CSI-2 DPHY lanes
Conectividade	Gigabit Ethernet, Optional Wi-Fi via USB adapter
Portas USB	4x USB 3.0
Portas GPIO	40-pin expansion header
Energia	5V 4A via barrel jack or micro-USB
Dimensões	100 mm x 80 mm x 29 mm
Peso	80 g

Fonte: (NVIDIA DEVELOPER, 2024) (Adaptado)

2.1.2 Logitech C920s Pro HD Webcam

A captura de imagens em alta definição foi realizada utilizando a Logitech C920s Pro HD Webcam, escolhida por sua confiabilidade na obtenção de imagens de qualidade, mesmo em ambientes com variações de iluminação. A câmera suporta resolução de até 1080p a 30 quadros por segundo (fps), o que garante a precisão necessária para as etapas subsequentes de detecção e contagem de objetos. Além disso, sua capacidade de foco automático e o amplo campo de visão de 78° tornam o dispositivo ideal para monitoramento constante e preciso em aplicações industriais. As especificações técnicas completas da câmera estão apresentadas na Tabela 2, que detalha suas principais características.

Tabela 2 – Especificações da Logitech C920s Pro HD Webcam

Especificação	**Detalhes**
Resolução Máxima	1080p/30fps - 720p/30fps
Campo de Visão (FOV)	78°
Foco	Foco Automático
Tecnologia de Vídeo	H.264
Microfone	Microfones Estéreo Integrados
Obturador de Privacidade	Sim
Conectividade	USB-A Plug-and-Play
Dimensões (CxLxA)	43,3 mm x 94 mm x 71 mm
Peso	162 g

Fonte:(LOGITECH, 2024) (Adaptado)

2.1.3 Monitor

Um monitor foi utilizado para visualização em tempo real das imagens capturadas pela câmera, bem como para o acompanhamento do processo de desenvolvimento e ajustes necessários no sistema. A interface gráfica exibida no monitor permitiu a análise contínua da eficácia do sistema de contagem.

Tabela 3 – Especificações do Monitor LCD com Visão Completa

Especificação	**Detalhes**
Tamanho da Tela	7 polegadas
Resolução	1024 x 600 pixels
Tipo de Tela	LCD, Full View
Conectividade	HDMI compatível
Entradas de Alimentação	5V USB ou 12V DC
Modos de Exibição	Suporte para exibição horizontal e vertical
Dimensões	164 mm x 110 mm x 26 mm
Peso	Aproximadamente 280 g

Fonte:(GRANADO, 2024) (Adaptado)

2.1.4 Tripé

Para estabilizar a câmera e assegurar que as imagens fossem capturadas sob ângulos consistentes, foi utilizado um tripé ajustável. Essa ferramenta evitou distorções de imagem devido a movimentos involuntários, garantindo que a visão da câmera permanecesse alinhada à linha de produção simulada.

Tabela 4 – Especificações do Tripé Suawin Universal 1,8m

Especificação	**Detalhes**
Modelo	Tripé Suawin Universal
Altura Máxima	1,8 metros
Material	Alumínio
Compatibilidade	Câmeras DSLR, câmeras digitais e smartphones
Cabeça do Tripé	Cabeça com ajuste para panorâmica e inclinação
Suporte para Celular	Incluso
Peso Máximo Suportado	Até 2 kg
Altura Mínima	50 cm
Peso	650 g
Cor	Preto

Fonte:(MERCADO LIVRE, 2024) (Adaptado)

2.2 RECURSOS DE SOFTWARES UTILIZADOS

2.2.1 Python

O Python é linguagem de programação dinâmica e orientada a objeto, que pode ser utilizada no desenvolvimento de qualquer tipo de aplicação, científica ou não. (COELHO, 2007)

A linguagem de programação Python começou a ser desenvolvida ao final dos anos 80, na Holanda, por Guido van Rossum. Guido foi o principal autor da linguagem e continua até hoje desempenhando um papel central no direcionamento da evolução. Guido é reconhecido pela comunidade de usuários do Python como "Benevolent Dictator For Life" (BDFL), ou ditador benevolente vitalício da linguagem. (COELHO, 2007)

A linguagem inclui diversas estruturas de alto nível (listas, dicionários, data/hora, complexos e outras) e uma vasta coleção de módulos prontos para uso, além de frameworks de terceiros que podem ser adicionados. Também inclui recursos encontrados em outras linguagens modernas, tais como geradores, introspecção, persistência, metaclasses e unidades de teste. Multiparadigma, a linguagem suporta programação modular e funcional, além da orientação a objetos. Mesmo os tipos básicos no Python são objetos. A linguagem é interpretada através de bytecode pela máquina virtual Python, tornando o código portátil. Com isso é possível compilar aplicações em uma plataforma e rodar em outros sistemas ou executar direto do código-fonte. (BORGES, 2014)

Python é um software de código aberto (com licença compatível com a General Public License [GPL], porém menos restritiva, permitindo que o Python seja incluído incorporado em produtos proprietários). A especificação da linguagem é mantida pela Python Software Foundation (PSF). (BORGES, 2014)

No contexto da programação orientada a objetos, o conceito de classe é fundamental. Uma classe pode ser definida como um modelo que descreve as características (atributos) e os comportamentos (métodos) de um determinado tipo de dado. A partir dessa classe, é possível criar objetos, que são instâncias concretas da classe, contendo valores específicos para os atributos definidos. Dessa forma, a orientação a objetos permite a organização do código de forma modular e reutilizável, facilitando o desenvolvimento e a manutenção dos programas.

Os métodos são funções associadas a uma classe que definem os comportamentos que os objetos dessa classe podem executar. Além dos métodos, Python também possibilita a criação de funções, que são blocos de código reutilizáveis que não pertencem a uma classe específica. As funções são amplamente utilizadas para executar tarefas específicas e para promover a organização e reutilização do código. Uma função em Python é definida utilizando a palavra-chave `def` e pode receber parâmetros e retornar valores para a função que a chamou.

2.2.2 Bibliotecas e módulo

Para o desenvolvimento deste projeto de conclusão de curso, serão utilizadas as bibliotecas **NumPy** e **OpenCV**.

NumPy (abreviação de Numerical Python) é uma biblioteca Python de código aberto para computação científica. Ela permite trabalhar com arrays e matrizes de forma natural. A biblioteca contém uma longa lista de funções matemáticas úteis, incluindo algumas para álgebra linear, transformações de Fourier e rotinas para geração de números aleatórios. LAPACK, uma biblioteca de álgebra linear, é usada pelo módulo de álgebra linear do NumPy se estiver instalada no seu sistema. Caso contrário, o NumPy fornece sua própria implementação. LAPACK é uma biblioteca conhecida, originalmente escrita em Fortran, na qual o MATLAB também se baseia. De certa forma, o NumPy substitui algumas das funcionalidades do MATLAB e Mathematica, permitindo prototipagem interativa rápida. (IDRIS, 2015)

Durante o desenvolvimento, a biblioteca será importada como **`np`**, que é uma convenção amplamente utilizada para simplificar o acesso às suas funcionalidades no código. No contexto deste projeto, NumPy é essencial para definir os limites de cor da máscara utilizada no processamento de imagens e para realizar a contagem de pixels que atendem aos critérios estabelecidos, desempenhando um papel crucial na análise e manipulação dos dados obtidos.

Por outro lado, a OpenCV (Open Source Computer Vision) é uma biblioteca de programação, de código aberto, desenvolvida inicialmente pela Intel Corporation. O OpenCV implementa uma variedade de ferramentas de interpretação de imagens, indo desde operações simples como um filtro de ruído, até operações complexas, tais como a

análise de movimentos, reconhecimento de padrões e reconstrução em 3D (MARENGONI; STRINGHINI, 2009). No projeto em questão, OpenCV desempenha um papel fundamental, permitindo a captura de vídeo pela câmera, a leitura dos frames e a execução do processamento digital de imagens. Para acessar as funcionalidades da OpenCV, utilizamos o módulo **`cv2`**, que é a interface principal para Python, onde por meio desse módulo, é possível instanciar objetos e chamar funções específicas para realizar operações como filtragem, detecção e manipulação de imagens.

Ao utilizar `cv2`, estamos chamando métodos que pertencem à classe que define a biblioteca OpenCV, de modo a realizar ações específicas sobre as imagens capturadas. Dessa forma, `cv2` nos proporciona as ferramentas necessárias para capturar, processar e analisar os dados visuais, que são essenciais para o sucesso do projeto de visão computacional.

O módulo **`datetime`** é uma biblioteca do Python que disponibiliza classes para manipulação de datas e horas. Apesar de sua complexidade e da ampla gama de funcionalidades que oferece, é frequentemente empregado para capturar a data e hora atuais, realizar operações de aritmética com datas e formatar datas e horas para exibição. Neste projeto, a biblioteca datetime é utilizada com um propósito informativo específico: permitir que o usuário verifique os horários exatos das detecções em tempo real. Isso facilita o monitoramento preciso de eventos e a coleta de dados cronológicos essenciais para análises subsequentes.

2.3 ESTRUTURA MODULAR DO PROJETO

O projeto foi desenvolvido com uma estrutura modular que promove organização, reutilização de código e clareza na implementação das funcionalidades. Essa estrutura é composta por dois arquivos principais: `main.py` (ANEXO 01) e `wutils.py` (ANEXO 02). A divisão em módulos foi planejada para separar a lógica principal de execução, contida em `main.py`, das classes e funções auxiliares, implementadas em `wutils.py`.

2.3.1 Arquivo main.py

O arquivo `main.py` é o ponto de entrada do programa, sendo responsável por inicializar e executar o algoritmo principal. Ele importa a classe `ContadorGarrafaAI`, que encapsula toda a lógica de visão computacional e processamento de dados.

A função `main()` tem como objetivo principal criar um objeto chamado `algoritmo` que instância a classe `ContadorGarrafaAI` e invocar seu método `iniciar()`, que contém o superlaço principal para captura, processamento e exibição de imagens. Isso demonstra a separação clara entre a lógica de execução e a implementação das funcionalidades, garantindo modularidade.

2.3.2 Arquivo wutils.py

O arquivo `wutils.py` concentra a implementação de todas as classes auxiliares usadas no projeto. Cada classe foi desenvolvida com uma responsabilidade específica, seguindo os princípios da programação orientada a objetos:

1. **Camera:** Gerencia a captura de quadros da câmera em tempo real. É responsável por abrir o fluxo de vídeo, capturar frames e liberar os recursos ao final da execução.
2. **PDI:** Implementa as técnicas de PDI, como a aplicação de máscara e a contagem de pixels brancos em uma região de interesse. Essa classe permite segmentar objetos relevantes no vídeo.
3. **Contador:** Controla a lógica de contagem das garrafas tampadas. Baseado na quantidade de pixels brancos detectados, ele incrementa o contador de garrafas somente quando os critérios são atendidos, prevenindo contagens duplicadas.
4. **Desenhos:** Gerencia a renderização de informações no vídeo, como retângulos para destacar a região de interesse e textos que exibem o número de garrafas contadas, além de outras informações relevantes, como o horário atual.
5. **ContadorGarrafaAI:** Integra todas as classes mencionadas e orquestra o fluxo do programa, desde a captura de quadros até a exibição do vídeo processado.

2.3.3 Funcionamento geral do sistema

Durante a execução, o arquivo `main.py` inicia o superlaço principal chamando o método `iniciar` da classe `ContadorGarrafaAI`.

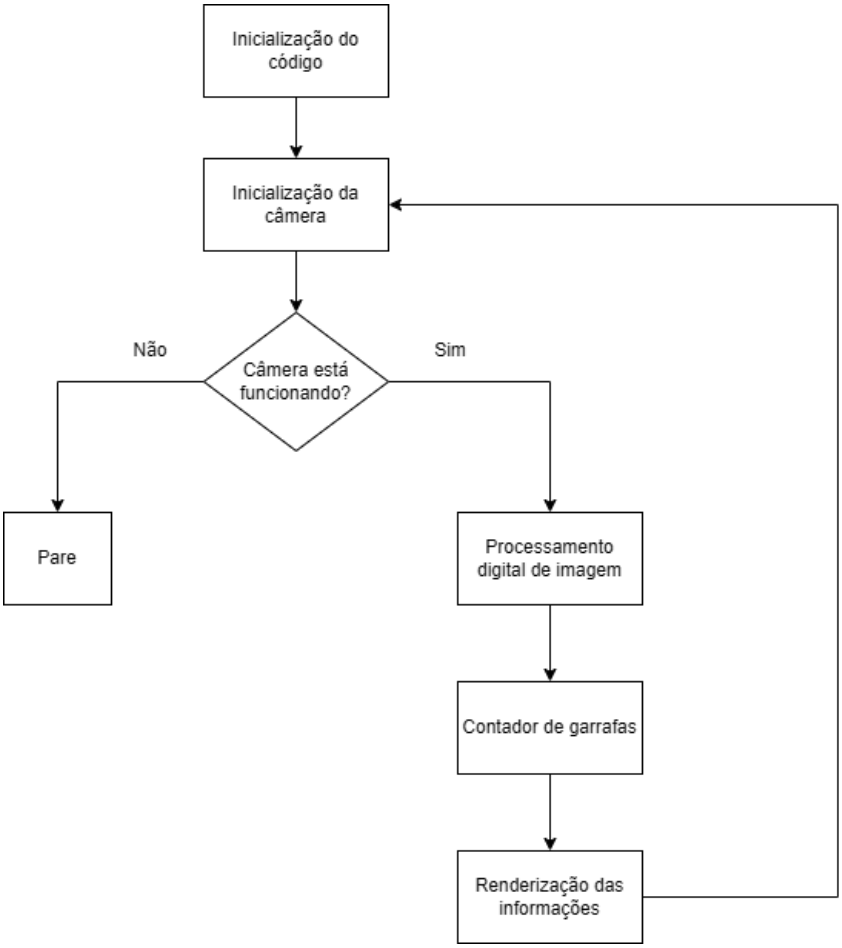
Código 1 – Instância da classe ContadorGarrafaAI

```
1 def main():
2     algoritmo = ContadorGarrafaAI()
3     algoritmo.iniciar()
4
5 if __name__ == "__main__":
6     main()
```

Esse método implementa um fluxo estruturado que abrange cinco etapas principais: captura de frames em tempo real utilizando a classe `Camera`, aplicação de técnicas de processamento digital de imagens pela classe `PDI` para detectar e contar pixels brancos na região de interesse, atualização da contagem de garrafas com base nos critérios estabelecidos pela classe `Contador`, renderização das informações no vídeo por meio da classe `Desenhos` e, finalmente, exibição do vídeo processado em uma janela intitulada ”Vídeo Principal”.

A Figura 15 apresenta o fluxograma geral do sistema proposto, onde os detalhes específicos de cada bloco do fluxograma serão descritos nas subseções correspondentes.

Figura 15 – Fluxograma geral do projeto



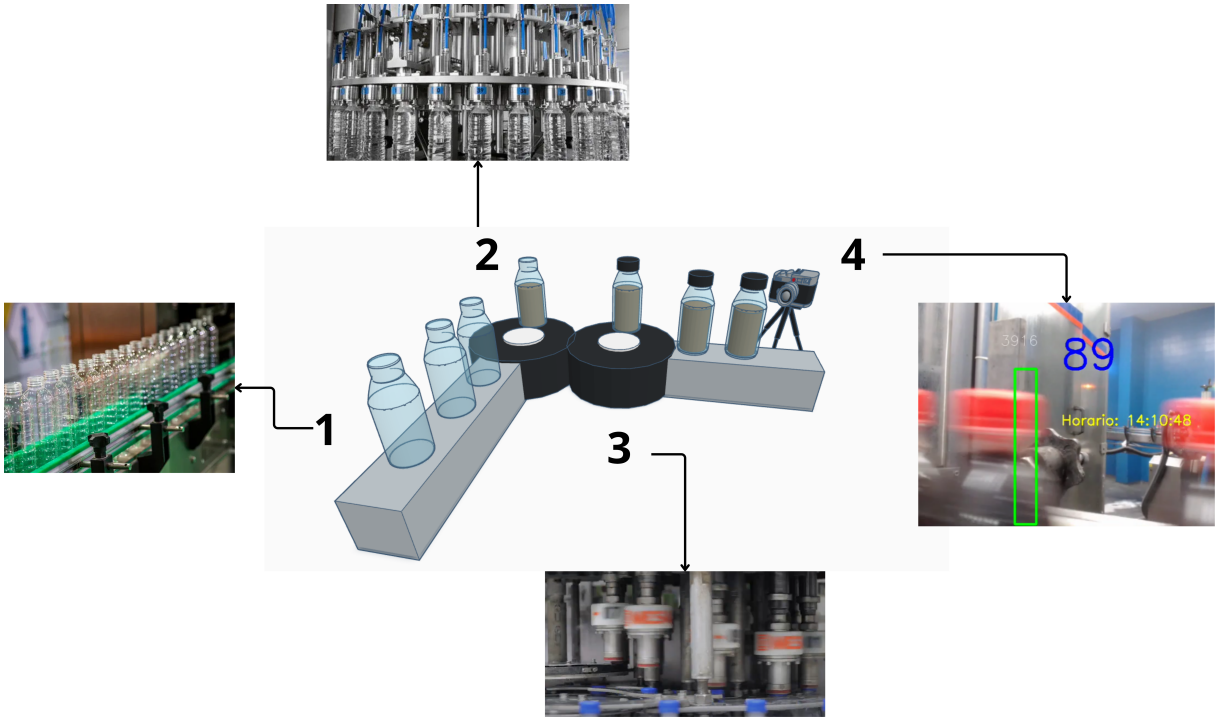
Fonte: Autoria própria

Em seguida, a Figura 16 traz uma representação tridimensional do processo, destacando a configuração física do sistema e a interação entre os componentes.

O processo inicia com a entrada das garrafas na linha de produção, onde recipientes vazios são organizados e transportados pela esteira. Na segunda etapa, as garrafas passam pelo processo de envase, sendo preenchidas com o líquido de maneira precisa por meio de máquinas envasadoras automatizadas.

Na sequência, ocorre a terceira etapa, em que as garrafas são tampadas, uma vez que as tampas são aplicadas e fixadas para assegurar a vedação correta e a qualidade do produto. Por fim, entra em ação o sistema de visão computacional, responsável pela quarta etapa do processo. Nesse estágio, a webcam captura imagens em tempo real das garrafas tampadas na linha de produção e realiza a contagem.

Figura 16 – Arquitetura da proposta



Fonte: Autoria própria

2.4 INICIALIZAÇÃO DA CÂMERA

Quando se trabalha com visão computacional, a inicialização da câmera é um passo fundamental que prepara o sistema para a captura contínua e análise de imagens. Este processo envolve configurar a câmera para começar a transmitir imagens que serão processadas pelo software, geralmente utilizando bibliotecas especializadas como o OpenCV, que é amplamente utilizada para tais aplicações devido à sua robustez e ampla gama de funcionalidades.

No contexto do projeto, após instanciar a classe `ContadorGarrafaAI` ao objeto `algoritmo`, temos como atributo a criação de 4 novos objetos, sendo eles `camera`, `pdi`, `contador` e `desenhos`, ambos instanciando respectivamente as classes, `Camera`, `PDI`, `Contador` e `Desenhos`.

Código 2 – Atributos da classe ContadorGarrafaAI

```
1 class ContadorGarrafaAI:
2     def __init__(self):
3         self.camera = Camera()
4         self.pdi = PDI()
5         self.contador = Contador()
6         self.desenhos = Desenhos(208, 142, 46, 332)
```

2.4.1 Configuração da câmera com OpenCV

A configuração da câmera é realizada por meio da classe `Camera` que encapsula a funcionalidade de inicialização e gerenciamento do dispositivo de captura de vídeo. A câmera é ativada criando-se uma instância da classe `VideoCapture`, atribuída ao atributo `video`. Essa instância permite que o sistema se conecte à câmera e inicie a captura de frames em um loop contínuo. O índice fornecido ao `VideoCapture` é configurado como 1, especificando qual dispositivo de captura será utilizado; valores como 0 geralmente se referem à câmera integrada do dispositivo, enquanto valores maiores indicam câmeras externas conectadas ao sistema.

Código 3 – Atributo da classe Camera

```
1 class Camera:
2     def __init__(self):
3         self.video = cv2.VideoCapture(1)
```

2.4.2 Processamento frame a frame

Uma vez que a câmera está inicializada e transmitindo, o processamento de imagens ocorre de maneira contínua, com cada imagem capturada sendo tratada como um quadro independente. Esse processo é realizado dentro de um superlaço, onde o OpenCV lê e processa cada frame sequencialmente a partir do fluxo de vídeo da câmera.

O superlaço continua indefinidamente, processando um frame após o outro, até que uma condição específica de saída seja atendida. Essa condição pode ser o atingimento de um objetivo determinado pela aplicação ou pode ser interrompido por uma ação do usuário.

2.4.3 Verificação da funcionalidade da câmera

Dentro do superlaço do script principal, responsável por processar cada frame capturado pela câmera, o código realiza uma verificação fundamental a cada iteração para garantir o funcionamento adequado da câmera. Essa verificação é implementada por meio do método `obter_frame()` do objeto `camera`.

Ao chamar o método `obter_frame()`, duas variáveis, `ret` e `frame`, são inicializadas com o resultado do método `read()` do objeto `video`. O método `read()` retorna dois valores: um booleano, armazenado em `ret`, que indica se o frame foi capturado com sucesso, e o próprio frame capturado, armazenado na variável `frame`.

Código 4 – Método obter_frame da classe Camera

```
1 class Camera:
2     def obter_frame(self):
```

```
3           ret, frame = self.video.read()
4           if not ret:
5               print("Erro ao ler o quadro do video. Finalizando o loop.
                    ")
6               return None
7           return frame
```

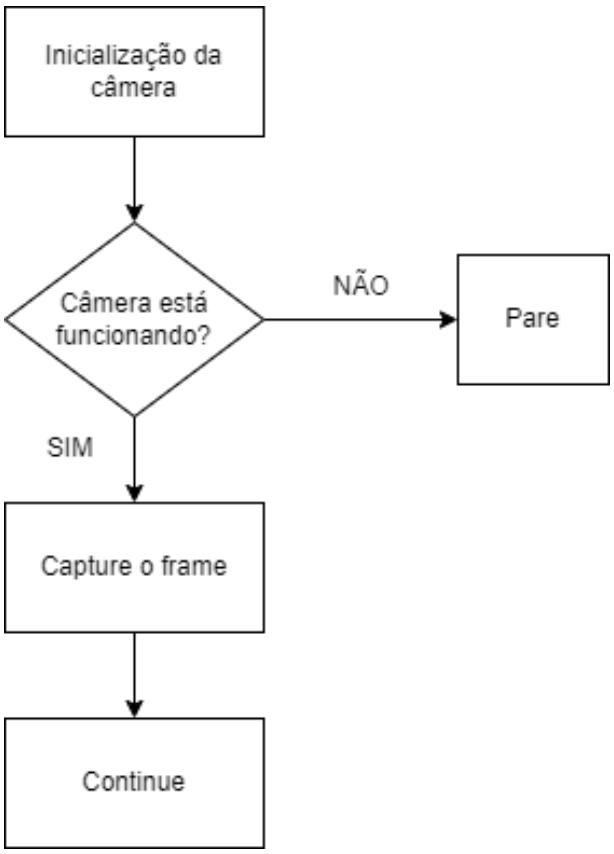
Se o valor de `ret` for `False`, uma mensagem de erro é exibida no console para informar a falha na leitura da câmera, e o método `obter_frame()` retorna o valor `None`. Em seguida, no superlaço principal, o valor retornado pelo método é verificado. Caso o valor de `frame` seja `None`, o laço é interrompido com o comando `break`. Essa medida de segurança é essencial para evitar que o código continue tentando processar frames inexistentes, o que poderia resultar em erros ou até mesmo no travamento do sistema.

Código 5 – Aferição do valor da variável frame

```
1 class ContadorGarrafaAI:
2     def iniciar(self):
3         while True:
4             frame = self.camera.obter_frame()
5             if frame is None:
6                 break
```

Por outro lado, se o valor de `ret` for `True`, o código prossegue para processar o frame capturado, seguindo o fluxo normal da aplicação. Logo, a visualização do fluxograma da câmera pode ser vista na Figura 17

Figura 17 – Fluxograma da Câmera



Fonte: Autoria própria

2.5 PROCESSAMENTO DIGITAL DE IMAGEM

Ao instanciar o objeto `pdi` à classe `PDI`, são definidos como atributos variáveis relacionadas à limiarização e à região de interesse da imagem (ROI).

As variáveis utilizadas para limiarização, chamadas de `lower_bound` e `upper_bound`, são inicializadas com arrays do tipo `numpy`.

Além disso, as variáveis relacionadas à região de interesse, denominadas `x`, `y`, `w` e `h`, são definidas com coordenadas que delimitam uma área específica dentro do frame capturado. Essas variáveis representam, respectivamente, as posições horizontal (`x`) e vertical (`y`), bem como a largura (`w`) e altura (`h`) da região de interesse.

Vale resaltar que os valores definidos para as variáveis `lower_bound` e `upper_bound`, utilizadas na limiarização da imagem, bem como as coordenadas `x`, `y`, `w` e `h`, que delimitam a ROI, foram obtidos **empiricamente**. O processo envolveu ajustes iterativos e experimentação prática para identificar os parâmetros que melhor se adequam às necessidades do processamento de imagem, garantindo uma segmentação e definição eficiente da área de interesse. Adicionalmente, é importante destacar que o ambiente de trabalho **não conta com controle de iluminação**. Esse fator foi considerado durante o processo empírico de ajuste dos parâmetros.

Código 6 – Atributos da classe PDI

```
class PDI:
    def __init__(self):
        self.lower_bound = np.array([0, 94, 115])
        self.upper_bound = np.array([255, 255, 255])
        self.x, self.y, self.w, self.h = 208, 142, 46, 332
```

2.5.1 Conversão de espaço de cores

Após confirmar o funcionamento adequado da câmera, cada frame capturado, inicialmente no formato padrão de cores BGR, é convertido para o espaço de cores HSV.

Essa conversão é realizada utilizando o método `aplicar_mascara()` do objeto `pdi`, passando como argumento a variável `frame`.

Código 7 – Inicialização do processamento digital de imagem

```
class ContadorGarrafaAI:
    def iniciar(self):
        while True:
            frame = self.camera.obter_frame()
            if frame is None:
                break

            # Aplicar PDI
            mascara = self.pdi.aplicar_mascara(frame)
```

Dentro desse método, é criado uma variável chamada `hsv` que recebe a função `cvtColor()` da biblioteca OpenCV, que permite transformar o frame de um espaço de cores para outro.

Código 8 – Transformação do modelo de cor BGR para HSV

```
class PDI:
    def aplicar_mascara(self, frame):
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

Para realizar essa conversão, a função `cvtColor()` recebe dois parâmetros principais:

- O `frame`, que contém a imagem capturada no formato BGR.
- O código de conversão de cores `cv2.COLOR_BGR2HSV`, que instrui a função a transformar o espaço de cores do formato BGR para HSV (observando que a OpenCV utiliza BGR como padrão em vez de RGB).

2.5.2 Limiarização

Após a conversão do frame para o espaço de cores HSV, o processo de limiarização é realizado ainda dentro do método `aplicar_mascara()`. Nesse contexto, uma variável chamada `mascara` é inicializada para armazenar o resultado de uma máscara binária gerada com base nos limites inferior (`lower_bound`) e superior (`upper_bound`) definidos previamente.

Código 9 – Geração da máscara binária

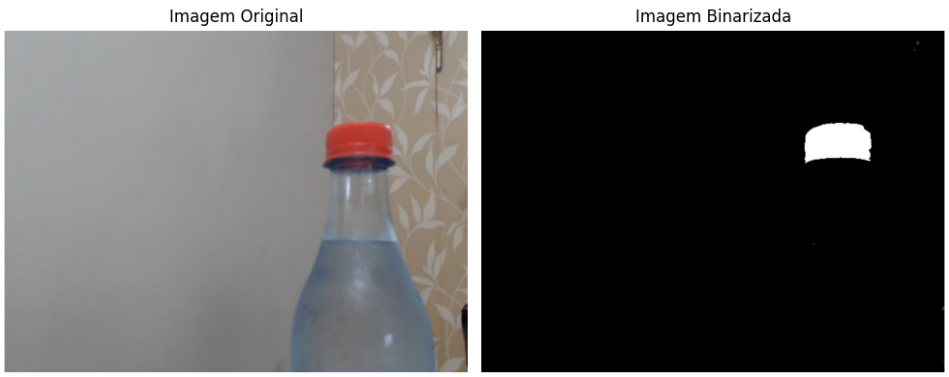
```
1 class PDI:
2     def aplicar_mascara(self, frame):
3         hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
4         mascara = cv2.inRange(hsv, self.lower_bound, self.upper_bound
              )
5         return mascara
```

A máscara é criada utilizando a função `inRange()` da biblioteca OpenCV. Essa função avalia cada pixel da imagem convertida para HSV e determina se ele pertence à faixa de cores especificada. O comportamento da função é descrito a seguir:

- Pixels cujos valores se encontram dentro do intervalo definido são classificados como pertencentes ao objeto de interesse e recebem o valor 255 (branco) na máscara.
- Pixels fora desse intervalo são considerados como não pertencentes ao objeto de interesse e recebem o valor 0 (preto).

Por fim, o método `aplicar_mascara()` retorna a variável `mascara`, que contém a máscara binária resultante, conforme a Figura 18, pronta para ser utilizada nas etapas seguintes do pipeline de processamento digital de imagem.

Figura 18 – Resultado da aplicação dos limiares para binarização do frame



Fonte: Autoria própria

2.5.3 Definição da região de interesse

Após a criação da máscara binária, o próximo passo no pipeline de processamento é determinar a quantidade de pixels brancos dentro de uma região de interesse, que é cuidadosamente definida para focar apenas na área do frame onde os objetos de interesse aparecem, neste caso, as tampas das garrafas.

Nessa lógica, cria-se uma variável chamada de `brancos` que recebe do objeto `pdi` o método `contar_brancos`, passando a máscara binária como argumento.

Código 10 – Inicialização da contagem de brancos

```
class ContadorGarrafaAI:
    def iniciar(self):
        while True:
            frame = self.camera.obter_frame()
            if frame is None:
                break

            # Aplicar PDI
            mascara = self.pdi.aplicar_mascara(frame)
            brancos = self.pdi.contar_brancos(mascara)
```

Este método utiliza as coordenadas previamente definidas na classe `PDI` para delimitar a ROI. As coordenadas (`x`, `y`) e dimensões (`w`, `h`) da ROI foram configuradas como atributos da classe e especificam precisamente a posição e o tamanho do retângulo dentro do frame onde as garrafas tampadas passarão.

Dentro do método `contar_brancos()`, é criada uma variável chamada `recorte`, que isola a subseção correspondente à ROI da máscara binária, onde esse recorte é obtido utilizando a sintaxe de fatiamento do NumPy.

O `recorte` contém apenas os pixels da máscara que estão dentro da ROI, eliminando informações desnecessárias de outras áreas do frame.

Código 11 – Recorte da ROI

```
class PDI:
    def contar_brancos(self, mascara):
        recorte = mascara[self.y:self.y+self.h, self.x:self.x+self.w]
```

2.5.4 Contagem de brancos

Dentro do método `contar_brancos()`, após a definição e extração da ROI do frame binarizado, procede-se à contagem de pixels brancos na imagem processada. A contagem dos pixels brancos é realizada utilizando a função `countNonZero()` da biblioteca OpenCV.

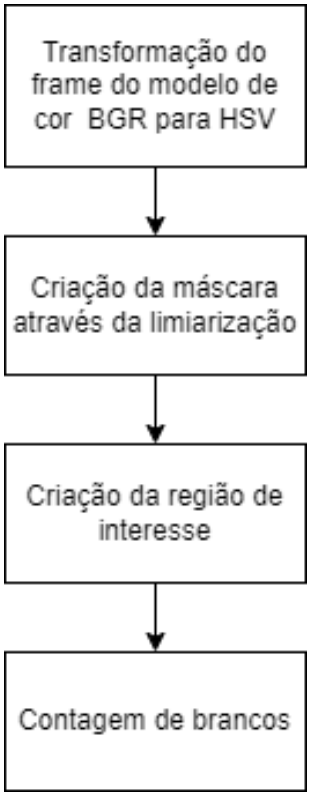
Esta função é especialmente projetada para analisar imagens binárias, contabilizando todos os pixels que possuem valor diferente de zero, ou seja, os pixels brancos. No contexto do método, a função `countNonZero()` é chamada com o recorte da ROI como argumento, armazenado na variavel `brancos`.

Código 12 – Contagem de brancos

```
1 class PDI:
2     def contar_brancos(self, mascara):
3         recorte = mascara[self.y:self.y+self.h, self.x:self.x+self.w]
4         brancos = cv2.countNonZero(recorte)
5         return brancos
```

Logo, o método `contar_brancos()` retorna o valor armazenado em `brancos`, que será utilizado em etapas subsequentes para a lógica de contagem e identificação de objetos no sistema. O fluxograma do processamento digital de imagem está visível na Figura 19

Figura 19 – Fluxograma do PDI



Fonte: Autoria própria

2.6 ESTRUTURA DE CONTAGEM

Para realizar a contagem das garrafas, o projeto utiliza o objeto `contador`, que é uma instância da classe `Contador()`. Esta classe foi projetada para gerenciar a lógica de contagem e possui dois atributos principais:

- **`contador`**: uma variável que armazena o número total de garrafas detectadas, inicialmente definida como `0`.
- **`liberado`**: uma variável de controle que evita múltiplas contagens da mesma garrafa durante a análise, inicialmente definida como `True`.

Código 13 – Atributos da classe Contador

```
1 class Contador:
2     def __init__(self):
3         self.contador = 0
4         self.liberado = True
```

2.6.1 Contador de garrafas

Com o número de pixels brancos calculado na ROI, uma nova variável chamada `contagem` é criada, a qual recebe o resultado do método `atualizar_contagem()` do objeto `contador`.

Código 14 – Contagem das garrafas

```
1  class ContadorGarrafaAI:
2      def iniciar(self):
3          while True:
4              frame = self.camera.obter_frame()
5              if frame is None:
6                  break
7
8              # Aplicar PDI
9              mascara = self.pdi.aplicar_mascara(frame)
10             brancos = self.pdi.contar_brancos(mascara)
11
12             # Atualizar contagem
13             contagem = self.contador.atualizar_contagem(brancos)
```

Esse método implementa a lógica principal de contagem de garrafas tampadas, assegurando a precisão do sistema.

A lógica funciona da seguinte forma: o contador de garrafas é incrementado apenas quando o número de pixels brancos na ROI excede um limiar predefinido de 1500 pixels **e** a variável de controle `liberado` está definida como `True`. O limiar de 1500 pixels foi cuidadosamente ajustado para garantir que apenas garrafas tampadas sejam contabilizadas, evitando a contagem incorreta de fragmentos, ruídos ou outros elementos presentes no frame.

A variável de controle `liberado` desempenha um papel fundamental nesse processo. Inicialmente definida como `True`, ela permite a detecção e contagem de uma nova garrafa. Assim que uma garrafa satisfaz as condições de detecção, o contador é incrementado em +1, e `liberado` é imediatamente alterado para `False`. Isso bloqueia novas contagens enquanto a garrafa atual ainda está dentro da ROI, prevenindo contagens duplicadas.

Quando o número de pixels brancos na ROI cai abaixo de 1500, indicando que a garrafa anterior já passou completamente pela área monitorada, a variável `liberado` é resetada para `True`. Isso sinaliza que o sistema está pronto para detectar e contabilizar uma nova garrafa.

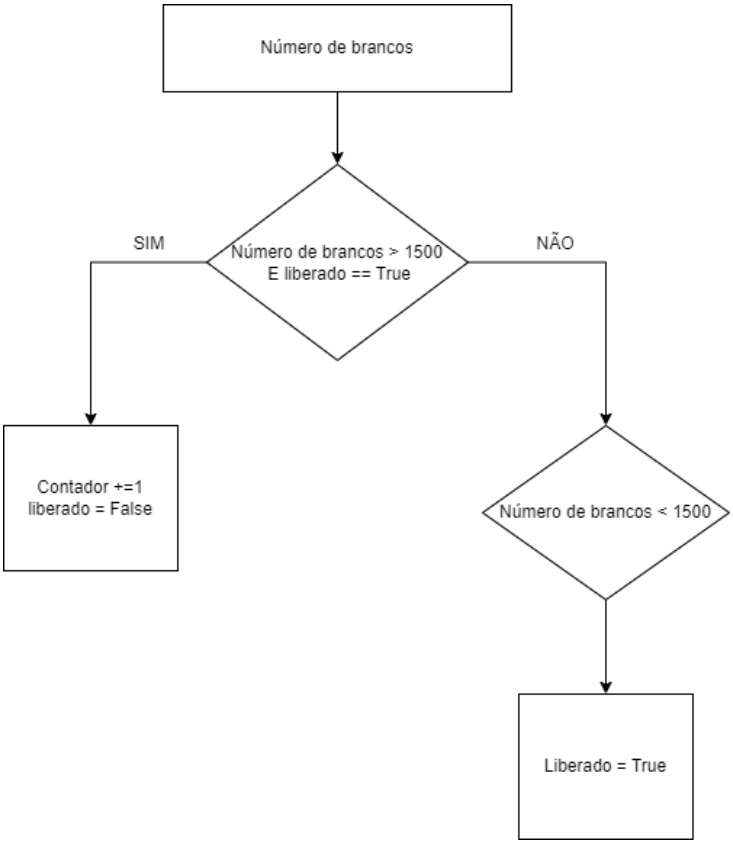
Por fim, o método retornará o valor da variável `contador` que será utilizado para a renderização de informação no frame.

Código 15 – Método para atualizar a contagem das garrafas

```
1 class Contador:
2     def atualizar_contagem(self, brancos):
3         if brancos > 1500 and self.liberado:
4             self.contador += 1
5             self.liberado = False
6         elif brancos < 1500:
7             self.liberado = True
8         return self.contador
```

O fluxograma do contador está visível na Figura 20

Figura 20 – Fluxograma do Contador



Fonte: Autoria própria

2.7 RENDERIZAÇÂO DE INFORMAÇÃO NO FRAME

Para apresentar ao usuário informações como tempo atual, contagem de garrafas e o estado de detecção (presença de uma garrafa tampada), utiliza-se o objeto `desenhos`, que é uma instância da classe `Desenhos`.

Código 16 – Renderização das informações

```
class ContadorGarrafaAI:
    def iniciar(self):
        while True:
            frame = self.camera.obter_frame()
            if frame is None:
                break

            # Aplicar PDI
            mascara = self.pdi.aplicar_mascara(frame)
            brancos = self.pdi.contar_brancos(mascara)

            # Atualizar contagem
            contagem = self.contador.atualizar_contagem(brancos)
```

```
14
15              # Desenhar no frame
16              self.desenhos.desenhar_retangulo(frame, self.contador.
                    liberado)
17              self.desenhos.desenhar_informacoes(frame, brancos,
                    contagem)
18
19              # Exibir video
20              cv2.imshow("Video Principal", frame)
21
22
23          self.camera.liberar_camera()
24          cv2.destroyAllWindows()
```

Essa classe recebe como argumentos os valores da ROI e, com base neles, define os atributos `x`, `y`, `w` e `h`, responsáveis por delimitar a área de processamento no frame.

Código 17 – Atributos da classe Desenhos

```
1  class Desenhos:
2      def __init__(self, x, y, w, h):
3          self.x = x
4          self.y = y
5          self.w = w
6          self.h = h
```

O processo de exibição é realizado em duas etapas principais. Primeiro, o método `desenhar_retangulo()` é chamado, recebendo como argumentos o frame atual e o estado da variável de controle `liberado`. Este método desenha um retângulo ao redor da ROI no frame processado. Quando não há detecção de uma garrafa tampada (`liberado = True`), o retângulo é exibido na cor rosa. No entanto, se uma garrafa tampada for detectada (`liberado = False`), o retângulo muda para a cor verde, indicando a detecção ao usuário.

Código 18 – Desenho de um retângulo ao redor da ROI

```
1  class Desenhos:
2      def desenhar_retangulo(self, frame, liberado):
3          if not liberado:
4              cv2.rectangle(frame, (self.x, self.y), (self.x + self.w,
                   self.y + self.h), (0, 255, 0), 4)
5          else:
6              cv2.rectangle(frame, (self.x, self.y), (self.x + self.w,
                   self.y + self.h), (255, 0, 255), 4)
```

Em seguida, o método `desenhar_informacoes()` é executado, recebendo o frame, o número de pixels brancos (`brancos`) e a contagem de garrafas (`contagem`) como argumentos. Nesse método, uma variável chamada `horario_atual` é criada e utilizando o módulo `datetime`,onde é registrado o horário atual no formato de horas, minutos e segundos. Posteriormente, as informações do horário, da contagem de garrafas e do número de pixels brancos são sobrepostas no frame por meio da função `putText()` da biblioteca OpenCV, garantindo que o usuário visualize essas informações em tempo real no frame processado.

Código 19 – Desenho da contagem e horário

```
class Desenhos:
    def desenhar_informacoes(self, frame, brancos, contador):
        horario_atual = datetime.now().strftime("%H:%M:%S")
        cv2.putText(frame, str(brancos),
                    (self.x - 30, self.y - 50),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    1, (255, 255, 255), 1)
        cv2.putText(frame, f"Horario: {horario_atual}",
                    (self.x + 100, self.y + 120),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    1, (0, 255, 255), 2)
        cv2.putText(frame, str(contador),
                    (self.x + 100, self.y),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    3, (255, 0, 0), 5)
```

Após o processamento, o frame atualizado é exibido em uma janela utilizando o método `imshow()` do OpenCV. Para interromper a execução do código, o sistema verifica continuamente se a tecla `q` foi pressionada. Caso positivo, o laço principal é encerrado e o método `liberar_camera()` do objeto `camera` é chamado, liberando os recursos associados à câmera. Além disso, o método `cv2.destroyAllWindows()` é executado para fechar todas as janelas abertas pelo OpenCV, garantindo a liberação completa dos recursos de interface gráfica e evitando janelas residuais após a finalização do programa.

3 RESULTADOS E DISCUSSÕES

3.1 RESPOSTA DE REFERÊNCIAS

No contexto da visão computacional, os dados de *ground truth* incluem um conjunto de imagens e um conjunto de rótulos nessas imagens, definindo um modelo para o reconhecimento de objetos, incluindo a contagem, localização e relações de características principais. Os rótulos são adicionados por um humano ou automaticamente por meio de análise de imagem, dependendo da complexidade do problema. (KRIG, 2014)

Neste projeto, a resposta de referências foi realizada através da validação do sistema utilizando o *ground truth*, definido manualmente a partir de três amostras distintas. Durante o período de coleta das amostras, cada garrafa que passou pela linha de produção foi contada manualmente, criando assim um conjunto de dados de referência confiável para posterior comparação com os resultados gerados pelo sistema de visão computacional.

Vale resaltar que o sistema foi desenvolvido apenas para detectar garrafas tampadas, ou seja, caso passe garrafas destampadas, o sistema não terá um contador para as mesmas.

3.1.1 Amostra 01

A primeira amostra consiste em um vídeo de 29 segundos com um total de amostra de 78 garrafas contadas manualmente, sendo 78 garrafas tampadas e 0 garrafas destampadas. Ademais, após o processamento realizado pelo sistema de visão, o mesmo conseguiu contabilizar todas as garrafas tampadas.Os resultados foram inseridos na tabela abaixo.

Tabela 5 – Resultados do Ground Truths para amostra 01

	Garrafas Tampadas
Real	78
Nvidia Jetson	78

Fonte: Autoria Própria

3.1.2 Amostra 02

Para a segunda amostra, a gravação continha 62 segundos, com 167 garrafas contadas manualmente. Dentre essas amostras, 165 foram garrafas tampadas e 2 foram garrafas destampadas. Nessa lógica, o sistema conseguiu contabilizar todas as garrafas tampadas e não contabilizou nenhuma garrafa destampada.

Tabela 6 – Resultados do Ground Truths para amostra 03

	Garrafas Tampadas
Real	165
Nvidia Jetson	165

Fonte: Autoria Própria

3.1.3 Amostra 03

A terceira amostra mostrou um comportamento diferente relacionado à contagem das garrafas em relação as amostras anteriores. Em um vídeo de 99 segundos, foram contadas 270 garrafas manualmente, onde 266 eram garrafas tampadas e 4 garrafas destampadas.

Após o processamento do sistema de visão, foram contabilizado apenas 264 garrafas, contendo uma diferença de duas em relação à contagem manual. Em relação as garrafas destampadas, o sistema não contabilizou nenhuma.

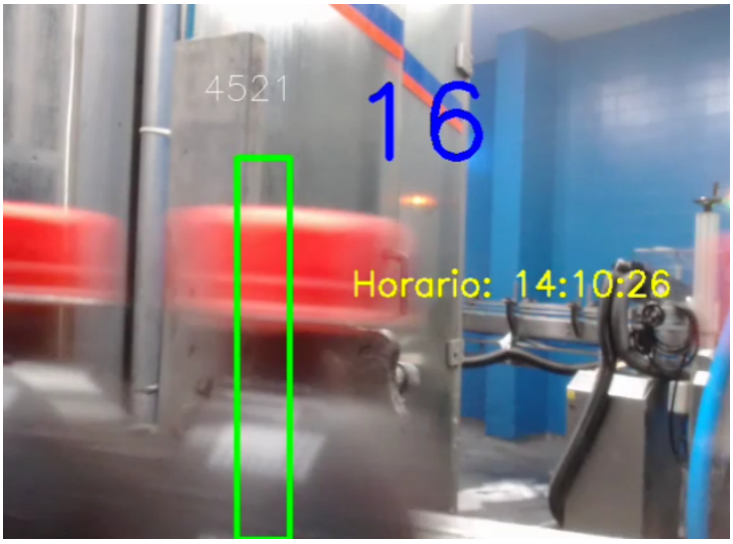
Tabela 7 – Resultados do Ground Truths para amostra 03

	Garrafas Tampadas
Real	266
Nvidia Jetson	264

Fonte: Autoria Própria

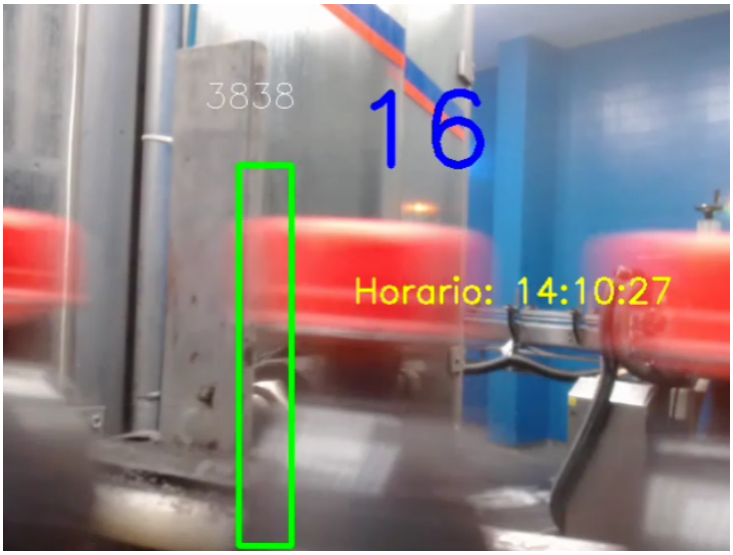
Na contagem das garrafas número 16 e número 89, observou-se que o sistema não contabilizou corretamente essas garrafas, resultando em falsos negativos, conforme as Figuras 21, 22, 23, 24. Esse erro ocorreu porque, na região de interesse, o número de pixels brancos não diminuiu o suficiente para ficar abaixo do limiar estabelecido (1500). Como consequência, o sistema não registrou essas garrafas, uma vez que o critério de detecção não foi atendido, impactando na contagem.

Figura 21 – Primeira contagem errada do sistema no segundo 26



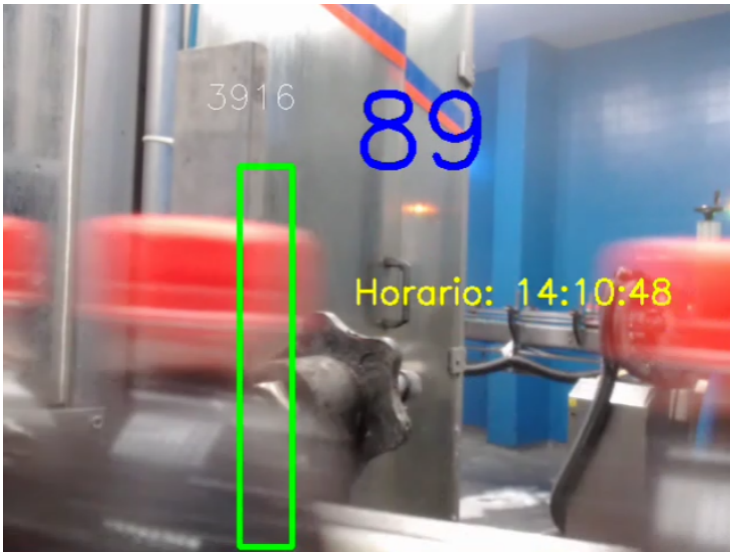
Fonte: Autoria Própria

Figura 22 – Primeira contagem errada do sistema no segundo 27



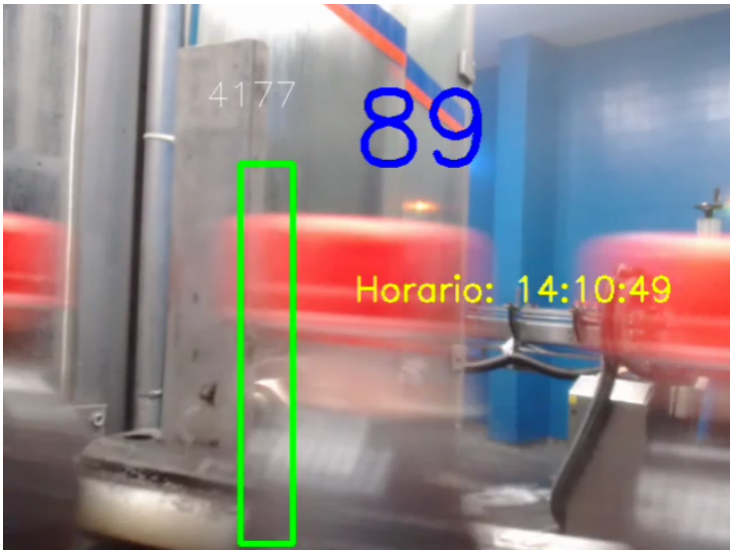
Fonte: Autoria Própria

Figura 23 – Segunda contagem errada do sistema no segundo 48



Fonte: Autoria Própria

Figura 24 – Segunda contagem errada do sistema no segundo 49



Fonte: Autoria Própria

3.2 MÉTRICAS

Após obter os resultados do Ground Truth, é possível realizar uma análise mais profunda do desempenho do sistema de visão computacional implementado no projeto. Essas análises são feitas por meio de diversas métricas, como por exemplo, precisão, *recall*, f1-score, que permitem avaliar a eficácia e a precisão do sistema em detectar corretamente as garrafas tampadas na linha de produção.

3.2.1 Matriz de confusão

A matriz de confusão de uma hipótese h oferece uma medida efetiva do modelo de classificação, ao mostrar o número de classificações corretas versus as classificações preditas para cada classe, sobre um conjunto de exemplos T. O número de acertos, para cada classe, se localiza na diagonal principal da matriz. Os demais elementos representam erros na classificação. A matriz de confusão de um classificador ideal possui todos esses elementos iguais a zero, uma vez que ele não comete erros. (MONARD; BARANAUSKAS, 2003)

Por simplicidade, considere um problema de duas classes. Com apenas duas classes, usualmente rotuladas como "+" (positivo) e "" (negativo), as escolhas estão estruturadas para predizer a ocorrência ou não de um evento simples. Neste caso, os dois erros possíveis são denominados falso positivo (FP) e falso negativo (FN). (MONARD; BARANAUSKAS, 2003)

No contexto do projeto, temos a matriz de confusão descrita da seguinte forma:

- **Verdadeiro Positivo (VP)**: São as garrafas que foram detectadas pelo sistema de visão computacional e que estavam realmente presentes na linha de produção, conforme indicado pelo ground truth. Representam os casos em que o sistema acertou ao identificar corretamente uma garrafa.
- **Verdadeiro Negativo (VN)**: Refere-se aos momentos em que o sistema não detectou nenhuma garrafa e, de acordo com o ground truth, realmente não havia nenhuma garrafa presente na linha de produção.
- **Falso Positivo (FP)**: São os casos em que o sistema de visão computacional indicou a presença de uma garrafa, porém, de acordo com o ground truth, não havia nenhuma garrafa naquele momento.
- **Falso Negativo (FN)**: Representa as situações em que o sistema de visão computacional não detectou uma garrafa que, de acordo com o ground truth, estava presente na linha de produção. Esse tipo de erro é crítico, pois significa que o sistema não conseguiu contar uma garrafa que deveria ter sido registrada.

Tabela 8 – Matriz de confusão

		Detectada	
		Sim	**Não**
Real	**Sim**	VP = 506	FN = 2
	Não	FP = 0	VN = 6

Fonte: Autoria Própria

3.2.2 Precisão

A precisão é a taxa com que todos os exemplos classificados como positivos são realmente positivos. Nenhum exemplo negativo é incluído. (MATOS et al., 2009)

Sendo assim, a precisão é uma métrica para avaliar a qualidade das previsões positivas, uma vez que ela indica o quão confiáveis são as detecções realizadas pelo modelo, medindo a proporção de acertos entre todas as vezes que o sistema indicou a presença de uma garrafa. Sua formula é dada por:

$$\text{Precisão} = \frac{\text{VP}}{\text{VP} + \text{FP}} \tag{10}$$

Logo, podemos utilizar os valores encontrados na matriz de confusão e encontrar a precisão para o cenário específicado.

$$\text{Precisão} = \frac{\text{VP}}{\text{VP} + \text{FP}} = \frac{506}{506 + 0} = 1 \tag{11}$$

Sendo assim, neste cenário, o sistema alcançou uma precisão de 100%, indicando que todas as previsões feitas pelo sistema para a presença de garrafas estavam corretas. Nenhum caso de Falsos Positivos foi registrado, o que é um excelente resultado, pois significa que o sistema não indicou a presença de garrafas onde não havia.

3.2.3 Revocação

A revocação (*recall*) é uma taxa que classifica como positivos todos os exemplos que são positivos. Nenhum exemplo positivo é deixado de fora. Apresenta uma indicação do quanto do total de informação relevante foi recuperada. (MATOS et al., 2009)

Ademais, a revocação, também chamado de sensibilidade, é outra métrica para avaliar o desempenho do sistema. No contexto do projeto, a revocação mede a capacidade do sistema de detectar corretamente todas as garrafas tampadas que estavam realmente presentes na linha de produção, ou seja, ele avalia o quão eficaz o sistema foi em não deixar de detectar uma garrafa. Sua fórmula é dado por:

$$\text{Recall} = \frac{\text{VP}}{\text{VP} + \text{FN}} \tag{12}$$

Substituindo os valores da fórmula do *recall* pelo da matriz de confusão, obtemos:

$$\text{Recall} = \frac{\text{VP}}{\text{VP} + \text{FN}} = \frac{506}{506 + 2} \approx 0,996 \tag{13}$$

Nessa lógica, o sistema detectou aproximadamente 99,6% de todas as garrafas tampadas que realmente estavam presentes na linha de produção.

3.2.4 Medida-F

A Medida-F é definida como a média harmônica ponderada da precisão e revocação, conforme apresentado na Equação 14. F_β mede a eficácia da recuperação em relação ao valor atribuído à Beta (β). (MATOS et al., 2009)

Pesos comumente utilizados para β são: F_2 (revocação é o dobro da precisão) e $F_{0,5}$ (precisão é o dobro da revocação). A precisão tem peso maior para valores $\beta < 1$, enquanto que $\beta > 1$ favorece a revocação. (MATOS et al., 2009)

$$F_\beta = \frac{(1+\beta^2) \times (P \times R)}{(\beta^2 \times P + R)}, \quad \text{onde} \quad \beta = \frac{1-\alpha}{\alpha} \tag{14}$$

A Medida-F foi derivada por van Rijsbergen (1979), baseada na medida de eficiência, conforme apresentado na Equação 15. (MATOS et al., 2009)

$$E = 1 - \left(\frac{1}{\left(\frac{\alpha}{P} + \frac{(1-\alpha)}{R} \right)} \right) \tag{15}$$

A relação entre a Medida-F_β e a medida de eficiência é: $F_\beta = 1 - E$. Quando a precisão e a revocação têm o mesmo peso ($\beta = 1$), a medida é conhecida como Medida-F_1, ou **F-Score** balanceada, conforme apresentado na Equação 16. (MATOS et al., 2009)

$$F = \frac{2 \times P \times R}{P + R} \tag{16}$$

No contexto do projeto, o F1 Score é encontrado a partir do cálculo:

$$F1 = 2 \times \frac{\text{Precisão} \times \text{Recall}}{\text{Precisão} + \text{Recall}} = 2 \times \frac{1 \times 0{,}996}{1 + 0{,}996} \approx 0,998 \tag{17}$$

Com o F1-Score de aproximadamente 99,8% indica que o sistema utilizado no projeto tem um desempenho muito bom, conseguindo equilibrar de forma eficiente tanto a precisão (evitando detecções erradas) quanto o *recall* (garantindo que quase todas as garrafas tampadas sejam detectadas).

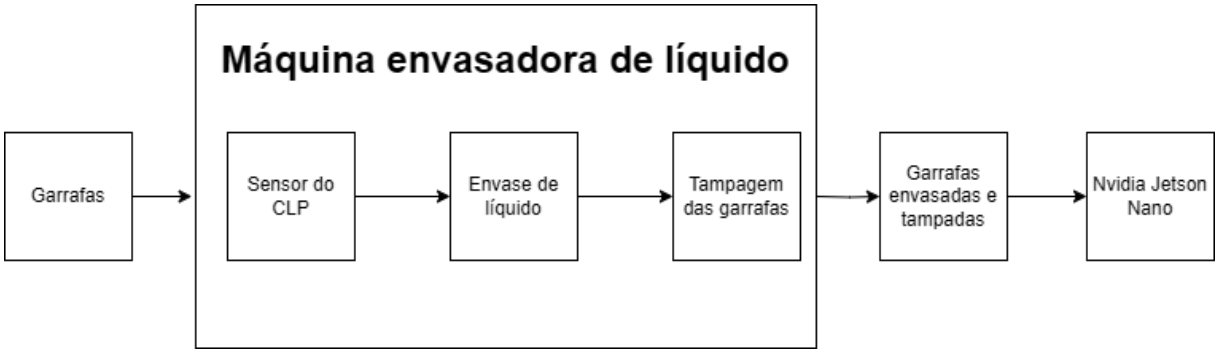
3.3 TESTES EM CAMPO

Após a conclusão das métricas do projeto, diversos testes foram conduzidos em ambiente de fábrica para validar a solução proposta.

Um ponto fundamental é a configuração dos sensores na linha de produção, que ajuda a entender a dinâmica da contagem de garrafas. O sensor conectado ao Controlador Lógico Programável (CLP) está posicionado dentro da máquina enchedora, antes do processo de

envase das garrafas, enquanto o sistema de detecção baseado na plataforma NVIDIA Jetson está instalado logo após a enchedora, conforme a Figura 25.

Figura 25 – Localização dos sistemas de contagem de garrafas



Fonte: Autoria própria

Essa disposição oferece uma visão mais completa do processo, mas também gera uma variabilidade nas contagens devido à distância física entre os dois pontos de medição. Para mitigar essa discrepância, as validações foram realizadas sempre que a máquina passava por uma parada programada ou não. Durante essas pausas, verificava-se se ainda havia garrafas dentro da máquina e caso não houvesse, era possível fazer uma comparação mais precisa entre a quantidade contada pelos dois sistemas.

Por fim, será criada uma tabela contendo as seguintes colunas: a primeira coluna registrará o horário da aferição; a segunda coluna indicará o valor presente na memória do CLP, uma vez que algumas validações ocorreram após o início do turno; a terceira coluna exibirá o valor real do CLP, desconsiderando o valor já marcado na memória do CLP durante a primeira validação; a quarta coluna apresentará o valor contabilizado pelo sistema de detecção baseado em visão computacional; a quinta coluna mostrará a diferença em número de garrafas entre o CLP e o sistema de detecção; e, por fim, a sexta coluna destacará essa diferença em termos percentuais.

3.3.1 Teste 01

A primeira validação do sistema ocorreu no dia 02 de abril de 2024. Às 14:52, após uma parada para a troca do líquido da enchedora, o valor inicial de 958 garrafas estava registrado na memória do CLP, conforme a Figura 26.

Figura 26 – Contagem inicial da máquina

Fonte: Autoria própria

Durante essa primeira validação, ocorreram 5 paradas, permitindo a comparação das contagens entre os dois sistemas.

Figura 27 – Implementação do sistema de contagem de garrafas tampadas

Fonte: Autoria própria

Às 15:41, uma troca de peças na máquina marcou o término da etapa de validação. Os resultados dessas comparações estão resumidos na tabela a seguir:

Tabela 9 – Resultados da Validação - 02 de Abril de 2024

Horário	CLP Memória	CLP Real	Nvidia Jetson	Diferença	Diferença (%)
14:52	958	0	0	0	0.0%
15:00	1368	410	359	51	12.4%
15:04	1740	782	719	63	8.7%
15:06	1967	1015	940	75	7.3%
15:15	2731	1773	1696	77	4.3%
15:41	4494	3536	3454	82	2.3%

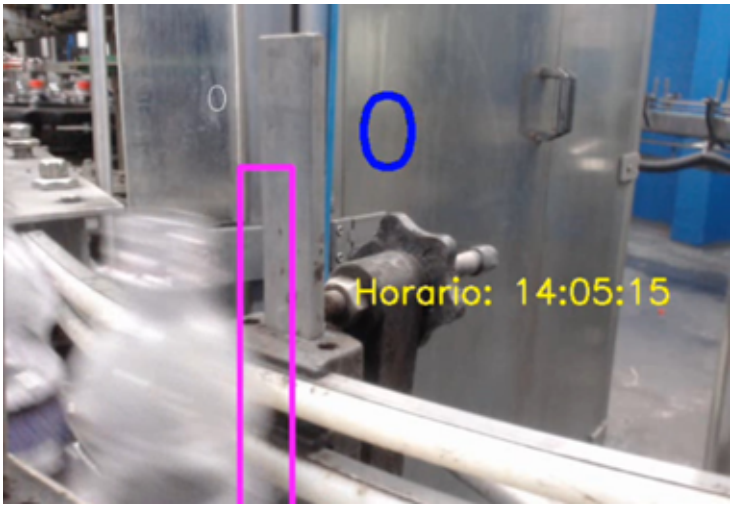
Fonte: Autoria Própria

Após essa primeira validação, foi observado que o sistema estava corretamente contabilizando as garrafas tampadas conforme programado. No entanto, percebeu-se que ele não conseguia registrar as garrafas tampadas que caíam durante o processo, resultando em uma pequena discrepância nas contagens. Esse ponto será abordado em futuras otimizações para garantir que todas as garrafas, incluindo aquelas que caem, sejam contabilizadas adequadamente.

3.3.2 Teste 02

A segunda validação ocorreu no dia 08 de abril de 2024, onde a posição de detecção das garrafas foi ajustada, conforme a Figura 28, com o objetivo de minimizar o problema das garrafas caídas relatados na validação 01.

Figura 28 – Sistema de detecção de garrafas com câmera ajustada



Fonte: Autoria própria

Durante essa validação, a coleta de dados começou antes da troca de turno, o que permitiu sincronizar o início da contagem da enchedora com o sistema de contagem de-

senvolvido. Ademais, é possível verificar que as garrafas que não estavam tampadas não foram contabilizadas pelo sistema, conforme a Figura 28. Logo, a máquina teve apenas duas paradas ao longo do processo, o que resultou em apenas duas validações.

Tabela 10 – Resultados da Validação - 08 de Abril de 2024

Horário	**CLP Memória**	**CLP Real**	**Nvidia Jetson**	**Diferença**	**Diferença (%)**
14:09	0	473	471	2	0.4%
14:13	0	875	865	10	1.2%

Fonte: Autoria Própria

3.3.3 Teste 03

A validação realizada no dia 09 de abril de 2024 seguiu os mesmos parâmetros de operação estabelecidos na validação 02. A câmera foi posicionada no mesmo local, mantendo a mesma perspectiva para capturar as garrafas logo após o processo de tampagem das garrafas envasadas. Além disso, o início da contagem foi sincronizado com o CLP, garantindo que ambas as contagens, tanto do sistema de visão computacional quanto do CLP, fossem iniciadas ao mesmo tempo.

Tabela 11 – Resultados da Validação - 09 de Abril de 2024

Horário	**CLP Memória**	**CLP Real**	**Nvidia Jetson**	**Diferença**	**Diferença (%)**
14:09	0	0	0	0	0.0%
14:50	5155	5155	5118	37	0.7%
15:23	9690	9690	9624	66	0.6%
16:01	11899	11899	11813	86	0.7%

Fonte: Autoria Própria

3.4 DISCURSSÕES DOS RESULTADOS OBTIDOS

Os testes realizados ao longo deste trabalho evidenciaram a eficiência e as limitações do sistema de contagem baseado em visão computacional comparado ao CLP. A análise dos dados coletados durante os três testes forneceu subsídios para discutir aspectos relevantes do desempenho da solução desenvolvida, incluindo precisão, desafios operacionais e potencial para melhoria.

3.4.1 Diferença de garrafas contadas

Os resultados obtidos nos testes mostram que o sistema proposto apresenta uma pequena diferença percentual em relação ao CLP variando entre 0,4% e 12,4%. No primeiro

teste, a maior discrepância foi observada no início do processo, com uma redução gradual ao longo das medições subsequentes. No terceiro teste, a diferença percentual foi consistentemente baixa, atingindo um valor máximo de 0,7%. Isso indica que, uma vez ajustado, o sistema é capaz de acompanhar o desempenho esperado em termos de contagem.

3.4.2 Problemas identificados

Os testes também revelaram limitações significativas no sistema proposto. No primeiro teste, por exemplo, foi constatado que garrafas que caíam durante o transporte não eram contabilizadas pelo sistema de visão computacional, o que resultava em discrepâncias consideráveis nas contagens. Esse problema foi parcialmente mitigado na validação subsequente, ao reposicionar a câmera para capturar as garrafas em um ângulo que minimizasse o impacto de tais ocorrências.

3.4.3 Impacto do posicionamento e da sincronização

Os ajustes no posicionamento da câmera, realizados entre as validações 01 e 02, demonstraram impacto positivo na precisão do sistema. O alinhamento correto da câmera com a esteira transportadora permitiu uma melhor captura das garrafas, reduzindo significativamente as discrepâncias entre o sistema proposto e o CLP. Além disso, a sincronização das contagens foi fundamental para garantir a comparabilidade dos dados. No primeiro teste, diferenças maiores foram registradas devido à falta de alinhamento inicial, enquanto nos testes subsequentes a sincronização mitigou esse problema.

3.4.4 Considerações sobre as diferenças percentuais

As diferenças percentuais registradas variaram de forma significativa entre os testes. No primeiro teste, a maior discrepância ocorreu logo após o início do turno, com 12,4%. Essa variação pode ser atribuída ao fato de que o sistema ainda estava em processo de ajuste e calibração. Por outro lado, nas validações 02 e 03, as diferenças foram menores, permanecendo abaixo de 1,2%, o que demonstra que o sistema foi melhor adaptado à operação da linha de produção.

Essa redução também pode ser explicada pelo aumento significativo no número de amostras de garrafas tampadas analisadas, aliado a um crescimento menor no número de garrafas destampadas. Esses resultados indicam que a precisão é amplamente influenciada pelas condições iniciais de configuração, pelo ambiente operacional e pela proporcionalidade das amostras avaliadas.

CONCLUSÃO

O desenvolvimento de uma solução de visão computacional para a contagem de garrafas tampadas em uma linha de produção fabril demonstrou ser uma alternativa viável e eficiente para automatizar um processo anteriormente suscetível a erros humanos e ineficiências de sistemas tradicionais. Com o uso de técnicas de processamento digital de imagens, o sistema foi capaz de alcançar uma precisão elevada na contagem de garrafas, reduzindo significativamente o índice de erros e aumentando a confiabilidade das operações.

A implementação da plataforma NVIDIA Jetson Nano foi essencial para o sucesso do projeto. Mesmo sem o uso de aceleração por GPU, o dispositivo mostrou-se adequado ao executar as tarefas de captura, processamento e contagem em tempo real, dentro das limitações impostas pelo ambiente fabril. A combinação da Jetson Nano com a webcam Logitech C920s e o ajuste na região de interesse permitiram uma contagem precisa das garrafas tampadas, especialmente em um ambiente com iluminação não controlada.

Os resultados das validações em campo, comparados com o ground truth manual, confirmaram a eficácia do sistema. A solução apresentou uma precisão de 100% e uma taxa de *recall* de 99.6%, garantindo que a maioria das garrafas tampadas fossem corretamente detectadas e contadas. Com relação ao F1-Score, obteve-se o valor de 99,8%, o que evidencia um desempenho confiável e robusto para o contexto de contagem de objetos em linha de produção.

Alguns desafios foram enfrentados ao longo do projeto, como a contagem de garrafas tampadas caídas. Contudo, ajustes no posicionamento da câmera e nos parâmetros de contagem reduziram consideravelmente esses problemas, o que destaca a flexibilidade da visão computacional em se adaptar a diferentes cenários produtivos. Além disso, foi identificada a necessidade de melhorias na segmentação de objetos para evitar falsos negativos em casos onde o objeto não preenchia suficientemente o limite de detecção.

Para trabalhos futuros, recomenda-se explorar melhorias que ampliem a eficiência e a aplicabilidade do sistema de contagem de garrafas. A implementação em hardware mais simples, como microcontroladores de baixo custo, poderia reduzir custos e simplificar a instalação em ambientes fabris de menor complexidade. Além disso, a adição de funcionalidades para detecção de rótulos, contagem de garrafas destampadas e monitoramento do nível de líquido nas garrafas permitiria uma análise mais abrangente da qualidade do produto, identificando garrafas com rótulos ausentes ou aplicados incorretamente e monitorando o nível de enchimento. A integração de um banco de dados para armazenar os dados coletados em tempo real também enriqueceria o sistema, permitindo análises históricas que auxiliariam na otimização do processo e servindo como base para um módulo de monitoramento remoto. Com essa infraestrutura, gestores poderiam acom-

panhar e ajustar a produção conforme necessário, o que elevaria a eficiência operacional e tornaria o sistema uma ferramenta versátil e acessível para empresas de diferentes portes e setores.

REFERÊNCIAS

ALENCAR, M. ***Detailing Sampling and Quantization***. 2012. Acesso em: 15 de março de 2024. Disponível em: ⟨https://maalencar.wordpress.com/2012/03/15/detailing-sampling-and-quantization/⟩.

ARDILA, L. S. C. ***Sistema de caracterización morfológica de racimos de banano en la zona del Urabá Antioqueno mediante procesamiento digital de imágenes***. 2021. Acesso em: 11 de junho de 2024. Disponível em: ⟨https://repository.eia.edu.co/server/api/core/bitstreams/24f50840-0f47-446a-9bf2-7b52489d7692/content⟩.

ARTERO, A. O. ***Inteligência artificial: teórica e prática***. 1. ed. São Paulo: Editora Livraria de Física, 2009.

BACKES, A. R.; JUNIOR, J. J. M. S. ***Introdução à visão computacional usando MATLAB®***. 1. ed. Rio de Janeiro: Alta Books, 2016.

BARBA-GUAMÁN, L.; NARANJO, J.; ORTIZ, A. Deep learning framework for vehicle and pedestrian detection in rural roads on an embedded gpu. ***Electronics***, MDPI, v. 9, n. 4, p. 589, 2020. Acesso em: 11 de junho de 2024. Disponível em: ⟨https://www.mdpi.com/2079-9292/9/4/589⟩.

BORGES, L. E. ***Python para desenvolvedores: aborda Python 3.3***. [S.l.]: Novatec Editora, 2014.

COELHO, F. C. ***Computação Científica com Python***. [S.l.]: Lulu. com, 2007.

GONZALEZ, R. C. ***Processamento digital de imagens***. 3. ed. São Paulo: Pearson Prentice Hall, 2010.

GRANADO. ***Monitor de tela LCD com visão completa x HD HDMI compatível com Raspberry Pi***. 2024. Acesso em: 2024-11-16. Disponível em: ⟨https://bra.grandado.com/products/monitor-de-tela-lcd-com-visao-completa-x-hd-hdmi-compativel-com-raspberry-pi?variant=UHJvZHVjdFZhcmlhbnQ6MzUzNDg2MDEw⟩.

HUANG, J.-H. ***Huang's Law and the Future of AI Chips***. 2023. Acesso em: 18 maio 2024. Disponível em: ⟨https://blogs.nvidia.com/blog/huangs-law-dally-hot-chips/⟩.

IDRIS, I. ***NumPy: Beginner's Guide***. [S.l.]: Packt Publishing Ltd, 2015.

IMOBILIS. ***Segmentação de Instâncias***. 2020. Acesso em: 10 de junhi de 2024. Disponível em: ⟨https://www2.decom.ufop.br/imobilis/segmentacao-instancias/⟩.

KRIG, S. Ground truth and annotations. In: ***Computer Vision Metrics: Survey, Taxonomy, and Analysis***. Springer, 2014. p. 91–115. Acesso em: 2024-11-17. Disponível em: ⟨https://link.springer.com/chapter/10.1007/978-1-4302-5930-5_7⟩.

LOGITECH. ***C920s Pro HD Webcam***. 2024. Acesso em: 2024-11-16. Disponível em: ⟨https://www.logitech.com/pt-br/products/webcams/c920s-pro-hd-webcam.960-001257.html?srsltid=AfmBOoo22xXJ4XvWzH-ZmmkduoULP-stuCtvg8GfeJHfwxsw9QWZT1IK⟩.

LUQUE, L. ***Aula 1 – Problema: Filtragem de Imagens***. 2016. Acesso em: 11 de junho de 2024. Disponível em: ⟨https://treinamentomaratona.wordpress.com/2016/06/08/aula-1-problema-filtragem-de-imagens/⟩.

MARENGONI, M.; STRINGHINI, S. Tutorial: Introdução à visão computacional usando opencv. ***Revista de Informática Teórica e Aplicada***, v. 16, n. 1, p. 125–160, 2009.

MATOS, P. F. et al. Relatório técnico "métricas de avaliaçao". ***Universidade Federal de Sao Carlos***, 2009.

MERCADO LIVRE. ***Suawin Tripé Câmera Universal 1,8m Canon Suporte Celular Tripod Preto***. 2024. ⟨https://www.mercadolivre.com.br/suawin-tripe-cmera-universal-18m-canon-suporte-celular-tripod-preto/p/MLB38167517#polycard_client=search-nordic&wid=MLB3788294395&sid=search&searchVariation=MLB38167517&position=7&search_layout=grid&type=product&tracking_id=1e227e85-2741-4907-99bd-c7c7be29efdd⟩. Acessado em 19 de novembro de 2024.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. ***Sistemas inteligentes-Fundamentos e aplicações***, v. 1, n. 1, p. 32, 2003.

NVIDIA DEVELOPER. ***Jetson Nano Developer Kit Carrier Board P3449 B01 Specification***. 2024. Acesso em: 2024-11-16. Disponível em: ⟨https://developer.nvidia.com/jetson-nano-developer-kit-carrier-board-p3449-b01-specification⟩.

Prime Control. ***Visão Computacional: Conceito e Aplicações***. 2023. Acesso em: 18 de maio de 2024. Disponível em: ⟨https://blog.primecontrol.com.br/case/visao-computacional-conceito-e-aplicacoes⟩.

PÉREZ, M. A. A. Espacios de color rgb, hsi y sus generalizaciones a n-dimensiones. ***INAOE, Tonantzintla***, 2009.

SILVA, J. S. M. A. da. Como análise sensorial contribui para o controle de qualidade na indústria de refrigerante. ***Mostra de Inovação e Tecnologia São Lucas (2763-5953)***, v. 4, n. 1, 2023.

SINGH, M.; INDU, S. Bgr to hsv based text extraction from manuscripts using slidebars. In: IEEE. ***2021 Asian Conference on Innovation in Technology (ASIANCON)***. [S.l.], 2021. p. 1–6.

STIVANELLO, M. E. Inspeção industrial através de visão computacional. ***Monografia (Monografia)—Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau***, p. 33, 2004.

WEG. ***Vantagens do Uso de Visão Computacional na Indústria 4.0***. 2024. Acesso em: 18 de maio de 2024. Disponível em: ⟨https://www.weg.net/digital/blog/vantagens-do-uso-de-visao-computacional-na-industria-4-0/⟩.

WIKIPÉDIA. ***HSV***. 2023. Acesso em: 25 de maio de 2024. Disponível em: ⟨https://pt.wikipedia.org/wiki/HSV⟩.

WIKIPÉDIA. ***Limiarização por Equilíbrio do Histograma***. 2024. Acesso em: 10 de junho de 2024. Disponível em: ⟨https://pt.wikipedia.org/wiki/Limiariza{\c{c}}{\~a}o_por_equil{\'\i}bri⟩.

ANEXO 01

Código 20 – Código Python - main.py

```python
from wutils import ContadorGarrafaAI

def main():
    algoritmo = ContadorGarrafaAI()
    algoritmo.iniciar()

if __name__ == "__main__":
    main()
```

ANEXO 02

Código 21 – Código Python - wutils.py

```python
import cv2
import numpy as np
from datetime import datetime


class Camera:
    def __init__(self):
        self.video = cv2.VideoCapture(1)

    def obter_frame(self):
        ret, frame = self.video.read()
        if not ret:
            print("Erro ao ler o quadro do v deo. Finalizando o loop.")
            return None
        return frame

    def liberar_camera(self):
        self.video.release()


class PDI:
    def __init__(self):
        self.lower_bound = np.array([0, 94, 115])
        self.upper_bound = np.array([255, 255, 255])
        self.x, self.y, self.w, self.h = 208, 142, 46, 332

    def aplicar_mascara(self, frame):
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        mascara = cv2.inRange(hsv, self.lower_bound, self.upper_bound)
        return mascara

    def contar_brancos(self, mascara):
        recorte = mascara[self.y:self.y+self.h, self.x:self.x+self.w]
        brancos = cv2.countNonZero(recorte)
        return brancos

```

```
37
38 class Contador:
39     def __init__(self):
40         self.contador = 0
41         self.liberado = True
42
43     def atualizar_contagem(self, brancos):
44         if brancos > 1500 and self.liberado:
45             self.contador += 1
46             self.liberado = False
47         elif brancos < 1500:
48             self.liberado = True
49         return self.contador
50
51
52 class Desenhos:
53     def __init__(self, x, y, w, h):
54         self.x = x
55         self.y = y
56         self.w = w
57         self.h = h
58
59     def desenhar_retangulo(self, frame, liberado):
60         if not liberado:
61             cv2.rectangle(frame, (self.x, self.y), (self.x + self.w,
                   self.y + self.h), (0, 255, 0), 4)
62         else:
63             cv2.rectangle(frame, (self.x, self.y), (self.x + self.w,
                   self.y + self.h), (255, 0, 255), 4)
64
65     def desenhar_informacoes(self, frame, brancos, contador):
66         horario_atual = datetime.now().strftime("%H:%M:%S")
67         cv2.putText(frame, str(brancos),
68                     (self.x - 30, self.y - 50),
69                     cv2.FONT_HERSHEY_SIMPLEX,
70                     1, (255, 255, 255), 1)
71         cv2.putText(frame, f"Horario: {horario_atual}",
72                     (self.x + 100, self.y + 120),
73                     cv2.FONT_HERSHEY_SIMPLEX,
74                     1, (0, 255, 255), 2)
75         cv2.putText(frame, str(contador),
```

```
76                      (self.x + 100, self.y),
77                      cv2.FONT_HERSHEY_SIMPLEX,
78                      3, (255, 0, 0), 5)
79
80
81  class ContadorGarrafaAI:
82      def __init__(self):
83          self.camera = Camera()
84          self.pdi = PDI()
85          self.contador = Contador()
86          self.desenhos = Desenhos(208, 142, 46, 332)
87
88      def iniciar(self):
89          while True:
90              frame = self.camera.obter_frame()
91              if frame is None:
92                  break
93
94              # Aplicar PDI
95              mascara = self.pdi.aplicar_mascara(frame)
96              brancos = self.pdi.contar_brancos(mascara)
97
98              # Atualizar contagem
99              contagem = self.contador.atualizar_contagem(brancos)
100
101             # Desenhar no frame
102             self.desenhos.desenhar_retangulo(frame, self.contador.
                    liberado)
103             self.desenhos.desenhar_informacoes(frame, brancos,
                    contagem)
104
105             # Exibir v deo
106             cv2.imshow("Video Principal", frame)
107
108
109         self.camera.liberar_camera()
110         cv2.destroyAllWindows()
```