INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA – CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELETRÔNICA

EMMANUEL REITZ GUESSER

SISTEMA DE VISÃO COMPUTACIONAL APLICADO NA INDÚSTRIA:
Inspeção de data e lote com OCR

FLORIANÓPOLIS, 2026.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA – CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELETRÔNICA

EMMANUEL REITZ GUESSER

SISTEMA DE VISÃO COMPUTACIONAL APLICADO NA INDÚSTRIA: Inspeção de data e lote com OCR

Trabalho de Conclusão submetido ao Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina como parte dos requisitos para obtenção do título de Engenheiro Eletrônico.

Orientador:
Prof. Carlos Gontarski Speranza, Dr. Eng.

FLORIANÓPOLIS, 2026.

Ficha de identificação da obra elaborada pelo autor.

Guesser, Emmanuel Reitz
Sistema de Visão Computacional Aplicado na Indústria: inspeção de data e lote com OCR / Emmanuel Reitz Guesser; orientação de Carlos Gontarski Speranza. - Florianópolis, SC, 2026.
70 p.

Trabalho de Conclusão de Curso (TCC) - Instituto Federal de Santa Catarina, Câmpus Florianópolis. Bacharelado em Engenharia Eletrônica. Departamento Acadêmico de Eletrônica.
Inclui Referências.

1. Visão Computacional. 2. Reconhecimento Óptico de Caracteres. 3. Automação Industrial. 4. Raspberry Pi. 5. Tesseract. I. Gontarski Speranza, Carlos. II. Instituto Federal de Santa Catarina. III. Sistema de Visão Computacional Aplicado na Indústria.

SISTEMA DE VISÃO COMPUTACIONAL APLICADO NA INDÚSTRIA

EMMANUEL REITZ GUESSER

Este trabalho foi julgado adequado para obtenção do título de Engenheiro Eletrônico e aprovado na sua forma final pela banca examinadora do Curso de Engenharia Eletrônica do Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina.

Florianópolis, 13 de fevereiro, 2026.

Banca Examinadora:

gov.br
Documento assinado digitalmente
CARLOS GONTARSKI SPERANZA
Data: 23/02/2026 08:43:33-0300
Verifique em https://validar.iti.gov.br

_____

Prof. Carlos Gontarski Speranza, Dr. Eng.
Instituto Federal de Santa Catarina

gov.br
Documento assinado digitalmente
DANIEL LOHMANN
Data: 23/02/2026 09:21:28-0300
Verifique em https://validar.iti.gov.br

_____

Prof. Daniel Lohmann, Me. Eng.
Instituto Federal de Santa Catarina

gov.br
Documento assinado digitalmente
ROBINSON PIZZIO
Data: 23/02/2026 14:55:40-0300
Verifique em https://validar.iti.gov.br

_____

Prof. Robinson Pizzio, Dr. Eng.
Instituto Federal de Santa Catarina

Dedico este trabalho a toda minha família,
amigos e professores que me apoiaram nesta jornada.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Adelmo e Marinéia, pelo incentivo e o encorajamento de trilhar meu caminho, tenho certeza que o apoio de vocês foi fundamental para a realização deste sonho e para a minha formação como indivíduo e profissional.

Aos meus amigos e colegas de classe, pelas horas de estudo compartilhadas, as discussões e os momentos de descontração foram essenciais para superar as dificuldades e seguir a jornada, mesmo que sigam caminhos diferentes, contribuíram com parte do meu aprendizado.

Aos professores que fizeram parte da minha formação na instituição, expresso minha gratidão pelo conhecimento transmitido e pelo comprometimento com um ensino de excelência. Um agradecimento especial ao Prof. Carlos Gontarski Speranza, por ter aceitado este convite de me orientar, sua paciência e incentivo foram fundamentais para o desenvolvimento e a conclusão deste trabalho.

RESUMO

Este trabalho apresenta o desenvolvimento de um protótipo de sistema de visão computacional para a automação da inspeção de códigos de data de validade e número de lote em embalagens industriais, visando a conformidade com a ANVISA. O objetivo geral foi projetar e validar um sistema capaz de realizar a leitura e validação automática de caracteres utilizando técnicas de pré-processamento e reconhecimento óptico de caracteres (OCR). A metodologia empregada envolveu o uso da plataforma Raspberry Pi 5, integrada a uma câmera USB, iluminação e sensores de disparo (*trigger*) fotoelétricos sincronizados por meio de uma placa de interface de entrada e saída isolada desenvolvida para o projeto. No desenvolvimento do software, utilizou-se a linguagem Python e a biblioteca OpenCV para a implementação de uma *pipeline* de pré-processamento que incluiu filtros de mediana, binarização pelo método de Otsu e operações morfológicas de fechamento, abertura e erosão para a normalização das imagens. O estudo realizou uma análise comparativa entre os algoritmos de OCR Tesseract e EasyOCR, além de implementar um treinamento incremental do motor neural do Tesseract especificamente para fontes matriciais (*dot-matrix*) utilizando a ferramenta *tesstrain*. Os resultados demonstraram que o Tesseract, após o treinamento específico e o pré-processamento adequado, atingiu 100% de assertividade em fontes matriciais inicialmente ilegíveis, operando com uma eficiência temporal aproximadamente dez vezes superior ao EasyOCR no hardware embarcado. Identificou-se, contudo, que as limitações do sensor *rolling shutter* causam distorções geométricas em capturas de alta velocidade, sugerindo a adoção de sensores *global shutter* para tais cenários. Conclui-se que a solução proposta é tecnicamente viável e eficaz para a inspeção em tempo real, validando o uso de ferramentas de software livre para fortalecer a confiabilidade operacional e a rastreabilidade na indústria.

Palavras-chave: Visão Computacional. Reconhecimento Óptico de Caracteres. Automação Industrial. Raspberry Pi. Tesseract.

ABSTRACT

This work presents the development of a prototype computer vision system for automating the inspection of expiration date and batch number codes on industrial packaging, ensuring compliance with ANVISA (Brazilian Health Regulatory Agency) standards. The overall objective was to design and validate a system capable of automatically reading and validating characters using preprocessing techniques and optical character recognition (OCR). The methodology employed involved the use of the Raspberry Pi 5 platform, integrated with a USB camera, lighting, and photoelectric trigger sensors synchronized through an isolated input/output interface board developed for the project. The software development utilized the Python language and the OpenCV library to implement a preprocessing pipeline that included median filters, binarization using the Otsu method, and morphological operations of closure, opening, and destruction for image normalization. This study conducted a comparative analysis between the Tesseract and EasyOCR OCR algorithms, and implemented incremental training of the Tesseract neural engine specifically for raster (point matrix) sources using the tesstrain tool. The results demonstrated that Tesseract, after specific training and appropriate preprocessing, achieved 100% accuracy on initially illegible raster sources, operating with a temporal efficiency approximately ten times greater than EasyOCR on embedded hardware. However, we identified that the limitations of the rolling shutter sensor cause geometric distortions in high-speed captures, suggesting the adoption of global rolling shutter sensor for such scenarios. We conclude that the solution is a viable and effective technique for real-time inspection, validating the use of open-source software tools to strengthen operational reliability and traceability in the industry.

Keywords: Computer Vision. Optical Character Recognition. Industrial Automation. Raspberry Pi. Tesseract.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CCD	*Charge-Coupled Device* (Dispositivo de Carga Acoplada)
CLP	Controlador Lógico Programável
CMOS	*Complementary Metal-Oxide-Semiconductor* (Semicondutor de Óxido Metálico Complementar)
CNN	*Convolutional Neural Network* (Rede Neural Convolucional)
CPU	*Central Processing Unit* (Unidade Central de Processamento)
CRNN	*Convolutional Recurrent Neural Network* (Rede Neural Recorrente Convolucional)
GPIO	*General Purpose Input/Output* (Entrada/Saída de Propósito Geral)
GPU	*Graphics Processing Unit* (Unidade de Processamento Gráfico)
I/O	*In/Out* (Entrada/Saída)
LSTM	*Long Short-Term Memory* (Memória de Curto Prazo de Longo Alcance)
ms	Milissegundos
OCR	*Optical Character Recognition* (Reconhecimento Óptico de Caracteres)
ROI	*Region of Interest* (Região de Interesse)

SUMÁRIO

1 INTRODUÇÃO

A quarta revolução industrial, ou Indústria 4.0, tem impulsionado uma transformação digital nos processos de manufatura. Neste cenário, a automação surge como um dos pilares fundamentais para o aumento da eficiência, a redução de custos e principalmente garantir a qualidade e segurança dos produtos. Dentre as tecnologias emergentes, a visão computacional se destaca como uma ferramenta capaz de emular a capacidade humana de percepção visual para realizar tarefas de inspeção, medição e controle em ambientes industriais com alta velocidade e precisão.

A rastreabilidade de produtos é um requisito indispensável na indústria moderna, especialmente nos setores alimentício, farmacêutico e de bens de consumo. Informações como data de validade e número de lote, impressas diretamente nas embalagens, são vitais para o controle de estoque, a logística e, em casos de necessidade, para a execução de recalls, protegendo a saúde do consumidor e a reputação da marca.

Neste contexto, o Reconhecimento Óptico de Caracteres (*OCR - Optical Character Recognition*) surge como uma solução tecnológica que permite a "leitura" e interpretação automática de textos impressos a partir de imagens digitais. A aplicação desta tecnologia para a inspeção de datas e lotes automatiza um ponto crítico do controle de qualidade, garantindo que cada produto que deixa a linha de produção esteja corretamente identificado e em conformidade com as normas. Este trabalho se propõe a explorar o desenvolvimento e a aplicação de um sistema com essa finalidade, analisando seus componentes, desafios e o impacto potencial na otimização de processos industriais.

1.1 Justificativa

A ausência de um sistema de verificação automatizado para validar, em tempo real, os códigos de data e lote impressos nas embalagens representa um risco operacional relevante para a indústria alimentícia. Informações incorretas, ilegíveis ou ausentes podem resultar em descumprimento de normas regulatórias, colocar em risco a saúde pública e comprometer a imagem da empresa perante o consumidor.

De acordo com a Resolução RDC nº 259/2002 da ANVISA (BRASIL, 2002), responsável por estabelecer o Regulamento Técnico de Rotulagem de Alimentos Embalados, todas as informações relacionadas ao consumo devem ser exibidas de forma clara e nítida em todos os produtos. A não conformidade com essas exigências sujeita a empresa às penalidades previstas segundo a Lei Federal nº 6.437/1977 (BRASIL, 1977), que dispõe sobre infrações à legislação sanitária. Além disso, a norma determina que dados como data de validade e lote de produção são elementos obrigatórios, fundamentais para a segurança do consumidor e para a transparência das informações disponibilizadas na embalagem.

Sob a perspectiva do consumidor, a data de validade é a principal garantia de que o produto está próprio para o consumo. Já para a empresa, o número de lote constitui a base do processo de rastreabilidade, permitindo identificar e isolar rapidamente qualquer ocorrência de desvio de qualidade.

Assim, a implementação de um sistema automatizado de inspeção por visão computacional surge como uma medida estratégica para assegurar a conformidade regulatória e aprimorar o controle de qualidade. Com o avanço das tecnologias de captura de imagem, câmeras de alta resolução, o aumento da capacidade de processamento e a evolução dos algoritmos de OCR, a automatização desse processo tornou-se mais viável, precisa e eficiente. Além de reduzir o risco de erro humano, um sistema desse tipo fortalece a confiabilidade operacional e contribui para a integridade do processo produtivo e da percepção de qualidade do consumidor.

1.2 Definição do Problema

As linhas de produção industriais operam em alta velocidade, exigindo que o processo de impressão e verificação de códigos de data e lote seja executado em uma fração de segundo, o que torna a inspeção humana impraticável.

Portanto, o problema central que este trabalho busca solucionar é: como desenvolver um sistema de visão computacional com alta acurácia que utilize OCR para automatizar a inspeção e validação dos códigos de data de validade e lote

impressos em embalagens na linha de produção, superando os desafios de velocidade e ruídos do ambiente industrial.

1.3 Objetivo Geral

O objetivo geral deste trabalho é projetar e desenvolver um protótipo de sistema de visão computacional para a inspeção e validação automática de caracteres de data de validade e número de lote impressos em embalagens, utilizando técnicas de pré-processamento e reconhecimento óptico de caracteres (OCR), além de fornecer a integração necessária para o uso industrial.

1.4 Objetivos Específicos

Para alcançar o objetivo geral, os seguintes objetivos específicos serão perseguidos:

a) Realizar uma revisão bibliográfica sobre os fundamentos da visão computacional, técnicas de processamento de imagem para otimização de leitura (pré-processamento) e as principais tecnologias e algoritmos de OCR disponíveis;

b) Implementar o algoritmo para extrair o texto via OCR e realizar a comparação de caracteres entre o texto extraído do objeto alvo com o texto de referência, dado isso verificar a acurácia do algoritmo;

c) Desenvolver a base necessária para a integração do sistema de visão com o ambiente físico da linha de produção, implementando a comunicação com sensores de *trigger* para o disparo da captura de imagem e com controladores para sinalizar a aprovação ou rejeição de embalagens;

d) Validar a eficácia do sistema proposto através de testes com um conjunto de amostras de imagens, avaliando a acurácia, velocidade de processamento e robustez, a fim de avaliar o desempenho do sistema.

1.5 Estrutura do Trabalho

Este trabalho está dividido da seguinte forma: primeiramente a fundamentação teórica aborda sobre os fundamentos da visão computacional, processamento de imagem e algoritmos de reconhecimento ópticos de caracteres (OCR). Após a metodologia detalha o desenvolvimento do protótipo, abrangendo hardware, software e os métodos de integração. Em seguida, são apresentados e discutidos os resultados dos testes de desempenho do sistema, analisando seu desempenho. Por fim, as considerações finais consolidam as conclusões do estudo, suas contribuições e limitações, além de propor sugestões para trabalhos futuros para aprimorar a solução.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentada uma revisão da literatura fundamental para a compreensão deste estudo. Será abordado o problema levantado neste estudo, onde são apresentados os conceitos da rastreabilidade e controle de qualidade na indústria de bens de consumo e então serão abordados os fundamentos de visão computacional e o processamento digital de imagem, além das ferramentas necessárias para o desenvolvimento deste trabalho.

2.1 Fundamento de Visão Computacional

A visão computacional é o campo que busca desenvolver sistemas capazes de interpretar imagens capturadas de cenas reais, com o objetivo de extrair informações úteis para automação e controle de processos industriais. Um sistema completo de visão computacional industrial combina hardware e software para adquirir imagens, processá-las e tomar decisões baseadas nos dados obtidos (WEST, 2021).

Atualmente a visão computacional está presente em diversas aplicações, por exemplo reconhecimento óptico de caracteres (OCR), inspeção mecanizada, automatização de logística e na área de medicina (Szeliski, 2022).

2.1.1 A Imagem Digital

A imagem digital é composta por pixels, que são pequenos elementos digitais que carregam informações de intensidade luminosa e cor. Neste estudo serão abordados três tipos fundamentais de imagens digitais, que são frequentemente utilizadas nos sistemas de visão computacional industrial (WEST, 2021):

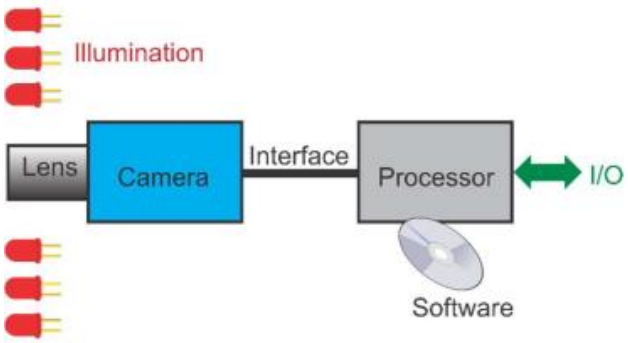
- Nas imagens coloridas, cada pixel apresenta componentes para as cores básicas vermelho, verde e azul (RGB). Cada componente é armazenado independentemente, a fim de representar a ampla gama de cores percebidas no espectro visível. Este modelo é o mais utilizado em monitores coloridos e câmeras, embora existam outros modelos de representação de cores (Gonzalez, 2010).

- Imagens em escala de cinza são formadas por pixels que armazenam valores de intensidade luminosa variando do preto absoluto (valor zero) ao branco puro (valor máximo), onde são representadas por valores de 8 bits, resultando em 256 tons de cinza possíveis. Em algumas aplicações, podem ser usados sensores que suportam 10 ou 12 bits para maior precisão. (WEST, 2021).
- Imagens binárias são simplificações das imagens em escala de cinza, contendo apenas dois níveis, geralmente 0 e 1, representando pixels claros ou escuros. A binarização é normalmente realizada por meio de um processo denominado limiarização (*thresholding*), onde valores abaixo de um limiar definido são considerados baixos (preto), e acima dele altos (branco) (WEST, 2021).

2.1.2 O Sistema de Visão Computacional Industrial

Um sistema de visão industrial é um conjunto integrado de componentes de hardware e software. Um sistema de visão de máquina típico é composto por cinco componentes principais: iluminação, lente, câmera, o processador de visão (que executa o software) e os canais de comunicação para interagir com outros maquinários (WEST, 2021).

Figura 1 - Componentes do sistema de visão



Fonte: West (2021).

Na figura 01 apresenta os componentes de um sistema de visão, inicia-se com a iluminação, que é projetada para iluminar a peça de forma a destacar a característica de interesse, em seguida, a lente foca a imagem da peça sobre o sensor da câmera e o sensor, por sua vez, converte a luz em uma imagem digital. Esta imagem é então enviada ao processador, que aplica algoritmos de processamento de

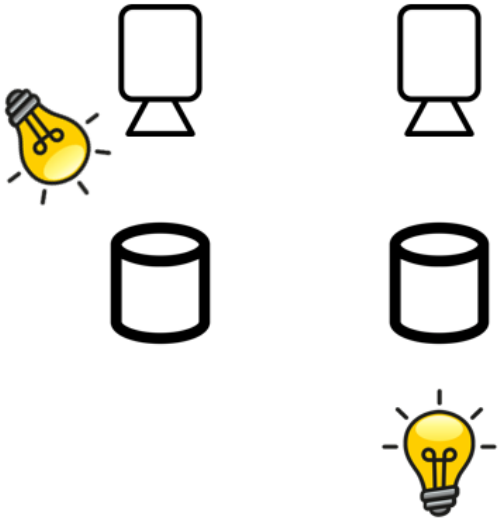
imagem para analisá-la e além de comunicar por meio da interface de I/O com outros equipamentos (WEST, 2021).

2.1.2.1 Iluminação

A iluminação é um dos elementos críticos para o sucesso de qualquer sistema de visão computacional industrial, pois afeta diretamente a qualidade da imagem capturada e a confiabilidade da análise subsequente (WEST, 2021).

De modo geral, as técnicas de iluminações podem ser divididas entre iluminação frontal ou traseira, no primeiro caso a fonte luminosa é posicionada do mesmo lado da câmera em relação ao objeto, iluminando diretamente a superfície a ser analisada, no outro caso, a luz é posicionada atrás do objeto em relação à câmera, criando um efeito de silhueta (WEST, 2021). A figura 2 exemplifica o posicionamento da iluminação nos dois cenários.

Figura 2 - Técnicas de Posicionamento de Luz. (a) Iluminação frontal. (b) Iluminação traseira



(a) (b)

Fonte: Adaptado WEST (2021)

A luz emitida pela iluminação, também pode ser direcionada ou difusa. Na direcionada, a luz é emitida em feixes estreitos e concentrados sobre regiões específicas, podendo projetar sombras e destacar a textura do objeto iluminado. Já a

iluminação difusa, a luz é espalhada de forma homogênea sobre a área, desse modo o sombreamento é menos evidente (WEST, 2021). A figura 3 mostra um comparativo do ângulo de emissão entre a iluminação direta e difusa.

Figura 3 - Tipos de emissão de iluminação. (a) Direcionada. (b) Difusa.

Fonte: Adaptado WEST (2021)

2.1.2.2 Lente

As lentes são responsáveis pela formação da imagem óptica sobre o sensor, determinando o campo de visão, distância focal, profundidade de campo e resolução óptica. Segundo West (2021), existem 3 tipos básicos de lente: entocêntricas, macro e telecêntricas. Na maior parte das aplicações são utilizadas lentes entocêntricas, estas podem ser focadas em uma faixa de distância de trabalho que se estende ao infinito e geralmente possuem uma abertura ajustável, como este tipo não foca de perto para atingir altas ampliações, as lentes macro são projetadas para atender a essa necessidade. Já as lentes telecêntricas, garantem ampliação constante e reduzem distorções de perspectiva, se comparada as anteriores (WEST, 2021).

2.1.2.3 Câmera

A câmera é o componente responsável pela captura da imagem formada pela lente, ela a converte para uma imagem digital para enfim ser processada. Essa imagem podendo ser monocromática, colorida ou até 3D.

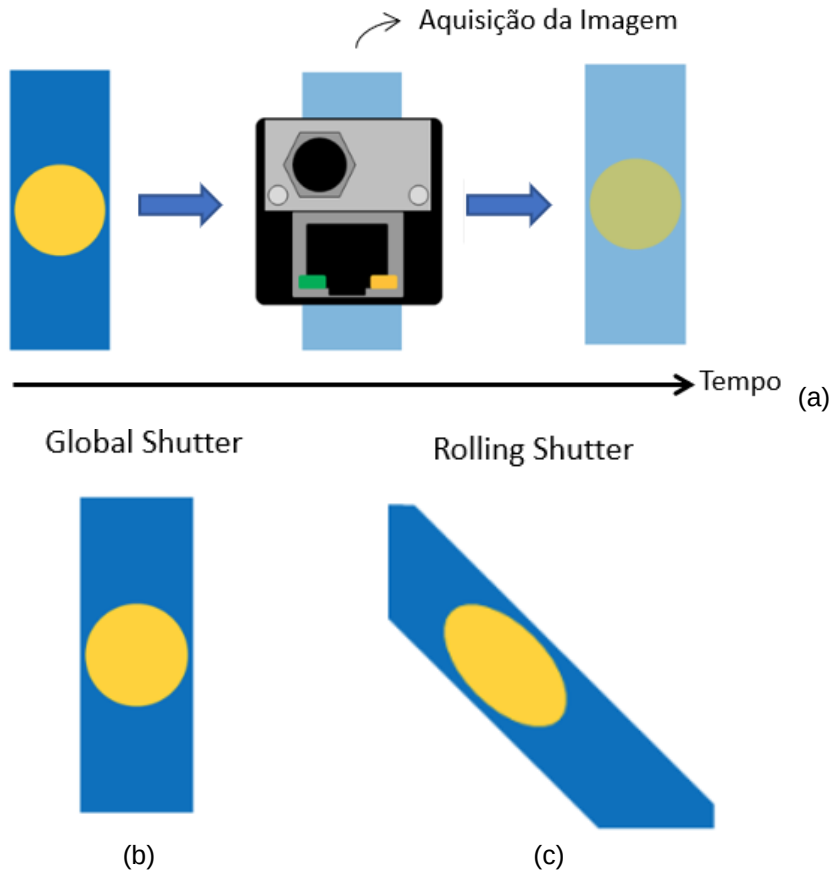
O módulo de câmera utiliza principalmente sensores dos tipos CCD (*charge-coupled device*) ou CMOS (*complementary metal oxide on silicon*),

responsáveis por converter a luz captada em sinais elétricos que formam a imagem digital. Historicamente, os CCD ofereciam melhor desempenho em aplicações de alta qualidade de imagem, no entanto, com a evolução tecnológica, os sensores CMOS tornaram-se predominantes na maioria das câmeras digitais atuais (SZELISKI, 2022).

Um aspecto técnico fundamental em câmeras são o tipo de obturador empregado, este é o mecanismo de exposição do sensor, que define como os pixels capturam a luz. Estes podem ser classificados em obturador global (*global shutter*) ou obturador rolante (*rolling shutter*).

No sensor com obturador global, todos os pixels que compõe a imagem iniciam e terminam o tempo de exposição simultaneamente. Já o obturador rolante, realiza a exposição e a leitura dos pixels de forma sequencial, linha por linha, formando a imagem (BAUMER, 2019).

Figura 4 - Tipos de obturador (a) Exemplo de uma aplicação com movimento do Objeto. (b) Imagem com obturador global. (c) Imagem com obturador rolante



Fonte: Adaptado BAUMER (2019)

Na figura 4, observa-se o aspecto da imagem formada pelos dois tipos de obturadores, quando o objeto está em alta velocidade. Nota-se que com obturador global, o sensor captura a cena em um único instante, resultando em uma imagem que "congela" o objeto observado, já com o obturador rolante, observamos um efeito de deslizamento.

2.1.2.4 Sinal de Sincronização (Trigger)

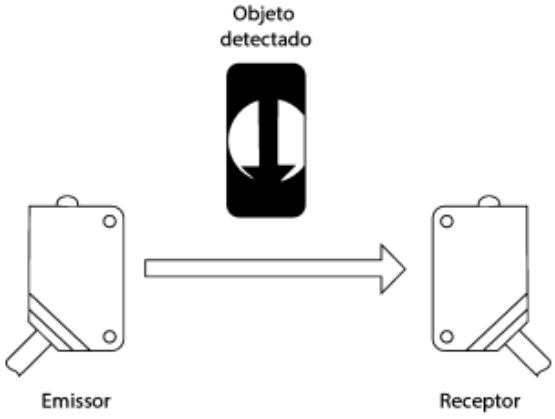
Segundo Lamb (2015), nos sistemas de visão industrial é necessário um sinal de sincronização para controlar o momento exato de captura da imagem, garantindo a sincronização entre o movimento do objeto e a aquisição visual, obtendo uma imagem nítida que é crucial para o funcionamento do sistema. Esse sinal é normalmente chamado de *trigger* e é gerado a partir de um sensor fotoelétrico.

O sensor fotoelétrico utiliza um feixe de luz emitido por um LED ou laser e um receptor que detecta se a luz é recebida ou interrompida pelo objeto (LAMB, 2015). Esse controle permite que o sensor dispare o sinal de *trigger* para a câmera captar a imagem no instante preciso em que o objeto está na posição correta.

Segundo Lamb (2015), há três tipos principais de sensores fotoelétricos:

- **Sensor de barreira**: esse modelo é composto por um emissor e receptor separados, onde o feixe é interrompido pelo objeto detectando o sinal como visto na figura 5;

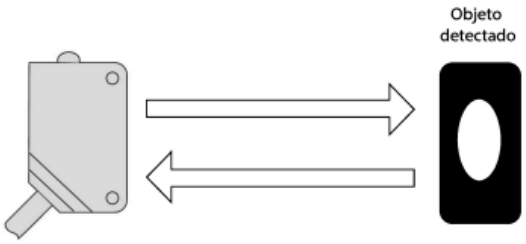
Figura 5 - Sensor barreira



Fonte: LAMB (2015)

- **Sensor difuso**: nesse caso o emissor e o receptor estão no mesmo dispositivo, em que a luz emitida é refletida pelo objeto, sinalizando sua detecção conforme ilustrado na figura 6.

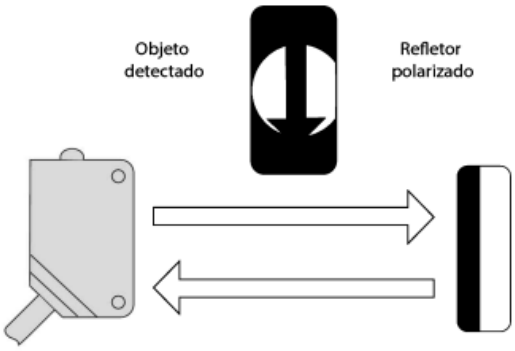
Figura 6 - Sensor difuso



Fonte: LAMB (2015)

- **Sensor retrorefletivo**: o emissor e receptor também estão no mesmo dispositivo como anteriormente, porém utiliza-se um refletor polarizado para refletir a luz em 90° para o receptor, conforme visto na figura 7, e desta forma, minimizando interferência com objetos reflexivos.

Figura 7 - Sensor Retrorefletivo



Fonte: LAMB (2015)

2.1.3 Entrada e saída de dados

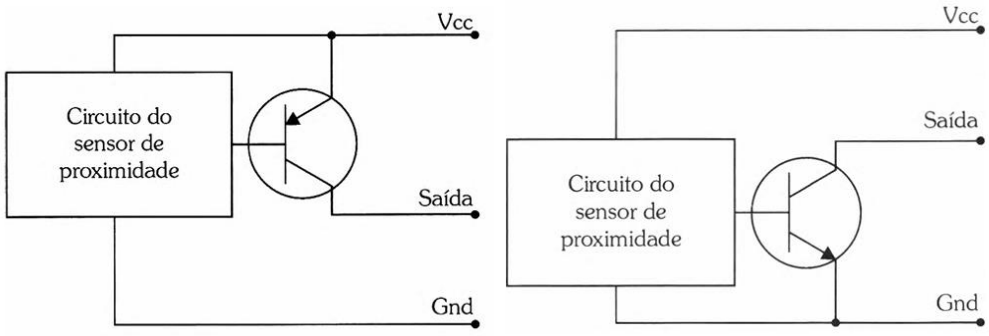
Segundo Lamb (2015), na maior parte dos sistemas de controle na indústria utilizam-se de sinais digitais, esses sinais são utilizados para entrada de sensores, interruptores e botões, e também como saída do sistema, por exemplo, em motores e válvulas.

Os sinais digitais normalmente operam em tensões de 24 VCC, porém dependendo das especificações do equipamento e do ambiente industrial podem utilizar de tensões menores (LAMB, 2015).

Segundo Franchi e Camargo (2008), a isolação elétrica é comumente realizada por meio de optoacopladores, estes utilizam um LED para produzir um pulso digital com infravermelho, e essa luz é detectada por um fototransistor que gera um pulso de tensão correspondente no circuito lógico, sendo que o espaço físico entre os componentes internos do optoacoplador garante a necessária separação galvânica entre as seções.

Em relação às tipologias de conexão, as entradas digitais de um controlador podem ser classificadas como do tipo fonte (*sourcing)*, também chamadas de PNP, ou do tipo dreno (*sinking)*, conhecidas como NPN (Franchi; Camargo, 2008).

Figura 8 - (a) Circuito PNP. (b) Circuito NPN.



(a) (b)

Fonte: FRANCHI; CAMARGO (2008)

Como pode ser visto na figura 8, em um sensor com saída PNP, o nível lógico comuta entre o fornecimento de uma tensão equivalente à alimentação e um circuito aberto. Por outro lado, sensores do tipo NPN operam com lógica negativa, enviando um sinal de referência ou negativo para a entrada do controlador ao detectarem um objeto (FRANCHI; CAMARGO, 2008).

2.2 Processamento de Imagem

O processamento de imagem é uma área que compreende métodos e técnicas para manipulação de imagem digitais. Ele é fundamental na preparação da

imagem para conseguirmos aplicar o algoritmo de OCR que está sendo desenvolvido neste trabalho.

Este capítulo apresenta os principais fundamentos, ferramentas e operadores do processamento digital de imagens, desenvolvendo a base teórica e prática essencial tanto para compreensão de conceitos quanto para o desenvolvimento das soluções enfrentadas.

2.2.1 Limiarização de Imagens

A limiarização é um dos procedimentos clássicos e mais utilizados para segmentação de imagens digitais. Constitui uma etapa fundamental em aplicações como reconhecimento de padrões, análise de formas, extração de objetos e preparação de dados para pós-processamento morfológico, especialmente por sua simplicidade conceitual aliada à eficiência computacional (GONZALEZ; WOODS, 2010; PEDRINI; SCHWARTZ, 2008). No entanto, apesar da aparente simplicidade, a escolha adequada do critério de limiarização e seu correto emprego impactam diretamente na qualidade dos resultados obtidos.

Segundo Gonzalez (2010), dada uma imagem *f(x,y)*, compostas por objetos claros e um fundo escuro, podemos agrupar a imagem em dois grupos dominantes onde um limiar *T* os separa. Então para todo ponto (x, y) maior que *T* é chamado de *ponto do objeto*, do outro modo ele é considerado *ponto de fundo*. Dessa forma, a imagem resultante pode ser definida por:

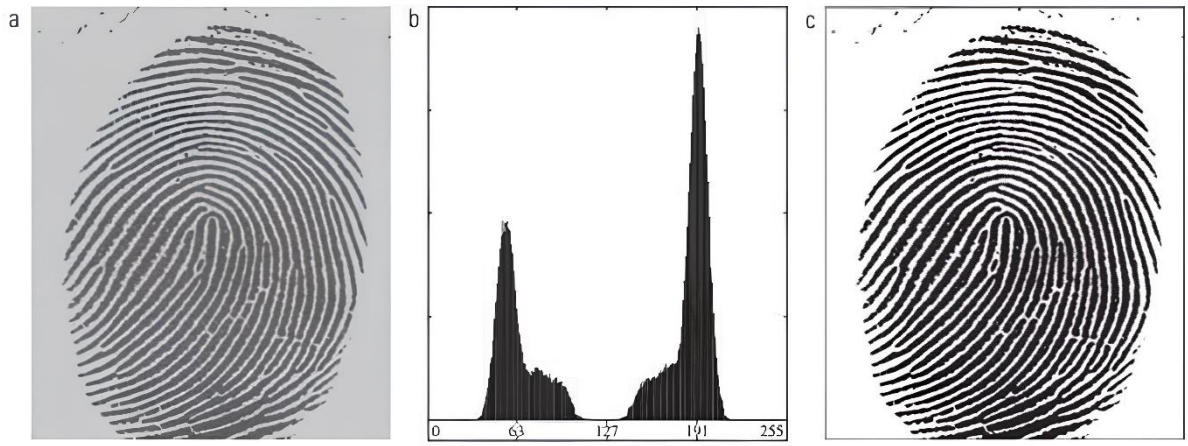
$$g(x,y) = \begin{cases} 1, & f(x,y) > T \\ 0, & f(x,y) \leq T \end{cases} \quad (1)$$

De acordo com Pedrini (2008), quando no processo de limiarização é utilizado um único valor de *T* para toda a imagem, ele é chamado de *limiarização global*, dessa forma, imagens que possuem certas variações nos níveis de cinza ou não possuem uma iluminação uniforme, o resultado pode não ser adequado. Para esse tipo de situação é necessário utilizar diferentes limiares em cada região da imagem, esse processo é conhecido por *limiarização local.*

2.2.1.1 Limiarização global

O processo de escolha do limiar pode ser simples, empregando a análise visual do histograma, ou automatizado, recorrendo a algoritmos quantitativos. Gonzalez e Woods (2010) enfatizam que a eficácia do processo de limiarização está diretamente relacionada à separação dos picos e à profundidade dos vales do histograma: contrastes nítidos facilitam a segmentação, enquanto o ruído, variações de iluminação e diferenças de refletância podem diminuir significativamente a qualidade dos resultados.

Figura 9 - Segmentação via limiarização global. (a) Imagem de uma impressão digital. (b) Histograma. (c) Imagem limiarizada.



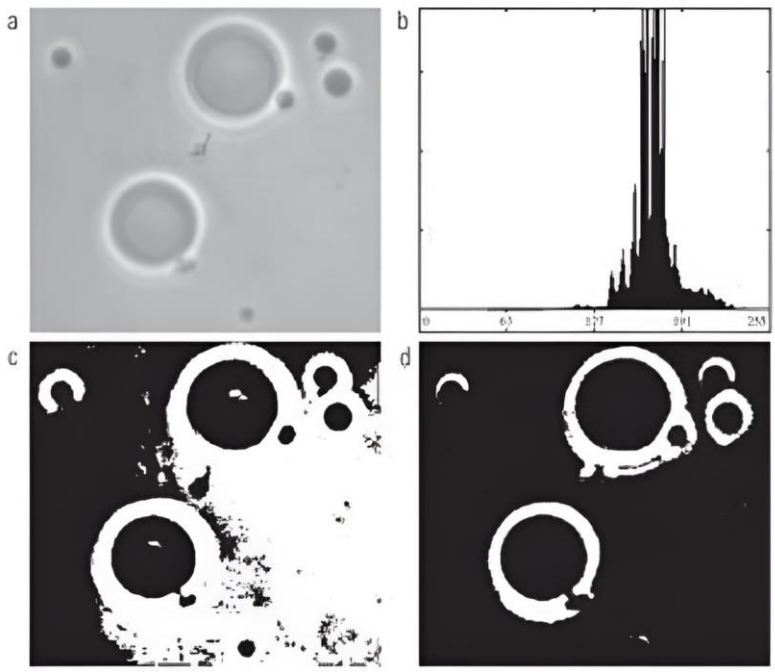
Fonte: GONZALEZ; WOODS (2010)

Na figura 09 temos o processo de limiarização global, primeiro em (a) apresenta-se uma imagem de uma impressão digital baixo contraste; em (b) observa-se o histograma correspondente e dado um limiar $T = 125$ obtém-se a imagem (c), neste caso como temos uma separação clara entre o objeto e fundo, o que também pode ser observada pelos dois picos no histograma, o algoritmo funciona sem dificuldades.

No entanto, para algumas imagens mais complexas, algoritmos automáticos se fazem necessários para a escolha do ponto de limiarização. Entre os métodos automáticos destaca-se o de Otsu (1979), que propõe a escolha do limiar que maximiza a separabilidade entre as classes, ou seja, que maximiza a variância entre classes em relação à variância total (GONZALEZ; WOODS, 2010; PEDRINI; SCHWARTZ, 2007). O algoritmo de Otsu é estatístico e utiliza apenas o histograma,

sendo amplamente reconhecido por sua robustez para imagens com clara separação de intensidades. A figura 10 mostra o processo de limiarização com a imagem de microscópio ótico de células polimerosomas.

Figura 10 - Limiarização global com método de Otsu. (a) Imagem de um microscópio. (b) Histograma da imagem. (c) Resultado obtido com uma estimativa. (d) Resultado obtido pelo algoritmo de Otsu.



Fonte: GONZALEZ; WOODS (2010)

Na figura 10 (a) temos a imagem original e em (b) seu histograma, como não há uma definição clara entre o objeto e fundo, a definição do ponto de segmentação *T.* A figura 10 (c) mostra o resultado obtido por uma estimativa, já figura 10 (d) mostra o resultado obtido pelo método de Otsu, se mostrando superior que a anterior.

2.2.1.2 Limiarização local

Quando a uniformidade das condições de iluminação não pode ser garantida, a limiarização global pode deixar de ser eficaz. A alternativa é a limiarização local ou adaptativa, onde o limiar é calculado para pequenas regiões da imagem em função de propriedades estatísticas locais, como média, mediana, mínimo, máximo e desvio padrão dos pixels da vizinhança (PEDRINI; SCHWARTZ, 2008).

Entre os métodos mais estudados estão os de Bernsen (baseado na média dos valores máximo e mínimo na vizinhança), Niblack e Sauvola (utilizam média e desvio padrão local, ajustando-se para iluminação não uniforme e preservação de detalhes em documentos e superfícies texturizadas). O tamanho da janela local é fator crucial: deve ser suficientemente grande para garantir uma boa estimativa estatística, mas pequena o bastante para preservar detalhes e não ser afetada significativamente por ruído (PEDRINI; SCHWARTZ, 2008).

Figura 11 - Comparação entre a limiarização global e local. (a) Imagem de um texto escrito. (b) Imagem limiarizada pelo algoritmo de Otsu. (c) Imagem limiarizada localmente.

Fonte: GONZALEZ; WOODS (2010)

Na figura 11 (a), temos a imagem de um texto escrito e devido a diferença na iluminação dela, o resultado da limiarização global não possui resultados adequados como pode-se ver em 11(b), já quando se faz o uso de limiarização local em 11(c), alcança-se um resultado superior ao anterior.

2.2.2 Filtragem espacial

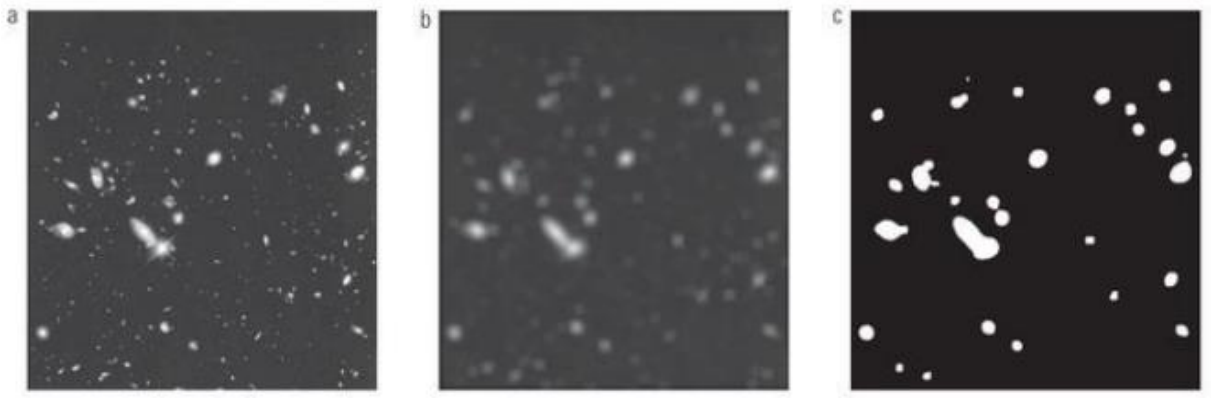
A filtragem espacial é uma das ferramentas mais versáteis e fundamentais no processamento digital de imagens, sendo empregada tanto para a remoção de ruídos quanto para o realce de detalhes e bordas (GONZALEZ; WOODS, 2010). Os filtros podem ser aplicados para suavizar pequenas variações indesejadas de intensidade ou, ao contrário, enfatizar descontinuidades e regiões de transição.

2.2.2.1 Filtros de suavização

Os filtros de suavização são utilizados para reduzir o ruído presente na imagem, eliminar pequenos detalhes, conectar descontinuidades e suavizar transições abruptas de intensidade (GONZALEZ; WOODS, 2010).

A suavização pode ser realizada por meio de filtros lineares, como o filtro de média (aritmética): cada pixel da imagem é substituído pela média dos valores de intensidade dos pixels na vizinhança definida pela máscara. Esse filtro é um típico passa-baixa, atuando na atenuação de ruídos, mas também causa certo borramento nas bordas, podendo reduzir detalhes finos importantes (GONZALEZ; WOODS, 2010). Na figura 12 podemos observar efeito do filtro de média, onde temos como exemplo a imagem gerada pelo telescópio Hubble em órbita da Terra, com isso é aplicado um filtro de média e então ela é limiarizada, gerando um maior destaque dos objetos de maior dimensão na imagem.

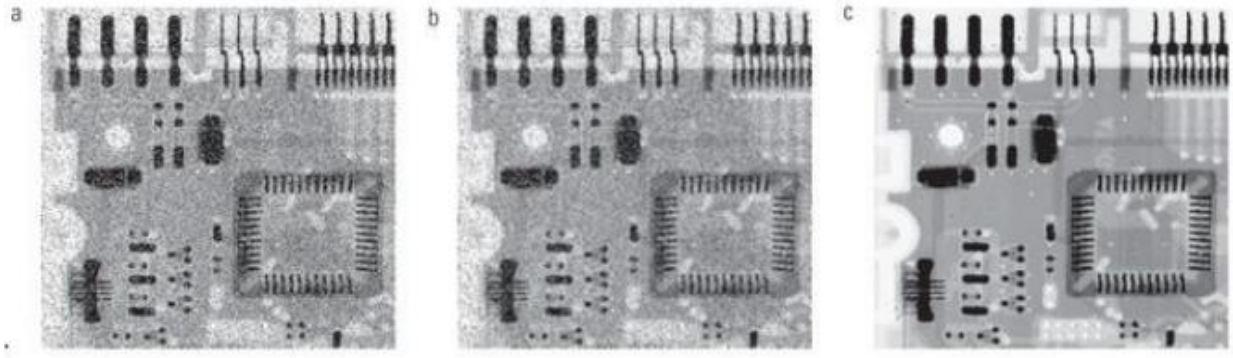
Figura 12 - Aplicação do filtro de média. (a) Imagem original. (b) Imagem com filtro média. (c) Imagem limiarizada.



Fonte: GONZALEZ; WOODS (2010)

Já o filtro de mediana é exemplo de filtro não linear muito utilizado, principalmente para ruído impulsivo (sal e pimenta), pois preserva bordas melhor que o filtro de média. Aqui, para cada pixel, substitui-se o valor original pela mediana dos valores de intensidade na vizinhança (GONZALEZ; WOODS, 2010). Na figura 13 temos uma aplicação do filtro de mediana, apresenta-se uma imagem de uma placa eletrônica corrompida por um ruído sal e pimenta, em (b) é aplicado um filtro de média e (c) um filtro de mediana, pode-se notar que o resultado do filtro de mediana é superior à média.

Figura 13 - Aplicação do filtro de mediana. (a) Imagem original. (b) Imagem com filtro de média. (c) Imagem com filtro de mediana.



Fonte: GONZALEZ; WOODS (2010)

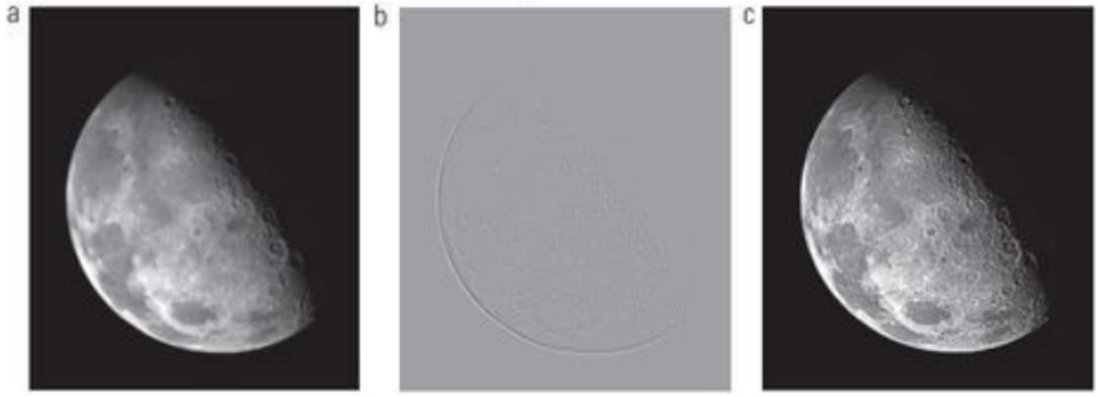
2.2.2.2 Filtros de Aguçamento

Os filtros de aguçamento têm como principal objetivo realçar regiões de descontinuidade, como bordas, linhas e alterações abruptas de intensidade, enfatizando detalhes que podem estar “diluídos” após a suavização ou pela natureza do próprio sistema de aquisição. Segundo Gonzalez e Woods (2010), os seguintes métodos são empregados para aumentar a nitidez da imagem:

- Filtros com derivadas: baseiam-se nas derivadas de primeira ou segunda ordem, que são operadas no domínio espacial por máscaras específicas. Derivadas de primeira ordem (ex.: gradiente, operadores de Sobel, Prewitt e Roberts) realçam transições de intensidade, indicando bordas. Enquanto que derivadas de segunda ordem (ex.: Laplaciano) tendem a tornar detalhes finos ainda mais evidentes (GONZALEZ; WOODS, 2010).
- Laplaciano: operador isotrópico (invariante à rotação) de segunda ordem, bastante usado para aguçamento de imagens, definido por uma máscara que inclui as diferenças entre o pixel central e seus vizinhos. A imagem aguçada pode ser obtida pela soma (ou subtração, dependendo do sinal da máscara) da imagem original com a resposta do Laplaciano. (GONZALEZ; WOODS, 2010). Na figura 14 temos um exemplo utilizando o laplaciano, em (a) temos a imagem original, (b) a máscara gerada pelo

laplaciano e em (c) a imagem resultante da soma da imagem original e da máscara.

Figura 14 - Aguçamento de imagens utilizando o laplaciano. (a) Imagem original. (b) Máscara gerada. (c) Soma da imagem original com a máscara.



Fonte: GONZALEZ; WOODS (2010)

- Máscara de nitidez (Unsharp Mask) e filtragem high-boost: consiste em subtrair uma versão suavizada da imagem original (obtendo uma máscara de realce), e depois somar uma fração ou múltiplo dessa máscara à imagem original. Esse método é amplamente utilizado na indústria gráfica e fotografia digital para aumentar a nitidez e o contraste local da imagem (GONZALEZ; WOODS, 2010).

2.2.3 Operações Morfológicas

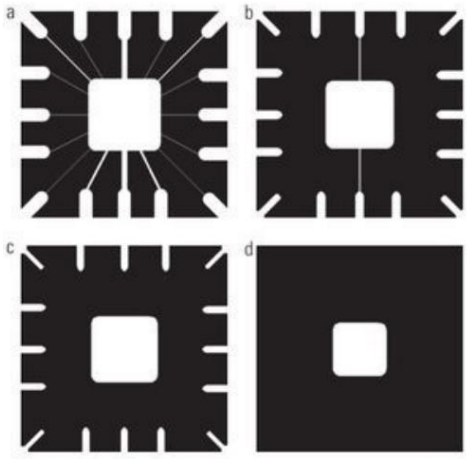
A morfologia matemática no processamento digital de imagens usa operadores que permitem manipular e analisar objetos presentes em imagens binárias (preto e branco), utilizando como ferramenta central o conceito de elemento estruturante. Os operadores morfológicos básicos e compostos são essenciais para extrair atributos de formato, eliminar ruídos, corrigir imperfeições e preparar imagens para análise de regiões (GONZALEZ; WOODS, 2010).

2.2.3.1 Erosão

A erosão é uma operação fundamental cujo efeito é “encolher” ou afinar objetos em uma imagem. Ela atua sobre os pixels de maior valor (branco) removendo-

os das bordas das regiões, reduzindo seu tamanho e podendo eliminar ruídos pequenos ou conexões delicadas. Ao aplicar a erosão, um pixel do objeto será mantido somente se todo o elemento estruturante couber inteiramente no objeto naquela posição. Assim, regiões menores que o elemento estruturante desaparecem (GONZALEZ; WOODS, 2010). Na figura 15, primeiro temos a imagem original e em seguida a imagem erodida utilizando elementos estruturantes quadrados de tamanhos 11 x 11, 15 x 15 e 45 x 45, respectivamente.

Figura 15 - Exemplos da utilização de erosão. (a) Imagem original. (b) a (d) Imagem erodida por elemento estruturante com tamanho de 11x11, 15x15 e 45x45, respectivamente com todos valores iguais a 1.

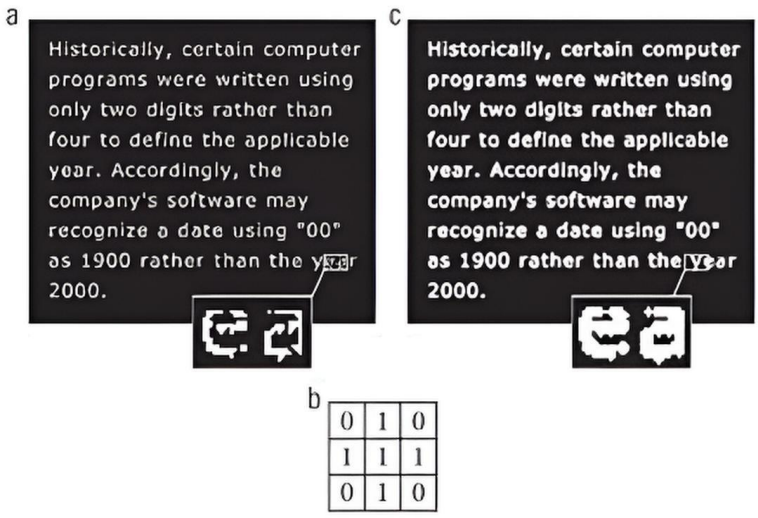


Fonte: GONZALEZ; WOODS (2010)

2.2.3.2 Dilatação

A dilatação é a operação dual à erosão. Ela faz “crescer” ou expandir os objetos em uma imagem, preenchendo pequenas lacunas e conectando regiões vizinhas. A dilatação insere um pixel na saída sempre que o elemento estruturante, posicionado sobre esse pixel, encontrar sobreposição com qualquer parte do objeto na imagem original (GONZALEZ; WOODS, 2010). Isso pode conectar objetos próximos e preencher buracos em suas bordas como pode ser visto na figura 16, onde em (a) apresenta-se uma imagem de um texto, (b) o elemento estruturante e (c) a dilatação de (a) por (b).

Figura 16 - Exemplo da utilização de dilatação. (a) Imagem original. (b) Imagem dilatada.



Fonte: GONZALEZ; WOODS (2010)

2.2.3.3 Abertura

A abertura é uma combinação de erosão seguida de dilatação, utilizando o mesmo elemento estruturante. Essa sequência é ideal para remover pequenas protuberâncias, ramos ou pontos ruidosos, sem alterar significativamente o formato ou o tamanho dos objetos principais. A abertura suaviza contornos, rompe estreitamentos e elimina pequenos objetos isolados (GONZALEZ; WOODS, 2010).

2.2.3.4 Fechamento

O fechamento consiste na aplicação da dilatação seguida de erosão. É usado para preencher pequenos buracos, lacunas ou falhas internas nos objetos, conectar regiões próximas e suavizar “vales” ou reentrâncias no contorno, preservando o formato principal dos objetos (GONZALEZ; WOODS, 2010).

2.3 Reconhecimento óptico de caracteres

O Reconhecimento Óptico de Caracteres, do inglês *Optical Character Recognition* (OCR), é uma tecnologia que converte imagens de texto em formato editável e legível por máquina, permitindo a digitalização automatizada de

documentos impressos, manuscritos ou digitalizados (IBM, 2024). Esta tecnologia desempenha papel fundamental na automação de processos industriais que demandam leitura e validação de informações textuais impressas em produtos, como datas de validade, números de lote e códigos de rastreabilidade (COGNEX, 2023).

2.3.1 Fundamentos e História do OCR

Os primeiros desenvolvimentos relacionados ao OCR remontam à década de 1920, quando Emanuel Goldberg criou uma máquina capaz de ler caracteres e convertê-los em código telegráfico, estabelecendo as bases conceituais para a leitura automatizada por máquinas (AWS, 2022).

Figura 17 - Evolução do OCR

Fonte: Adaptado AWS (2022), IBM (2024).

Nas décadas de 1950 e 1960, o OCR começou a tomar forma como tecnologia comercial, com empresas como a RCA desenvolvendo sistemas capazes de ler fontes específicas para aplicações bancárias e postais, sendo utilizados principalmente para automatizar o processamento de cheques e triagem de correspondências (AWS, 2022). Durante esse período, foram criadas as fontes OCR-A e OCR-B, projetadas especificamente para serem facilmente legíveis tanto por

humanos quanto por máquinas, permitindo que o OCR se tornasse mais consistente em aplicações financeiras e governamentais (AWS, 2022).

Um marco significativo ocorreu em 1974, quando Ray Kurzweil fundou a Kurzweil Computer Products, Inc. e desenvolveu o OCR *omni-font*, capaz de reconhecer textos impressos em praticamente qualquer fonte, onde essa tecnologia foi utilizada para criar uma máquina de leitura destinada a pessoas com deficiência visual (IBM, 2024). Em 1980, Kurzweil vendeu sua empresa para a Xerox, que posteriormente a transformou na Scansoft, mais tarde incorporada à Nuance Communications (IBM, 2024).

Na década de 1990, a tecnologia OCR se popularizou com a digitalização de jornais históricos e, desde então, passou por diversas melhorias tecnológicas (IBM, 2024). Nos anos 2000, redes neurais e as primeiras técnicas de aprendizado de máquina permitiram que o OCR superasse as limitações de fontes e layouts fixos, possibilitando a interpretação de texto manuscrito, digitalizações de baixa qualidade e layouts complexos com precisão muito superior (AWS, 2022).

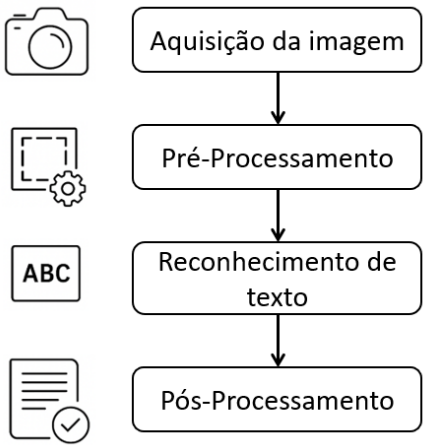
Atualmente, o OCR evoluiu de uma ferramenta de nicho para uma tecnologia fundamental na transformação digital, estando integrado em aplicativos móveis, plataformas de automação empresarial, suportando múltiplos idiomas e realizando captura de imagem em tempo real de forma contextualmente consciente (AWS, 2022).

2.3.2 Definição e Funcionamento do OCR

O OCR pode ser definido como o processo que converte uma imagem contendo texto em um formato de texto legível por máquina (AWS, 2022). Os sistemas de OCR utilizam uma combinação de hardware e software para converter textos físicos impressos em texto legível por máquina (IBM, 2024).

O processo completo de OCR pode ser dividido em etapas conforme na figura 18.

Figura 18 - Etapas do processo de OCR



Fonte: Elaboração própria. (2025)

Aquisição de imagem: Todas as páginas do documento são copiadas e, em seguida, o mecanismo de OCR converte o documento digital em uma versão binarizada (preto e branco). A imagem digitalizada ou *bitmap* é analisada em partes claras e escuras, identificando as áreas escuras como caracteres que precisam ser reconhecidos, enquanto as áreas claras são identificadas como plano de fundo (IBM, 2024; KONOVALCHUK, 2024).

Pré-processamento: A imagem digital é limpa para remover pixels estranhos e preparar o material para reconhecimento. Este processamento pode incluir correções de inclinação para alinhar o texto horizontalmente, remoção de ruídos e suavização de bordas, binarização, detecção de linhas e palavras; reconhecimento de script em documentos multilíngues; e segmentação ou isolamento de caracteres (IBM, 2024; KONOVALCHUK, 2024).

Reconhecimento de texto: As partes escuras são processadas para localizar letras alfabéticas, dígitos numéricos ou símbolos, normalmente tratando um caractere, uma palavra ou um bloco de texto de cada vez (IBM, 2024). Esta etapa utiliza algoritmos específicos de reconhecimento que serão detalhados na seção 2.3.3.

Pós-processamento: As informações coletadas são armazenadas como um arquivo digital. Sistemas mais avançados mantêm tanto a imagem original quanto as versões pós-OCR para facilitar a comparação e um gerenciamento de documentos mais completo (IBM, 2024). Nesta etapa, a precisão do OCR pode ser aumentada se

a saída for limitada por uma lista de palavras permitidas no documento e conhecimento gramatical da língua para corrigir erros (PETITCLERC, 2019).

2.3.3 Algoritmos *open-source* de OCR

Diversos algoritmos de OCR *open source* têm se consolidado como opções de referência tanto em pesquisa quanto em aplicações industriais e de software, destacando-se principalmente o Tesseract OCR e o EasyOCR. Esses algoritmos oferecem suporte a múltiplos idiomas, integração com bibliotecas de visão computacional e possibilidade de personalização, permitindo sua adoção em sistemas de inspeção, digitalização documental, automação de processos e aplicações em tempo real, com custo reduzido de licença (TESSERACT OCR, 2019; JAIDEDAI, 2020).

2.3.3.1 Tesseract OCR

Tesseract é um dos algoritmos de OCR open source mais antigos e amplamente utilizados, originalmente desenvolvido pela Hewlett-Packard nos anos 1980–1990 e posteriormente mantido pela Google (SMITH, 2007).

A partir da versão 4, o Tesseract incorporou um novo motor baseado em redes neurais recorrentes do tipo LSTM (Long Short-Term Memory), focado em reconhecimento de linhas de texto, mantendo ao mesmo tempo o motor legado baseado em padrões, o que permite selecionar diferentes modos de operação por meio do parâmetro OCR Engine Mode (OEM). Esse motor neural aumenta significativamente a acurácia em textos impressos, especialmente em cenários com fontes variadas e imagens mais desafiadoras, contando com modelos pré-treinados para mais de 100 idiomas e a possibilidade de ajuste fino ou treinamento de novos modelos específicos para domínios particulares (KONOVALCHUK, 2024; TESSERACT OCR, 2015).

A configuração do Tesseract permite ainda ajustar parâmetros como o modo de segmentação de página (Page Segmentation Mode – PSM), o idioma e arquivos de configuração adicionais, proporcionando flexibilidade para diferentes tipos de documentos e cenários de aquisição (TESSERACT OCR, 2015).

Além disso, o Tesseract oferece suporte ao treinamento incremental através da ferramenta *tesstrain*, dessa forma, fontes não padronizadas ou específicas que não existem no modelo pré-treinado, podem ser reconhecidas pelo algoritmo.

Existem três opções de treinamento, a primeira consiste em um ajuste fino do modelo partindo de uma linguagem já treinada, dessa forma novos dados são adicionados, corrigindo diferenças sutis, funcionando até mesmo com uma quantidade pequena de dados. A segunda é treinando novamente a camada superior usando novos dados, se a opção anterior não funcionar. E por último, podemos retreinar o modelo do zero, nesse caso é necessário um conjunto de treinamento muito representativo, caso contrário, o desempenho nos dados de treinamento pode ser bom, mas não refletir na realidade (TESSERACT OCR, 2022).

2.3.3.2 EasyOCR

EasyOCR é uma biblioteca de OCR open source mais recente, escrita em Python e construída para ser “pronta para uso”, oferecendo suporte nativo à mais de 80 idiomas e múltiplos alfabetos (latino, chinês, árabe, cirílico, entre outros), com foco em integração simples com aplicações de visão computacional (KONOVALCHUK, 2024). A biblioteca foi desenvolvida sobre *frameworks* de deep learning, utilizando modelos de detecção e reconhecimento de texto de última geração, o que a torna especialmente adequada para cenários com fontes diversas, layouts complexos e imagens capturadas em condições não controladas (JAIDEDAI, 2020).

No *pipeline* típico do EasyOCR, a etapa de detecção de texto utiliza modelos como CRAFT ou DB para localizar regiões de texto na imagem, enquanto o reconhecimento é realizado por uma arquitetura CRNN (Convolutional Recurrent Neural Network), combinando extração de características por redes convolucionais (por exemplo, ResNet ou VGG), rotulagem sequencial com LSTM e decodificação via CTC (*Connectionist Temporal Classification*) (MAHAJAN, 2023). Além disso, a API do EasyOCR foi projetada para ser simples, reduzindo a barreira de entrada: poucas linhas de código são suficientes para carregar o modelo, processar uma imagem e obter o texto reconhecido, o que favorece sua adoção em protótipos e sistemas industriais com desenvolvimento ágil (MAHAJAN, 2023).

3 METODOLOGIA

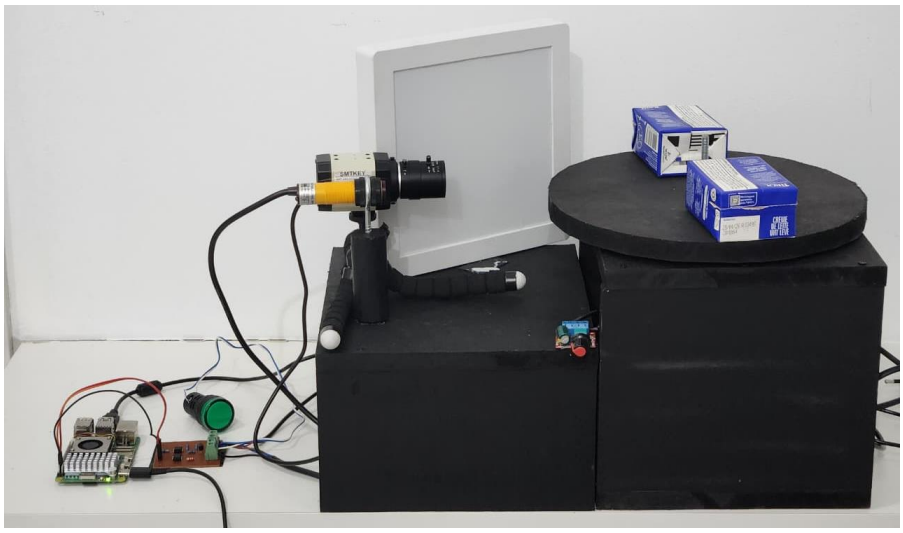
A metodologia adotada neste trabalho visa o desenvolvimento, implementação e validação de um sistema de visão computacional capaz de inspecionar automaticamente códigos de data de validade e número de lote impressos em embalagens de produtos industriais, utilizando processamento de imagens e algoritmos de Reconhecimento Óptico de Caracteres (OCR).

3.1 Procedimento Experimental

O procedimento experimental foi organizado em etapas, desde a montagem física do sistema até a validação do desempenho do protótipo:

1. Montagem e alinhamento dos módulos de captura, iluminação e disparo em uma bancada de testes. A bancada pode ser vista na figura 19, ela é composta por um suporte que rotaciona as amostras, simulando a passagem das embalagens em uma linha de produção. A velocidade de rotação é variável, permitindo avaliar o desempenho do sistema de visão em diferentes cenários de operação;

Figura 19 - Bancada de Teste



Fonte: Elaboração própria. (2026)

2. Coleta de 9 amostras de embalagens de diferentes tipos, dentre elas com fontes convencionais e de matriz de pontos (*Dot Matrix*), de forma a compor uma base de imagens experimental representativa das condições reais de aplicação. Algumas amostras utilizadas podem ser vistas na figura 20, já a amostragem completa está mais adiante no quadro 3;

Figura 20 - Amostras de Teste

Fonte: Elaboração própria. (2026)

3. Ajuste de foco, tempo de exposição e condições de iluminação, visando maximizar o contraste entre o fundo e os caracteres impressos;

4. Processamento de imagem, segmentando a região de interesse, seguido da aplicação de filtros de suavização para remoção de ruídos, ajuste e calibração dos parâmetros de limiarização e uso das operações morfológicas para completar ou corrigir caracteres quebrados, com base em técnicas de processamento digital de imagens descritas por Gonzalez e Woods (2010) e também Pedrini e Schwartz (2008).

5. Análise comparativa entre algoritmos de OCR, variando parâmetros de pré-processamento, com o objetivo de avaliar acurácia, tempo de processamento e robustez frente às diferentes condições de impressão das amostras. Para o cálculo da acurácia utilizasse como a métrica, a

taxa de erro de caractere (*Character Error Rate - CER*), ela baseia-se na Distância de Levenshtein, que quantifica o número mínimo de operações de edição, substituições *S*, deleções *D* e inserções *I,* necessárias para transformar o texto lido na referência. A fórmula de *CER* é definida pela equação 2, onde *N* representa o número total de caracteres no texto de referência.

$$CER = \frac{S+D+I}{N} \quad (2)$$

A partir do *CER,* a acurácia do sistema é calculada pela equação 3.

$$Acurácia(\%) = (1 - CER) * 100 \quad (3)$$

É importante ressaltar que, o valor de acurácia pode apresentar um resultado negativo caso o índice de CER seja superior a 1, isso ocorre quando a soma de erros supera o total de caracteres reais *N*. Este fenômeno pode ocorrer caso o algoritmo de OCR realiza "inserções" excessivas de caracteres que não existem na amostra original. Então para fins de análise, valores de acurácia negativos são interpretados como 0%, indicando total ilegibilidade da amostra.

6. Treinamento incremental do algoritmo OCR Tesseract, com o objetivo de melhorar a acurácia da leitura obtida.

3.2 Componentes do sistema

Para o desenvolvimento deste trabalho foi utilizada uma câmera USB SMTKEY 2K, equipada com sensor CMOS F5283 e resolução de 4 MP, capaz de adquirir imagens coloridas com interface UVC via USB, o que simplifica a integração com o sistema de processamento. A câmera foi acoplada a uma lente varifocal (2,8–12 mm), permitindo ajustar o campo de visão e o enquadramento de acordo com o tipo de embalagem e a distância de trabalho.

Para a iluminação da cena foi adotado um painel LED difuso branco, garantindo uma luminosidade adequada e podendo operar tanto em configuração de iluminação frontal quanto traseira. Essa flexibilidade permite otimizar o contraste entre fundo e caracteres impressos, de acordo com o material e a geometria da embalagem.

A sincronização da captura de imagens com a passagem das embalagens, foi feita com o sensor fotoelétrico E18-D80NK, do tipo difuso, com saída PNP e faixa de detecção ajustável entre 3 e 80 cm. Esse modelo foi escolhido por combinar ajuste de distância, saída digital e montagem simplificada, já que emissor e receptor se encontram no mesmo encapsulamento reduzindo a quantidade de componentes.

O processamento foi implementado em uma Raspberry Pi 5, sendo que, essa escolha deve-se à disponibilidade de GPIOs para leitura do disparo e acionamento de sinais digitais, suporte a USB para a câmera e a possibilidade de executar Linux, facilitando o desenvolvimento em Python e a integração com bibliotecas. Além disso, também foi desenvolvido uma placa que faz interface do GPIO da Raspberry com o sensor utilizado, adequando os níveis de tensão para o padrão industrial, este desenvolvimento será visto mais adiante no capitulo 3.3.1.

O software de processamento de imagens foi desenvolvido em Python, utilizando a biblioteca OpenCV para pré-processamento (conversão para escala de cinza, filtragem, limiarização e operações morfológicas) e manipulação das imagens capturadas. Já para o reconhecimento de caracteres, foram utilizadas as bibliotecas de OCR *open source*, Tesseract e EasyOCR, possibilitando a comparação de desempenho entre os algoritmos em diferentes cenários.

Alguns dos hardwares listado acima podem ser vistos no quadro 1:

Quadro 1 - Equipamentos utilizados

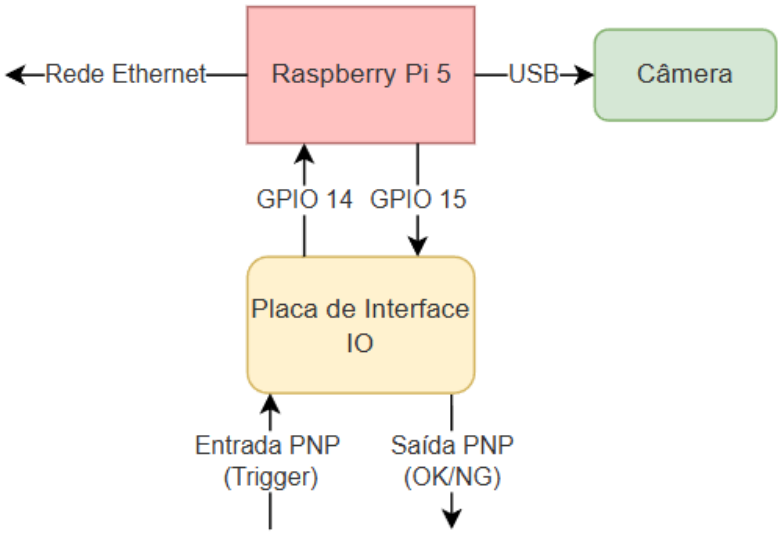
Câmera USB SMTKEY 2K	
Sensor fotoelétrico E18-D80NK	

Painel LED Difuso	
Raspberry Pi 5	

Fonte: Elaboração própria (2025)

Em resumo, a figura 21 apresenta o diagrama geral do sistema, composto pela Raspberry Pi 5, câmera, placa de interface de entrada e saída de sinal e comunicação entre os dispositivos.

Figura 21 - Diagrama do Sistema



Fonte: Elaboração própria (2025)

3.3 Desenvolvimento do Protótipo

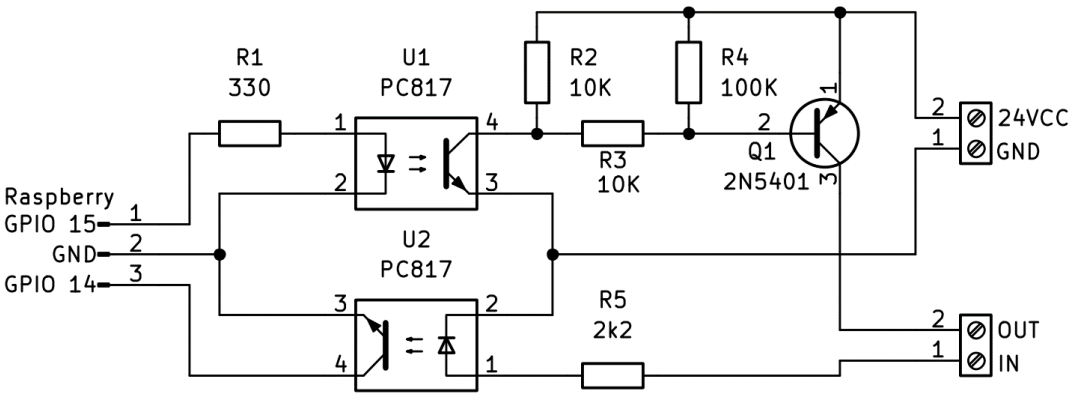
3.3.1 Placa de interface IO

De acordo com Lamb (2015), os sinais digitais comumente utilizados na indústria operam predominantemente em 24 VCC, com isso o sensor empregado neste trabalho também opera nesta faixa de tensão, seu sinal de saída é PNP 24 VCC, ou seja, quando é detectada a presença de um objeto, o sensor chaveia o sinal de saída para sua tensão de alimentação, caso não ela fica flutuando. Já o sinal de saída da inspeção também se deseja que seja PNP 24VCC para adequar o sistema ao padrão industrial, porém como as entradas e saídas digitais da Raspberry Pi trabalham na faixa de 0–3,3 V, não permitem a conexão direta com o meio.

Além disso, o ambiente industrial é sujeito a ruído elétrico e falhas, o que torna desejável o isolamento elétrico entre o controlador e o meio. Dessa forma, foi desenvolvida uma placa de interface capaz de adequar os níveis de tensão e fornecer isolamento entre a Raspberry Pi e os dispositivos de campo, aumentando a robustez e a segurança do sistema.

Sendo assim com o objetivo isolar eletricamente a Raspberry Pi e adaptar seus sinais de 3,3 V para a comutação de cargas em 24 VCC, foi desenvolvido o circuito presente na figura 22. Nela a saída do sensor de disparo é conectada em *IN* e o sinal do resultado da inspeção em *OUT,* e então é conectado aos pinos de GPIO 14 e 15 da Raspberry. Além de que a placa também deve ser alimentada com 24 VCC que é utilizado com sinal de saída.

Figura 22 - Circuito da placa de interface



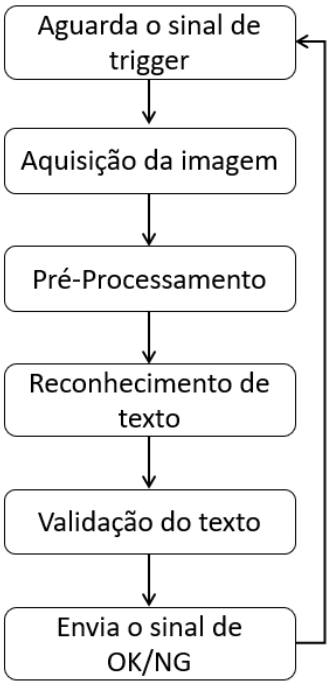
Fonte: Elaboração própria (2025)

Como pode ser visto no circuito da figura 22, o acoplamento é feito por dois optoacopladores PC817 (U1 e U2). No lado de saída, o transistor PNP 2N5401 (Q1), polarizado pelos resistores R2, R3, atua como estágio de potência, chaveando a alimentação de 24 VCC para o borne de saída, este fornecendo até cerca de 50mA. Já para a entrada, o resistor R5 limita a corrente de entrada no lado de campo, e aciona o optoacoplador conectado a entrada da Raspberry, também deve ser ressaltado que para o uso deste circuito é considerado o uso de *pullup* interno do GPIO 14 da Raspberry.

3.3.2 Implementação do processamento de imagens e OCR

O *pipeline* de processamento de imagens e OCR foi estruturado em etapas sequenciais, seguindo o fluxo clássico descrito por Konovalchuk (2024), como mostra o diagrama presente na figura 23.

Figura 23 - Diagrama do funcionamento do algoritmo



Fonte: Elaboração própria (2025)

A implementação do sistema começa com a captura da imagem dado um sinal de *trigger*, seguido pelo pré-processamento (conversão para tons de cinza, suavização, binarização e operações morfológicas) e então é aplicado o mecanismo

de OCR. A validação do texto é feita através de uma comparação direta dos caracteres do texto extraído do alvo com o texto de referência (código de deveria estar gravado na embalagem), caso houver divergência entre os textos, é enviado um pulso no pino de saída *OUT*, desse modo, esse sinal pode ser usado para a separação dessa embalagem que não possui o código de data e lote gravado corretamente, por exemplo.

Também deve ser ressaltado que foi medido o tempo total de execução do algortmo, para desse modo, poder avaliar a performance de cada configuração de OCR utilizada. O código completo do programa desenvolvido se encontra no apêndice A.

3.3.2.1 Aquisição sincronizada da imagem

A captura é disparada por um sinal de *trigger* no GPIO 14, detectado via interrupção de borda de descida com *debounce* de 50 ms. Após o *trigger*, devido ao *buffer* da câmera, os quatro primeiros frames são descartados para sincronizar com o momento exato da passagem da embalagem, evitando imagens borradas ou desalinhadas. Pelo driver da câmera é configurada a resolução 1280x720, ganho 60, exposição manual e balanceamento de branco desativado.

3.3.2.2 Pré-processamento da imagem

O pré-processamento é feito seguindo as seguintes etapas:

1. A região onde estão presentes a data e o lote da embalagem são recortados da imagem da câmera, esse ponto de interesse é carregado caso já estiver salvo ou pode ser definido pela janela de visualização da câmera;
2. Conversão para escala de cinza para reduzir complexidade computacional.
3. Filtro de mediana para remoção do ruído da imagem, preservando bordas melhor que o filtro de média.
4. Binarização adaptativa utilizando o método de Otsu para limiarização global automática, robusto a variações de iluminação.

5. Erosão morfológica, para afinar caracteres e remover artefatos pequenos, utilizando elemento estruturante retangular otimizado para texto impresso.

No quadro 2 temos o pré-processamento realizado em cada amostra, em geral foi utilizado combinações de diferentes filtros, devido cada embalagem possuir características diferentes. Já no caso das amostras 7, 8 e 9 foram utilizados o mesmo filtro devido serem o mesmo produto e dessa forma será possível avaliar a robustez diante de uma amostragem maior. Os filtros foram aplicados seguindo a sequência apresentada no quadro, é importante destacar que em todos os casos foi utilizado um *kernel* retangular anotado ao lado do respectivo filtro, ou seja, para Erosão (4,7), foi utilizado um *kernel* com tamanho 4 x 7. As amostras bem como o resultado do pré-processamento serão apresentadas no próximo capitulo.

Quadro 2 - Processamento utilizado

Amostra	Pré-processamento utilizado
1	Binarização (Otsu) → Fechamento (4,4)
2	Mediana (3) → Binarização (Otsu) → Erosão (2,2)
3	Mediana (3) → Binarização (Otsu) → Erosão (2,3)
4	Mediana (3) → Binarização (Otsu) → Fechamento (3,3) → Erosão (2,5)
5	Binarização (Otsu) → Fechamento (2,2)
6	Mediana (3) → Binarização (Otsu) → Fechamento (3,3) → Erosão (4,7)
7, 8 e 9	Mediana (9) → Binarização (Otsu) → Abertura (5,8) → Fechamento(3,3) → Erosão (3,3)

Fonte: Elaboração própria (2026)

3.3.2.3 Treinamento incremental do Tesseract

Para superar as limitações identificadas no reconhecimento de fontes matriciais com alta descontinuidade de pontos, implementou-se uma etapa de treinamento incremental para o motor neural do Tesseract, permitindo ajustar os

pesos da rede neural do algoritmo para que ele aprenda os padrões específicos de uma nova tipografia industrial, mantendo o conhecimento prévio do idioma base.

O treinamento foi executado utilizando a ferramenta *tesstrain*, um ambiente de automação baseado em *Makefile* que padroniza o *pipeline* de treinamento para as versões mais recentes do Tesseract. O procedimento experimental para a geração do modelo customizado seguiu as etapas abaixo:

1. **Preparação do Dataset:** Foram selecionadas as imagens das amostras 7, 8 e 9, que representavam os cenários de maior dificuldade de leitura. Estas imagens foram submetidas à *pipeline* de pré-processamento descrita na seção 3.3.2.2, garantindo que a rede neural recebesse dados normalizados e binarizados.
2. **Geração do *Ground Truth*:** Para cada imagem de treino, foi criado um arquivo de texto correspondente contendo a transcrição exata e correta dos caracteres presentes na cena. Este conjunto de pares (imagem/texto) serviu como a verdade conhecida para o cálculo do erro e ajuste dos pesos durante o aprendizado.
3. **Configuração e Parâmetros:** Utilizou-se como modelo base o arquivo *por.traineddata* (Português). O treinamento foi configurado para o modo incremental, com um limite máximo de 8.000 iterações. Esse volume de iterações foi definido para garantir a convergência do erro para níveis mínimos, permitindo que o modelo "aprendesse" a forma dos caracteres matriciais sem causar que comprometeria a leitura de outras fontes convencionais.

As demais configurações, como taxa de aprendizado e especificações da rede, foram mantidas nos valores padrão da ferramenta. Após o treinamento, o novo arquivo de dados treinado foi integrado ao sistema de visão, sendo carregado via parâmetro na biblioteca *pytesseract* para as fases de teste final e validação.

4 APRESENTAÇÃO DOS RESULTADOS

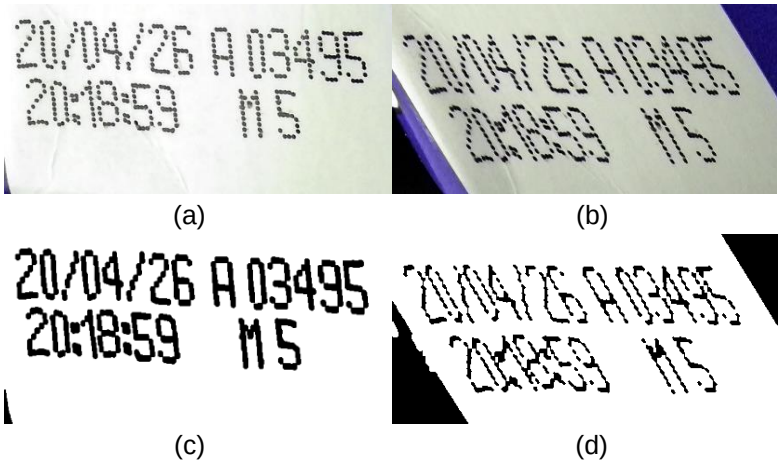
Neste capítulo, procede-se à análise detalhada dos resultados obtidos do protótipo de visão computacional desenvolvido para a inspeção de códigos de data e lote. A apresentação dos resultados está estruturada para avaliar de forma isolada e correlacionada a eficácia das técnicas de processamento digital de imagens e o desempenho dos algoritmos de reconhecimento óptico de caracteres (OCR) em hardware embarcado.

4.1 Limitações e efeito *rolling shutter* na imagem

Primeiramente, um fator crítico que influenciou o sistema de visão computacional, foi o sensor da câmera utilizada. Conforme descrito na metodologia, o protótipo utiliza uma câmera USB SMTKEY 2K equipada com um sensor CMOS *rolling shutter*, que realiza a varredura da cena de forma sequencial, linha por linha, para compor o quadro final da imagem digital.

Durante os experimentos com as amostras alta velocidade na bancada de testes, observou-se que essa característica gera distorções significativas nos caracteres. Como o objeto se desloca enquanto o sensor ainda está completando a leitura das linhas inferiores, a imagem resultante apresenta uma inclinação lateral ou alongamento, como apresentado na figura 24.

Figura 24 - (a) Aquisição em baixa velocidade. (b) Aquisição em alta velocidade. (c) Imagem (a) pré-processada. (d) Imagem (b) pré-processada.



(a) (b)

(c) (d)

Fonte: Elaboração própria (2026)

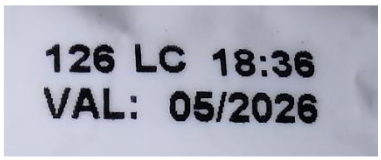
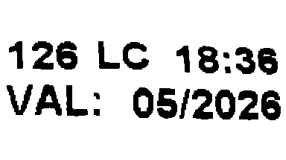
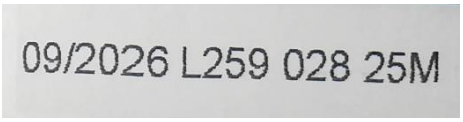
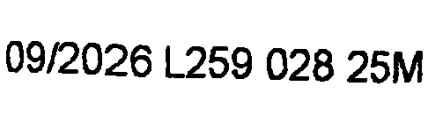
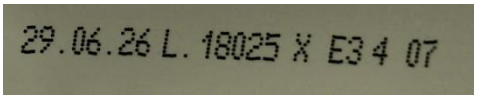
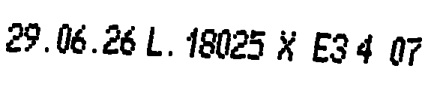
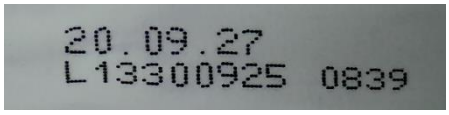
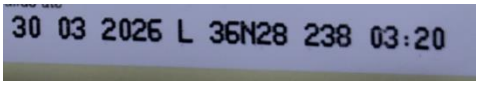
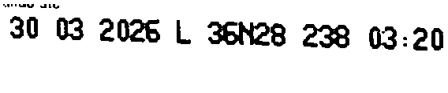
Como pode ser visto na figura 24, temos as aquisições das imagens em diferentes velocidades, em (a) e (c) o objeto estava em uma velocidade linear de aproximadamente 0,08 m/s e em (b) e (d) o objeto está à 0,75 m/s.

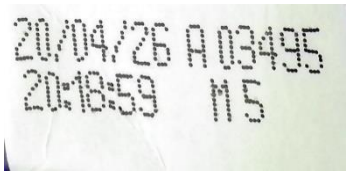
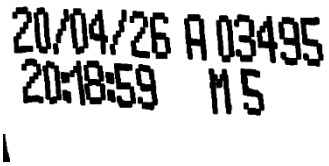
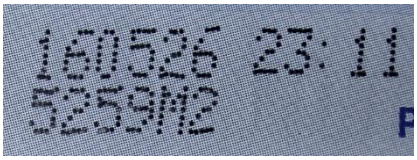
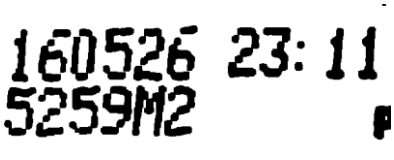
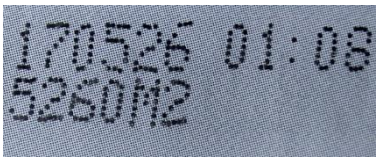
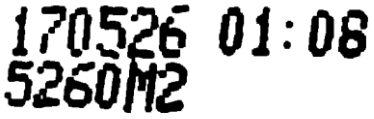
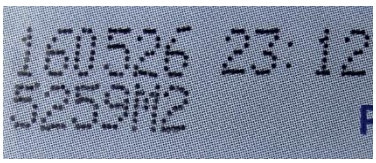
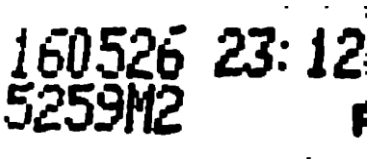
Devido a essas distorções em uma velocidade mais alta, as imagens capturadas nessas condições não foram aproveitadas para as métricas de assertividade do OCR apresentadas nos próximos resultados. A inclusão desses dados resultaria em índices de similaridade nulos, não por falha do pré-processamento de imagem ou do algoritmo de reconhecimento em si, mas por uma limitação física do hardware de aquisição.

4.2 Resultados do pré-processamento

O sucesso de um sistema de visão industrial depende diretamente da qualidade da imagem fornecida ao motor de processamento. O Quadro 3 ilustra visualmente a transformação das nove amostras capturadas após a aplicação da *pipeline* de filtros apresentada no quadro 2.

Quadro 3 - Resultado do pré-processamento

Amostra	Imagem Capturada	Imagem Pré-processada
1		
2		
3		
4		
5		

6		
7		
8		
9		

Fonte: Elaboração própria (2026)

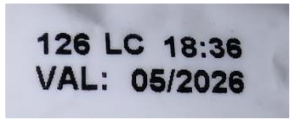
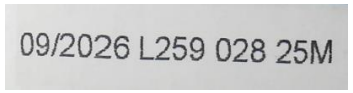
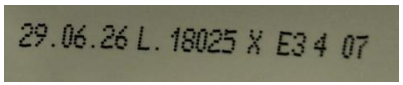
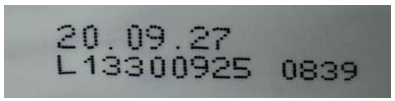
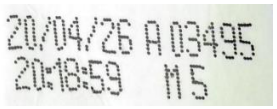
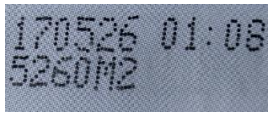
As amostras abrangem desde fontes convencionais sólidas (amostras 1 e 2) até fontes matriciais (*dot-matrix*) mais complexas (amostras 3 a 9). Observa-se no quadro 3 que as imagens originais apresentam variações de contraste e descontinuidades nos caracteres, especialmente nas amostras matriciais, que são normalizadas nas imagens pré-processadas. Essa etapa é crucial para remover ruídos e isolar o texto do plano de fundo, seguindo os princípios de segmentação descritos por Gonzalez e Woods (2010).

A aplicação do filtro de mediana nas amostras foi essencial para a remoção de ruídos impulsivos preservando as bordas dos caracteres. Operações morfológicas como o fechamento e a erosão foram empregadas para conectar glifos quebrados em fontes matriciais e ajustar a espessura dos traços para o OCR, permitindo a leitura das amostras com fonte *Dot Matrix*.

4.3 Análise dos resultados obtidos através dos algoritmos de OCR

A análise do quadro 4 demonstra a resultado e acurácia obtida dos algoritmos de OCR, quando são utilizadas as imagens brutas sem tratamento prévio.

Quadro 4 - Resultado do OCR (sem pré-processamento)

Amostra		Texto Escrito	Tesseract	%	EasyOCR	%
1		126 LC 18:36 VAL: 05/2026	126 LC 18:36 VAL: 05/2026	100	126 LC 18:36 VAL: 05/2026	100
2		09/2026 L259 028 25M	09/2026 L259 028 25M	100	09/2026 L259 028 25M	100
3		29.06.26 L. 18025 X E34 07	(NULL)	0	29.06.26 L. 18025 X E34 07	100
4		20.09.27 L13300925 0839	0. 09.27 LIZ300925 0839	82	2009.27 L13300925 0839	95
5		30 03 2026 L 36N28 238 03:20	30 03 2026 L 36N28 238 03:20	100	30 03 2026 L 36N28 238 03:20	100
6		20/04/26 A 03495 20:18:59 M5	MA A 3 A MM	4	20/04/26 A 03495 2018.59 M5	92
7		160526 23:11 5259M2	2.	0	160526 23.41 5259M2	89
8		170526 01:08 5260M2	(NULL)	0	170526 01:08 5260M2	100
9		160526 23:12 5259M2	(NULL)	0	60526 23 U 5 9	53

Fonte: Elaboração própria (2026)

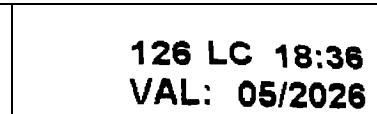
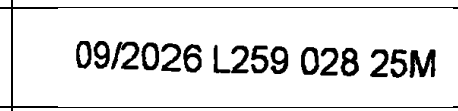
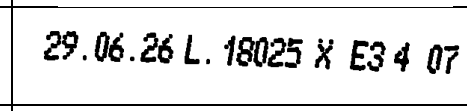
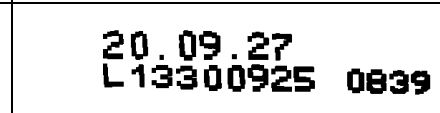
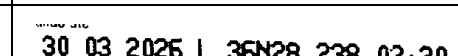
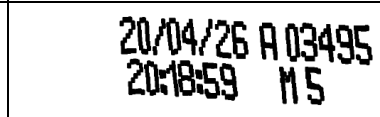
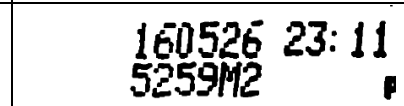
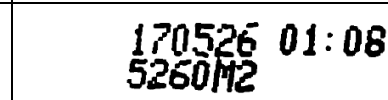
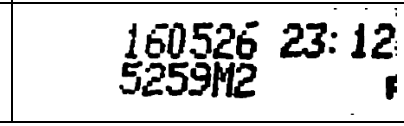
O algoritmo Tesseract apresentou falha crítica em grande parte das amostras, resultando em saídas nulas (*null*). Por outro lado, o EasyOCR mostrou-se significativamente mais robusto em condições adversas, mantendo índices de assertividade elevados em quase todas as amostras convencionais e matriciais, com destaque para a Amostra 8, onde obteve 100% contra 0% do Tesseract. Esse comportamento do EasyOCR justifica-se por sua arquitetura baseada em *deep learning*, projetada para lidar com fontes diversas em condições não controladas.

Com a aplicação da pipeline de filtros descrita no Quadro 2, o desempenho dos algoritmos foi reavaliado e apresentado no Quadro 5 na próxima página, revelando uma evolução notável na precisão. Avaliando agora os resultados obtidos com o tratamento da imagem, o Tesseract foi o maior beneficiado pelo pré-processamento. Amostras que anteriormente eram ilegíveis (3, 4 e 7) passaram a apresentar 100% de assertividade. Isso confirma que o Tesseract requer imagens mais limpas e caracteres sólidos para uma segmentação de linha e caractere eficiente.

Já para o EasyOCR, embora tenha mantido altos índices, observou-se uma queda de assertividade em amostras matriciais pré-processadas (Amostras 4, 7 e 9) em comparação à imagem bruta. Na Amostra 4, por exemplo, a precisão caiu de 95% para 56%. Isso sugere que a binarização agressiva pode remover características neurais que o EasyOCR utiliza nativamente para o reconhecimento do texto.

Contudo nas amostras 7, 8 e 9, que são o mesmo produto produzido na mesma linha em momentos diferentes, notou-se uma inconsistência na leitura obtida. A suspeita desse fato é que como houve uma variação na gravação da data e do lote, tenha prejudicado a leitura, visto que, o pré-processamento foi parametrizado de acordo com a amostra 7, adicionalmente foram testados com outros parâmetros para melhorar a leitura das amostras 8 e 9, porém não foi possível alcançar melhores resultados.

Quadro 5 - Resultado do OCR (com pré-processamento)

Amostra		Texto Escrito	Tesseract	%	EasyOCR	%
1		126 LC 18:36 VAL: 05/2026	126 LC 18:36 VAL: 05/2026	100	126 LC 18:36 VAL: 05/2026	100
2		09/2026 L259 028 25M	09/2026 L259 028 25M	100	09/2026 L259 028 25M	100
3		29.06.26 L. 18025 X E34 07	29.06.26 L. 18025 X E34 07	100	29.06.26 L. 18025 X E34 07	100
4		20.09.27 L13300925 0839	20.09.27 L13300925 0839	100	27 293380325 0839	56
5		30 03 2026 L 36N28 238 03:20	30 03 2026 L 36N28 238 03:20	100	30 03 2026 L 36N28 238 03:20	100
6		20/04/26 A 03495 20:18:59 M5	20/04/26 A 03495 M5	68	2004/26 A 03495 201859 M5	86
7		160526 23:11 5259M2	160526 23:11 5259M2	100	23: 11 J29532	42
8		170526 01:08 5260M2	0526 01:08	47	SZ88P2 01: 08	32
9		160526 23:12 5259M2	160526 23:11 5259M2	84	360572 23:12	47

Fonte: Elaboração própria (2026)

4.4 Análise dos tempos de processamento de cada algoritmo

A tabela 1 apresenta o tempo total de processamento necessário para cada execução na plataforma Raspberry Pi 5.

Tabela 1 - Tempo de Processamento

Amostra	Sem pré-processamento (s)		Com pré-processamento (s)	
	Tesseract	**EasyOCR**	**Tesseract**	**EasyOCR**
1	0.2092	1.8423	0.1475	1.6187
2	0.1781	1.1921	0.1382	1.1955
3	0.2120	1.0804	0.1373	1.0746
4	0.2069	1.4429	0.1412	1.2915
5	0.2317	0.9247	0.1527	0.9138
6	0.4287	2.3978	0.1557	2.3703
7	0.2870	2.0957	0.1473	1.9751
8	0.3174	2.0619	0.1446	2.0077
9	0.2648	2.0441	0.1629	1.9262

Fonte: Elaboração própria (2026)

O Tesseract provou ser drasticamente mais rápido que o EasyOCR em todos os cenários. Em imagens processadas, o tempo médio do Tesseract ficou em torno de 137 ms a 163 ms, enquanto o EasyOCR demandou entre 913 ms e 2,37 s. Esta discrepância decorre do fato de a Raspberry Pi operar o processamento exclusivamente via CPU. O EasyOCR, baseado em arquiteturas de Redes Neurais Convolucionais (CNN), exige aceleração por GPU para máxima eficiência. Em contrapartida, a arquitetura LSTM do Tesseract mostrou-se altamente compatível com o hardware embarcado.

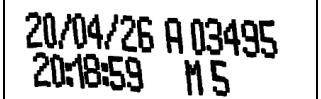
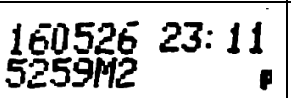
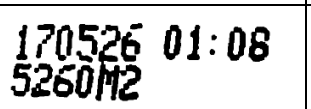
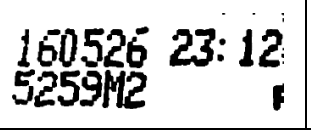
Adicionalmente, nota-se que mesmo com o tempo de execução do pré-processamento, a velocidade total de execução com o Tesseract é otimizada em até 2 vezes, devido a redução da complexidade de segmentação do algoritmo.

4.5 Resultados do treinamento incremental do Tesseract

Apesar da eficácia das técnicas de pré-processamento morfológico apresentadas nas seções anteriores, observou-se que amostras com fontes matriciais complexas e baixa densidade de pontos, como as amostras 6, 7, 8 e 9, mantiveram índices de assertividade inconsistentes. Conforme discutido na fundamentação teórica, o Tesseract utiliza uma arquitetura baseada em Redes Neurais Recorrentes (LSTM), que permite o ajuste fino para padrões de fontes específicos que não constam no conjunto de treinamento original.

Dessa forma, realizou-se um treinamento incremental focado na tipografia das amostras críticas, visando melhorar a capacidade de reconhecimento de caractere do algoritmo. Lembrando que foram utilizadas as amostras já pré-processadas. O quadro 6 apresenta os resultados alcançados após o treinamento.

Quadro 6 - Resultado do OCR (treino incremental)

Amostra		Texto Escrito	Texto Extraído	%	Tempo (ms)
6		20/04/26 A 03495 20:18:59 M5	20/04/26 A 03495 20:18:59 M5	100	181
7		160526 23:11 5259M2	160526 23:11 5259M2	100	180
8		170526 01:08 5260M2	170526 01:08 5260M2	100	172
9		160526 23:11 5259M2	160526 23:12 5255M2	88	180

Fonte: Elaboração própria (2026)

Os resultados obtidos demonstraram uma melhora expressiva na acurácia da leitura, a amostra 7 consolidou sua precisão, a 8 elevou de 47% para 100% de similaridade, enquanto a 9 apresentou 88%. Esse salto demonstra que a rede neural passou a identificar corretamente os glifos matriciais que anteriormente eram fragmentados ou confundidos pelo algoritmo.

Já em relação aos tempos de processamento, mantiveram-se na casa dos 0,18 s, o que é coerente com os resultados anteriores registrados na Tabela 1. Isso confirma que o treinamento incremental não aumentou significativamente a carga computacional da CPU na Raspberry Pi 5, a ponto de inviabilizar o novo modelo treinado.

5 CONSIDERAÇÕES FINAIS

O presente trabalho propôs o desenvolvimento e a validação de um protótipo de sistema de visão computacional de baixo custo para a automação da inspeção de códigos de data e lote em embalagens industriais. A pesquisa foi motivada pela necessidade crítica de conformidade com a Resolução RDC nº 259/2002 da ANVISA, que exige a legibilidade de informações essenciais para a rastreabilidade e segurança do consumidor.

Ao longo do desenvolvimento, demonstrou-se que a integração de hardware embarcado acessível, como a plataforma Raspberry Pi 5, com algoritmos de reconhecimento óptico de caracteres, é uma solução tecnicamente viável e capaz de operar em tempos de resposta compatíveis com processos industriais.

A análise comparativa entre os motores Tesseract e EasyOCR revelou que, para o hardware utilizado, o Tesseract apresenta uma eficiência superior. Embora o EasyOCR tenha demonstrado maior robustez nativa em imagens sem tratamento, o Tesseract, quando auxiliado por uma *pipeline* de pré-processamento adequada e treinamento incremental, atingiu índices de assertividade de até 100% em fontes matriciais complexas, que inicialmente eram ilegíveis. Também vale a pena ressaltar que, à medida que mais amostras sejam utilizadas no treinamento, maior a robustez do modelo gerado.

As técnicas de processamento digital de imagens, mostraram-se indispensáveis. O uso de filtros morfológicos para conectar os pontos da tipografia *dot-matrix* e a limiarização de *Otsu* para isolamento do texto permitiram a extração do texto das imagens. Por fim, a identificação das limitações impostas pelo sensor *rolling shutter* em altas velocidades de esteira foi essencial para o refinamento do hardware em futuras implementações.

No geral, o protótipo valida a hipótese de que é possível automatizar pontos críticos do controle de qualidade industrial com ferramentas de software livre e hardware comercial, fortalecendo a confiabilidade operacional e a rastreabilidade na indústria 4.0.

5.1 Sugestões para trabalhos futuros

Apesar dos resultados promissores alcançados, o sistema de visão desenvolvido possui margem para evoluções técnicas e funcionais que podem ampliar sua aplicabilidade e robustez. Para trabalhos futuros, sugerem-se alguns pontos de aprimoramento.

Embora o protótipo utilize uma placa de interface de I/O isolada para sinais de *trigger* e resultados binários (Aprovado/Reprovado), sugere-se a implementação de protocolos de comunicação em rede. Isso permitiria que o sistema de visão não apenas sinalizasse falhas, mas também enviasse o texto lido para sistemas de supervisão ou controladores lógicos programáveis (CLP), permitindo a criação de registros digitais de lote em tempo real e maior controle estatístico do processo.

Outro ponto também relevante para consolidar como um produto pronto para o usuário final, é a criação de uma interface gráfica do usuário intuitiva, que facilitaria a calibração do sistema por operadores de linha sem conhecimento técnico em programação. Embora a seleção da região de interesse que está o texto alvo seja definida pela janela de visualização da imagem, os ajustes dos parâmetros de filtragem e treinamento são realizados de forma manual no código fonte deste trabalho.

Para mitigar a distorção por movimento identificada nos testes, recomenda-se a substituição da câmera atual por modelos equipados com sensores *global shutter*. Esta mudança eliminaria a distorção nos caracteres, permitindo que a acurácia de 100% observada em capturas estáticas seja mantida mesmo em linhas de produção de altíssima cadência. Somada ao aprimoramento do treino, esta evolução técnica consolidaria o sistema como uma solução de alta performance para a inspeção industrial automatizada.

REFERÊNCIAS

AWS. **What is OCR? - Optical Character Recognition Explained**. Disponível em: https://aws.amazon.com/what-is/ocr/. Acesso em: 29 out. 2025.

BAUMER. **Rolling shutter, global shutter** – two principles of exposure. Disponível em: https://www.baumer.com/int/en/service-support/technical-information-industrial-cameras/rolling-shutter-global-shutter-two-principles-of-exposure-/a/rolling-shutter-global-shutter. Acesso em: 7 fev. 2026.

BENOIT, Brian. **Uma Abordagem mais Fácil para Automatizar o OCR de Embalagem em Linha a Velocidades de Linha**. Cognex, 2023. Disponível em: https://www.cognex.com/pt-br/blogs/deep-learning/an-easier-approach-for-automating-inline-packaging-ocr-at-line-speeds. Acesso em: 29 out. 2025.

BRASIL. **Lei nº 6.437, de 20 de agosto de 1977**. Configura infrações à legislação sanitária federal e estabelece as sanções respectivas. 24 ago. 1977. Disponível em: https://www.planalto.gov.br/ccivil_03/leis/l6437.htm. Acesso em: 15 nov. 2025.

BRASIL. Agência Nacional de Vigilância Sanitária. **Resolução RDC nº 259, de 20 de setembro de 2002**. Aprova o Regulamento Técnico sobre Rotulagem de Alimentos Embalados. 23 set. 2002. Disponível em: https://bvsms.saude.gov.br/bvs/saudelegis/anvisa/2002/rdc0259_20_09_2002.html. Acesso em: 15 nov. 2025.

FRANCHI, C. M; CAMARGO, V. L. A. **Controladores Lógicos Programáveis**: Sistemas discretos. 1.ed. São Paulo: Érica, 2008.

GONZALEZ, R. C.; WOODS, R. E**. Processamento Digital de Imagens**. 3. ed. São Paulo: Pearson Prentice Hall, 2010.

IBM. **O que é reconhecimento óptico de caracteres (OCR)?**. Disponível em: https://www.ibm.com/br-pt/think/topics/optical-character-recognition. Acesso em: 29 out. 2025.

JAIDEDAI. **EasyOCR**. GitHub, 2020. Disponível em: https://github.com/JaidedAI/EasyOCR. Acesso em: 16 nov. 2025.

KONOVALCHUK, N. **OCR Algorithms:** Types, Use Cases and Best Solutions. Itransition Computer Vision, 2024. Disponível em: https://www.itransition.com/computer-vision/ocr-algorithm. Acesso em: 29 out. 2025.

Lamb, F. **Automação industrial na prática**. Porto Alegre: AMGH, 2015.

LI, Ning. **An Implementation of OCR System Based on Skeleton Matching.** Computing Laboratory University of Kent at Canterbury, Reino Unido, 1991. Disponível em: https://kar.kent.ac.uk/21129/1/OCRNing.pdf. Acesso em 1 nov. 2025.

MAHAJAN, A. **EasyOCR:** A Comprehensive Guide. Medium, 2023. Disponível em: https://medium.com/@adityamahajan.work/easyocr-a-comprehensive-guide-5ff1cb850168. Acesso em: 16 nov. 2025.

Pedrini, H; Schwartz, W. R. **Análise de imagens digitais:** princípios, algoritmos e aplicações. São Paulo: Cengage Learning, 2008.

PETITCLERC, G. **A Comprehensive Guide to Optical Character Recognition (OCR)**. Moov.A, 2019. Disponível em: https://moov.ai/en/blog/optical-character-recognition-ocr. Acesso em: 29 out. 2025.

SMITH, R. **An Overview of the Tesseract OCR Engine**. Google Inc., 2007. Disponível em: https://tesseract-ocr.github.io/docs/tesseracticdar2007.pdf. Acesso em: 16 nov. 2025.

SZELISKI, R. **Computer vision:** Algorithms and applications. 2. ed. Springer, 2022.

WEST, P. **Fundamentals of Machine Vision**. San Jose: AUTOMATED VISION SYSTEMS, INC, 2021. Disponível em: https://www.autovis.com/images/pdf/resources/fundamentals-of-machine-vision.pdf. Acesso em: 22 out. 2025.

TESSERACT OCR. **Tesseract Open Source OCR Engine**. Disponível em: https://github.com/tesseract-ocr/tesseract. Acesso em: 16 nov. 2025

TESSERACT OCR. **How to train LSTM/neural net Tesseract**. Disponível em: https://tesseract-ocr.github.io/tessdoc/tess5/TrainingTesseract-5.html. Acesso em: 07 fev. 2026

APÊNDICES

APÊNDICE A – Código Completo do Programa

```
import cv2
import os
import time
import subprocess
import RPi.GPIO as GPIO
import pytesseract
import json
import logging

# ========= CONFIGURAÇÕES =========
PIN_PULSO = 14  # Entrada
PIN_SAIDA = 15  # Saída
DEBOUNCE_S = 0.05  # 50 ms
TEXTO_ESPERADO = "160526 23:11\n5259M2"

DEVICE_ID = 0                        # ID da Camera
SAVE_DIR = "./imagens"               # Caminho para salvar a imagem da camera
ROI_CONFIG_FILE = "roi_config.json"  # Configuracao do ROI

contador_pulsos = 0
ultimo_pulso = 0
pulso_recebido = False

roi_points = []
drawing = False

# LOG
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s [%(levelname)s] %(message)s',
    handlers=[
        logging.FileHandler("execucao_visao.log"),
        logging.StreamHandler()
    ]
)

# ========= FUNÇÃO DE ROI =========
def selecionar_roi(event, x, y, flags, param):
    global roi_points, drawing

    if event == cv2.EVENT_LBUTTONDOWN:
        roi_points = [(x, y)]
        drawing = True

    elif event == cv2.EVENT_MOUSEMOVE and drawing:
        roi_points = [roi_points[0], (x, y)]
```

```
    elif event == cv2.EVENT_LBUTTONUP:
        if drawing:
            roi_points = [roi_points[0], (x, y)]
            drawing = False

            try:
                with open(ROI_CONFIG_FILE, 'w') as f:
                    json.dump({'roi': roi_points}, f)
                logging.info(f"ROI salva: {roi_points}")
            except Exception as e:
                logging.error(f"Erro ao salvar JSON da ROI: {e}")


# ========= CONFIGURAÇÃO DA CÂMERA =========
def configurar_camera(device=0):
    cmds = [
        f"v4l2-ctl -d /dev/video{device} -c white_balance_automatic=0",
        f"v4l2-ctl -d /dev/video{device} -c auto_exposure=1",
        f"v4l2-ctl -d /dev/video{device} -c exposure_time_absolute=0",
        f"v4l2-ctl -d /dev/video{device} -c gain=60",
        f"v4l2-ctl -d /dev/video{device} -c brightness=128",
        f"v4l2-ctl -d /dev/video{device} -c contrast=128",
        f"v4l2-ctl -d /dev/video{device} --set-fmt-
video=width=1280,height=720,pixelformat=NV12"
    ]

    for cmd in cmds:
        subprocess.run(cmd, shell=True, check=False)

def inicializar_camera(device=0):
    cap = cv2.VideoCapture(device)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
    if not cap.isOpened():
        logging.error("Câmera não encontrada.")
        return None
    return cap

# ========= PROCESSAMENTO E OCR =========
def processar_e_salvar(frame, contador):
    start_time = time.time()
    # Recorte da ROI
    img_ocr = frame
    if len(roi_points) == 2:
        x1, y1 = roi_points[0]
        x2, y2 = roi_points[1]
```

```
        x_start, x_end = sorted([x1, x2])
        y_start, y_end = sorted([y1, y2])

        if x_end > x_start and y_end > y_start:
            img_ocr = frame[y_start:y_end, x_start:x_end]

    # Pré-processamento da Imagem
    grayscale = cv2.cvtColor(img_ocr, cv2.COLOR_BGR2GRAY)
    median = cv2.medianBlur(grayscale, 7)
    _, binary = cv2.threshold(median, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
    opened = cv2.morphologyEx(binary, cv2.MORPH_OPEN,
cv2.getStructuringElement(cv2.MORPH_RECT, (4, 8)))
    closed = cv2.morphologyEx(opened, cv2.MORPH_CLOSE,
cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3)))
    eroded = cv2.erode(closed, cv2.getStructuringElement(cv2.MORPH_RECT, (3,
3)))

    # Configuração OCR (Tesseract)
    config_tess = '--oem 3 --psm 6 -c tessedit_char_whitelist="0123456789M:"'
    texto = pytesseract.image_to_string(eroded, lang='DotMatrixOCR',
config=config_tess)

    cv2.imshow("Debug OCR", eroded)
    texto_limpo = texto.strip()
    tempo_proc = time.time() - start_time

    # Salva Log
    logging.info(f"[Trigger #{contador}] Temp: ({tempo_proc:.3f}s)")
    #cv2.imwrite(os.path.join(SAVE_DIR, f"trigger_{contador:04d}.png"),
eroded)

    return texto_limpo

# ========= INTERRUPÇÃO GPIO =========
def callback_pulso(channel):
    global contador_pulsos, ultimo_pulso, pulso_recebido
    agora = time.monotonic()
    # Debounce
    if (agora - ultimo_pulso) > DEBOUNCE_S:
        contador_pulsos += 1
        ultimo_pulso = agora
        pulso_recebido = True

# ========= MAIN =========
def main():
    global pulso_recebido, roi_points
```

```
# Setup
configurar_camera(DEVICE_ID)
cap = inicializar_camera(DEVICE_ID)
if not cap: return

# GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(PIN_PULSO, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(PIN_SAIDA, GPIO.OUT)
GPIO.output(PIN_SAIDA, GPIO.LOW) # Estado inicial
GPIO.add_event_detect(PIN_PULSO, GPIO.FALLING, callback=callback_pulso)

# Carrega ROI salva
if os.path.exists(ROI_CONFIG_FILE):
    try:
        with open(ROI_CONFIG_FILE, 'r') as f:
            dados = json.load(f)
            roi_points = [tuple(p) for p in dados.get('roi', [])]
        logging.info(f"ROI carregada: {roi_points}")
    except Exception as e:
        logging.warning(f"Aviso: ROI inválida ou corrompida ({e})")

# Interface
window_name = "Configurar ROI"
cv2.namedWindow(window_name)
cv2.setMouseCallback(window_name, selecionar_roi)

ret, frame = cap.read()
if not ret:
    logging.error("Erro: Não foi possível obter o frame inicial.")
    return

try:
    while True:
        key = cv2.waitKey(1) & 0xFF

        if pulso_recebido:
            pulso_recebido = False

            # Limpa o buffer da câmera
            for _ in range(5):
                cap.grab()
            ret_trig, frame_trig = cap.retrieve()

            if ret_trig:
                frame = frame_trig # Atualiza o frame atual para exibição
                texto_lido = processar_e_salvar(frame, contador_pulsos)
```

```
                    # Validação do texto
                    linhas_lidas = [l.strip() for l in texto_lido.split('\n')
if l.strip()]
                    linhas_esperadas = [l.strip() for l in
TEXTO_ESPERADO.split('\n') if l.strip()]

                    if linhas_lidas != linhas_esperadas:
                        logging.warning(f"NG: Texto divergente! \n Lido:
'{texto_lido}' \n Esperado: '{TEXTO_ESPERADO}'\n")

                        # Aciona a saída
                        GPIO.output(PIN_SAIDA, GPIO.HIGH)
                        time.sleep(0.1) # Pulso de 100ms
                        GPIO.output(PIN_SAIDA, GPIO.LOW)
                    else:
                        logging.info("OK: Texto validado com sucesso.\n")

            # Trigger Manual
            elif key == 13:
                for _ in range(5): cap.grab()
                ret_man, frame_man = cap.retrieve()
                if ret_man:
                    frame = frame_man
                    logging.info("Captura manual realizada.")

            # Desenho da Interface
            display_frame = frame.copy()

            if len(roi_points) == 2 and roi_points[0] != roi_points[1]:
                cv2.rectangle(display_frame, roi_points[0], roi_points[1], (0,
255, 0), 2)
                cv2.putText(display_frame, "ROI Ativa", (roi_points[0][0],
roi_points[0][1]-10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,255,0), 1)

            cv2.imshow(window_name, display_frame)
            if key == 27:
                break

    except KeyboardInterrupt:
        logging.info("Parando...")
    finally:
        cap.release()
        cv2.destroyAllWindows()
        GPIO.cleanup()
if __name__ == "__main__":
    main()
```

APÊNDICE B – Código Usado para Comparativo entre Tesseract e EasyOCR

```
import cv2
import pytesseract
import easyocr
import time


# Abre a imagem da amostra
frame = cv2.imread("imgs/1.png")

# Pré-processamento da imagem
start_time = time.time()
grayscale = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
median = cv2.medianBlur(grayscale, 9)
_, binary = cv2.threshold(median, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
opened = cv2.morphologyEx(binary, cv2.MORPH_OPEN,
cv2.getStructuringElement(cv2.MORPH_RECT, (3, 5)))
closed = cv2.morphologyEx(opened, cv2.MORPH_CLOSE,
cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3)))
eroded = cv2.erode(opened, cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3)))
time_1 = time.time() - start_time

# --- Tesseract ---
config_tess = (
    '--oem 3 --psm 6 '
    '-c tessedit_char_whitelist="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ.:/
"'
)

print(f"Tempo de pre-processamento: {time_1:.4f}s")
start_time = time.time()
text_1 = pytesseract.image_to_string(eroded, lang='DotMatrixOCR',
config=config_tess)
tess_time_1 = time.time() - start_time

start_time = time.time()
text_2 = pytesseract.image_to_string(eroded, lang='DotMatrix',
config=config_tess)
tess_time_2 = time.time() - start_time

# --- EasyOCR ---
easy_reader = easyocr.Reader(['en'], gpu=False)

start_time = time.time()
text_3 = easy_reader.readtext(frame, detail=0, paragraph=False,
                              allowlist='0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ.
:/ ')
```

```
easy_time_1 = time.time() - start_time

start_time = time.time()
text_4 = easy_reader.readtext(eroded, detail=0, paragraph=False,
                              allowlist='0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ.
:/ ')
easy_time_2 = time.time() - start_time

print("------------- Tesseract ---------------")
print(f"Texto OCR na imagem original ({tess_time_1:.4f}s):")
print(text_1)
print(f"Texto OCR na imagem processada ({tess_time_2:.4f}s):")
print(text_2)

print("------------- EasyOCR ------------------")
print(f"Texto OCR na imagem original ({easy_time_1:.4f}s):")
print(text_3)
print(f"Texto OCR na imagem processada ({easy_time_2:.4f}s):")
print(text_4)

# Salva tmp
cv2.imwrite("trigger_ocr_.png", eroded)
```