UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA

CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

RAMIRO CAMPOS DE CASTRO

MONITORAMENTO DE PRODUÇÃO DE TORRADAS EM UMA ESTEIRA INDUSTRIAL POR VISÃO COMPUTACIONAL

FORTALEZA

2023

RAMIRO CAMPOS DE CASTRO

MONITORAMENTO DE PRODUÇÃO DE TORRADAS EM UMA ESTEIRA INDUSTRIAL POR VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Paulo César Cortez

FORTALEZA

2023

RAMIRO CAMPOS DE CASTRO

MONITORAMENTO DE PRODUÇÃO DE TORRADAS EM UMA ESTEIRA INDUSTRIAL POR VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Aprovada em: 01 de Janeiro de 2023

BANCA EXAMINADORA

Prof. Dr. Paulo César Cortez (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Msc. Bruno Riccelli dos Santos Silva
Universidade Federal do Ceará (UFC)

Prof. Msc. Saulo Macedo Maia
Universidade Federal do Ceará (UFC))

À minha família, por todo seu apoio durante esta jornada. Meus pais, meus maiores incentivadores, que sempre investiram em mim, e me motivam a progredir.
Obrigado.

AGRADECIMENTOS

À minha mãe, Ana Jacqueline da Justa, por todo o amor, conselhos, companheirismo e esforços dedicados à minha criação

Ao meu pai, José de Castro Neto, pelos ensinamentos e conselhos nas minhas decisões.

Ao Prof. Dr. Paulo Cesar Cortez , pelo acompanhamento de iniciação científica e estágio, pelos ensinamentos e por me orientar neste trabalho.

Ao Doutorando em Engenharia Elétrica, Ednardo Moreira Rodrigues, e seu assistente, Alan Batista de Oliveira, aluno de graduação em Engenharia Elétrica, pela adequação do *template* utilizado neste trabalho para que o mesmo ficasse de acordo com as normas da biblioteca da Universidade Federal do Ceará (UFC).

À todos os professores por me proporcionar o conhecimento não apenas racional, mas a manifestação do caráter e afetividade da educação no processo de formação profissional, por tanto que se dedicaram a mim, não somente por terem me ensinado, mas por terem me feito aprender.

"O sonho é que leva a gente para frente. Se a gente for seguir a razão, fica aquietado, acomodado."

(Ariano Suassuna)

RESUMO

A Indústria 4.0 simboliza a união da indústria com os avanços tecnológicos que vêm acontecendo, como a inteligência artificial, internet das coisas e robótica. A visão computacional é uma das maneiras que a indústria pode usar para se modernizar, utilizando câmeras para monitorar seus processos e tomar decisões baseadas nos dados obtidos da análise das imagens dos processos. Neste trabalho, é apresentado um sistema de visão computacional que utiliza técnicas de processamento digital de imagem para o monitoramento de produção de torradas em correia industrial, possibilitando respostas automáticas para eventos que possam acontecer na linha de produção. Para a validação do sistema, foram produzidos vídeos de simulações de torradas sendo transportadas em correias industriais, para se ter um ambiente de teste controlado, onde é sabido o ângulo de rotação e o tamanho das torradas.
Foram obtidos resultados satisfatórios, onde o sistema conseguiu detectar todas as torradas, e a detecção do ângulo de rotação das torradas não apresentou erro maior do que 3 graus, mostrando que o sistema foi eficaz na detecção e análise das torradas na linha de produção.

Palavras-chave: Processamento Digital de Imagem. Visão Computacional. Correia industrial.

ABSTRACT

Industry 4.0 symbolizes the union of the industry with the technological advances that are currently happening, such as artificial intelligence, internet of things and robotics. Computer vision is one of the ways that the industry can modernize itself, using cameras to monitor its processes and make decisions based on the data obtained from the analysis of the images of the processes. In this work, it is presented a computer vision system that uses digital image processing techniques for monitoring the production of toasts on a belt conveyor, making it possible to automate responses to events that may occur in a production line. For the system validation, videos were made of simulations of toasts being transported in a belt conveyor, this way there is a controlled test environment, where the angle of rotation and size of the toasts are known. Satisfactory results were obtained, where the system was able to detect all the toasts, and the measured rotation degree of the toasts had a maximum error of three degrees, showing that this system was effective in detecting and analyzing the toasts presented at the production line.

Keywords: Digital image processing. Computer Vision. Conveyor Belt.

LISTA DE FIGURAS

Figura 1 – Imagem Digital 18
Figura 2 – (a) Imagem colorida RGB; (b) componente R; (c) componente G; (d) componente B 20
Figura 3 – Vizinhança 8 do pixel p 20
Figura 4 – Etapas do Processamento de Imagem 21
Figura 5 – Exemplos de aplicação dos filtros de média e mediana 22
Figura 6 – exemplo de limiarização 23
Figura 7 – elementos estruturantes: *(a)* conectividade-4, *(b)* conectividade-8 25
Figura 8 – Exemplo de erosão e dilatação. *(a)* imagem original, *(b)* elemento estruturante, *(c)* imagem resultante da erosão, *(d)* imagem resultante da dilatação 25
Figura 9 – Arquitetura do Sistema 27
Figura 10 – Torradas 30
Figura 11 – Materiais da correia 30
Figura 12 – (a) um *frame* do vídeo, (b) seção do *frame* 34
Figura 13 – Espaços de cor e seus componentes 35
Figura 14 – Pré-Processamento (a) entrada, (b) saída 36
Figura 15 – Histogramas da imagem, (a) antes e (b) depois do pré-processamento . . . 36
Figura 16 – Segmentação (a) entrada, (b) saída 37
Figura 17 – Pós-Processamento (a) entrada, (b) saída 38
Figura 18 – Exemplo do retângulo circunscrito 39
Figura 19 – Ângulo do objeto 39
Figura 20 – Interface de Usuário 41

LISTA DE TABELAS

Tabela 1 – Distribuições usadas na rotação das torradas durante criação dos vídeos . . . 31
Tabela 2 – Resultado do vídeo sem ruído do teste 1 42
Tabela 3 – Resultado do vídeo com ruído do teste 1 42
Tabela 4 – Resultado do vídeo sem ruído do Teste2 42
Tabela 5 – Resultado do vídeo com ruído do Teste2 42
Tabela 6 – Resultado do vídeo sem ruído do teste 3 43
Tabela 7 – Resultado do vídeo com ruído do teste 3 43
Tabela 8 – Resultado do vídeo sem ruído do teste 4 43
Tabela 9 – Resultado do vídeo com ruído do teste 4 43

LISTA DE ABREVIATURAS E SIGLAS

fps *frames per second*

PDI Processamento Digital de Imagem

LISTA DE SÍMBOLOS

$f(x,y)$	Representação espacial de uma imagem bidimensional de entrada
$g(x,y)$	Representação espacial de uma imagem bidimensional de saída
$k(s,t)$	Representação espacial de uma máscara, *kernel*, bidimensional
w	Comprimento da imagem real
w_p	Comprimento da imagem digital
R	Razão entre pixel e centímetros da imagem digital e real
θ_h	Ângulo de visão horizontal da camera
d_c	Distância entre a câmera e a correia
α	Ângulo de rotação
α_l	Ângulo limite de rotação
v	velocidade da correia
v_f	velocidade de um *frame*

SUMÁRIO

1 INTRODUÇÃO 15
1.1 Motivação 15
1.2 Objetivos 16
1.2.1 Objetivo Geral 16
1.2.2 Objetivos Específicos 16
2 FUNDAMENTAÇÃO TEÓRICA 17
2.1 Visão Computacional 17
2.2 Imagem Digital 18
2.2.1 Espaço de cores 19
2.2.2 Vizinhança entre pixels 19
2.3 Processamento Digital de Imagem(PDI) 20
2.3.1 Pré-Processamento 21
2.3.1.1 Filtros espaciais *21*
2.3.2 Segmentação 23
2.3.2.1 Limiarização *23*
2.3.3 Pós-Processamento 24
2.3.3.1 Operações Morfológicas *24*
2.3.4 Extração de Características 25
2.3.5 Reconhecimento e Classificação 26
3 MATERIAIS E MÉTODOS 27
3.1 Proposta 27
3.1.1 Servidor 28
3.1.2 Processamento da Imagem 28
3.2 Implementação 29
3.2.1 Criação dos vídeos 30
3.2.2 Servidor 32
3.2.2.1 Entrada de Imagem *32*
3.2.2.2 Parâmetros Necessários *32*
3.2.2.3 Resposta do Processamento *33*
3.2.3 Processamento da Imagem 33

3.2.3.1 *Pré-Processamento* *34*
3.2.3.2 *Segmentação* *35*
3.2.3.3 *Pós-Processamento* *36*
3.2.3.4 *Extração de Dados* *38*
3.2.3.5 *Reconhecimento e Classificação* *39*
3.3 **Teste de qualidade** 40
4 **RESULTADOS** 41
4.1 **Interface Web** 41
4.2 **Teste de qualidade** 41
4.2.1 ***Teste 1*** 42
4.2.2 ***Teste 2*** 42
4.2.3 ***Teste 3*** 43
4.2.4 ***Teste 4*** 43
5 **CONSIDERAÇÕES FINAIS** 44
5.1 **Conclusão** 44
5.2 **Limitações** 44
5.3 **Trabalhos Futuros** 45
REFERÊNCIAS 46
APÊNDICES 48
APÊNDICE A – Requerimentos do ambiente python 48
APÊNDICE B – Códigos-fontes utilizados para criação do vídeo 49
APÊNDICE C – Códigos-fontes utilizado para o servidor 63
APÊNDICE D – Códigos-fontes utilizados para o Processamento Digital de Imagem 76
APÊNDICE E – Código-fonte utilizado a Interface de Usuário 84

1 INTRODUÇÃO

Nos últimos anos, a sociedade vem passando por grandes avanços tecnológicos, permitindo diversas modernizações que melhoram a qualidade de vida dos seres humanos e impactam diversas atividades econômicas de forma positiva. Um dos elementos da sociedade que se beneficiou desses avanços tecnológicos foi a indústria, que está atualmente na sua quarta revolução, a Indústria 4.0. Segundo o Portal da Indústria (2023), a Indústria 4.0 é um conceito que representa a automação industrial e a integração de diferentes tecnologias como inteligência artificial, robótica, internet das coisas e computação em nuvem com o objetivo de promover a digitalização das atividades industriais, melhorando os processos e aumentando a produtividade.

De acordo com Firjan SENAI (2020), "automação é responsável por facilitar as atividades industriais, com maior eficiência, segurança operacional e redução de custos. Por conta da introdução dos sistemas automatizados de produção, temos hoje bens que são comparativamente mais baratos".

Nesse contexto, a visão computacional se encaixa de diversas maneiras na automação de processos da Indústria 4.0. Analisando as imagens de processos industriais, consegue-se obter dados referentes às etapas dos processos industriais, e a partir destes dados é possível realizar ações bem fundamentadas, por exemplo, detectar defeito em algum produto na linha de produção e acionar um atuador para removê-lo.

Na construção de sistema de visão computacional, é necessário a aplicação de técnicas de processamento digital de imagem (PDI), que consiste na utilização de operações matemáticas para manipular imagens digitais, a fim de que seja possível realizar uma análise da imagem, extraindo informações da mesma. A sequência padrão que um sistema de visão computacional é constituída por três etapas: aquisição; processamento e análise. A aquisição consiste na digitalização de uma imagem. O processamento modifica os valores da imagem digital, com o intuito de prepará-la para análise, seja por pessoas ou computadores. A análise é onde os dados da imagem digital serão avaliados, para que sejam extraídas informações como a quantidade de objetos em uma imagem e o que é cada objeto.

1.1 Motivação

O coração de uma empresa fabril é sua linha de produção, quando ela funciona corretamente, consegue interligar positivamente todos os outros setores: financeiro, comercial,

administrativo e afins (TOTVS, 2018). Entretanto é comum a ocorrência de gargalos na linha de produção.

Segundo Roser *et al.* (2002), um gargalo pode ser definido como qualquer estágio em um sistema que tenha o maior efeito na desaceleração ou na parada completa de um sistema, mesmo que somente por um instante no tempo ou uma média sobre um período de tempo mais longo. Os gargalos podem ter sérias consequências, como a perda da validade do estoque, o que causa atrasos no processo logístico, além de prejuízos financeiros e no atendimento ao cliente.

Diante desse contexto, este trabalho busca implementar um sistema de monitoramento, baseado no processamento de imagens, para o reconhecimento e análise dos objetos que estejam em transição em uma correia industrial.

O cenário que este trabalho será baseado para desenvolver o sistema é a linha de produção de torradas. Neste tipo de linha de produção, caso uma torrada esteja rotacionada acima de um certo limite, ela não entrará no pacote, causando gargalos na linha de produção das torradas.

1.2 Objetivos

Diante desse contexto de automação, visão computacional e o cenário exposto, os seguintes objetivos foram definidos para este trabalho.

1.2.1 Objetivo Geral

O objetivo principal deste trabalho é desenvolver um sistema de processamento de imagem que consiga reconhecer e analisar objetos, usando uma simulação de linha de produção de torradas como teste.

1.2.2 Objetivos Específicos

Além do objetivo geral, os seguintes objetivos específicos devem ser alcançados:

- Desenvolver um modelo de processamento de imagens para detectar, em uma linha de produção, torradas e seu ângulo de rotação
- Avaliar a qualidade do sistema na identificação das torradas
- Mostrar os dados obtidos para o usuário do sistema

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção traz o referencial teórico dos conceitos que sustentam a proposta deste trabalho

2.1 Visão Computacional

A visão computacional busca descrever o mundo que enxergamos por meio de imagens, e reconstruir suas propriedades, como forma, iluminação e distribuição de cor (SZELISKI, 2022). Assim, é possível que o computador analise uma imagem e extraia informações dela, como objetos que estão na imagem, os níveis de intensidade dos *pixels* dos objetos, entre outras características. Com estas informações, é possível reconhecer o que são os objetos e também classificá-los.

Segundo Szeliski (2022), as técnicas de visão computacional podem ser aplicadas em uma ampla variedade de domínios, como robótica, veículos autônomos, medicina, varejo e segurança. Alguns exemplos da aplicação da visão computacional são:

- Na medicina, a tecnologia auxilia no diagnóstico por imagem, seja ela gerada por raio x, tomografia ou ressonância magnética. Um exemplo de aplicação é o *Lung Image Analysis System*, desenvolvido por Felix *et al.* (2010), que é capaz de fazer uma reconstrução em três dimensões do pulmão, a partir de imagens de tomografia dos pulmões, assim auxiliando no diagnóstico de doenças pulmonares;
- Na agricultura, segundo Santos (2020), a visão computacional pode ser empregada na detecção de doenças e pragas, na estimação de safra e na avaliação não invasiva de atributos como qualidade, aparência e volume, além de ser componente essencial em sistemas robóticos agrícolas. Um exemplo é a inspeção automática de frutas, como no trabalho de Costa (2006), onde laranjas são avaliadas para saberem se estão adequadas para serem usadas na produção de suco;
- Na indústria, a visão computacional pode ser usada no monitoramento das linhas de produção, possibilitando a obtenção de informações sobre os produtos de forma automática, ou monitorando a qualidade dos equipamentos, permitindo que manutenções preventivas sejam feitas sob demanda, e evitando que equipamentos sejam danificados. Um exemplo é o trabalho de Xianguo *et al.* (2018), no qual é proposto um sistema de visão computacional para monitorar a condição de correias industriais, para detectar indícios de rasgos

longitudinais.

2.2 Imagem Digital

De acordo com Gonzalez e Woods (2009):

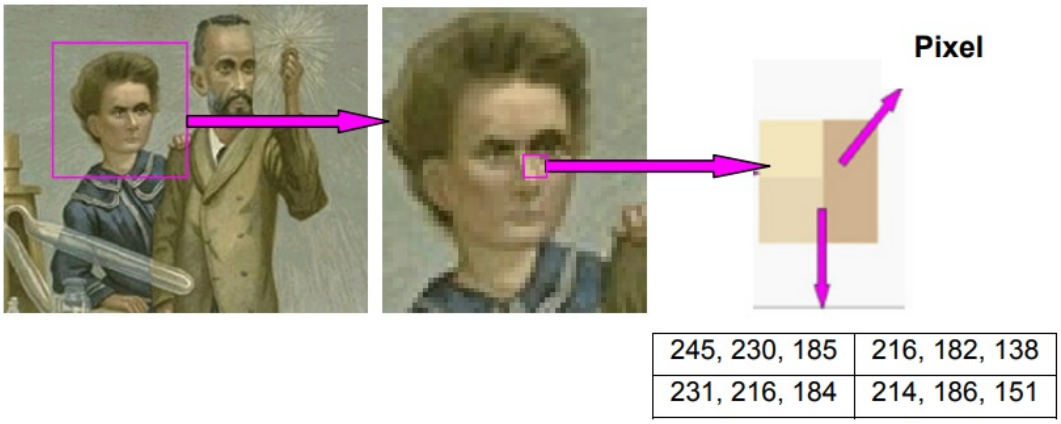
> Uma imagem pode ser definida como uma função bidimensional, f (x, y), em que x e y são coordenadas espaciais (plano), e a amplitude de f em qualquer par de coordenadas (x, y) é chamada de intensidade ou nível de cinza da imagem nesse ponto. Quando x, y e os valores de intensidade de f são quantidades finitas e discretas, chamamos de imagem digital.

Uma das formas de representar uma imagem digital é com a utilização de uma matriz numérica, conforme a Equação 2.1, onde seus elementos são denominados *pixels* (GONZALEZ; WOODS, 2009).

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,n-1) \\ f(1,0) & f(1,1) & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ f(m-1,0) & f(m-1,1) & \dots & f(m-1,n-1) \end{bmatrix} \tag{2.1}$$

Para imagens monocromáticas, os pixels contém apenas um nível de intensidade, que representa uma cor em níveis de cinza, já para imagens coloridas, os pixels contém valores que representam um espaço de cor, onde um pixel possui 3 valores de intensidade, um para cada componente do espaço de cor, como ilustrado na Figura 1.

Figura 1 – Imagem Digital



Fonte: OLIVEIRA (2004).

No caso de um vídeo, ele é composto por um conjunto de imagens que são mostradas em sequência. Cada imagem de um vídeo é chamada de quadro(*frame*), e a quantidade de quadros mostradas em segundo é a medida chamada de *frames per second*(fps).

2.2.1 Espaço de cores

O objetivo de um espaço de cores é facilitar a especificação das cores em alguma forma padronizada, amplamente aceita. Essencialmente, um modelo de cores é uma especificação de um sistema de coordenadas e um subespaço dentro desse sistema no qual cada cor é representada por um único ponto (GONZALEZ; WOODS, 2009).

Alguns exemplos de espaço de cor são: o *RGB*(*Red*, *Green*, *Blue* - Vermelho, Verde e Azul) usado para visualização de cores em telas digitais; o *CMY*(*Cyan*, *Magenta*, *Yellow* - Ciano, Magenta e Amarelo), usado impressão colorida; e o *HLS*(*Hue*, *Saturation*, *Lightness* - Matriz, Saturação e Luminosidade), que foi criado para se assemelhar a maneira que os seres humanos percebem as cores (GONZALEZ; WOODS, 2009; HASTING; RUBIN, 2012).

Em uma imagem colorida, cada pixel corresponde a um ponto no espaço de cores e possui um valor de intensidade para cada coordenada deste espaço. A imagem colorida também pode ser divida em imagens monocromáticas, uma para cada componente do seu espaço de cor.

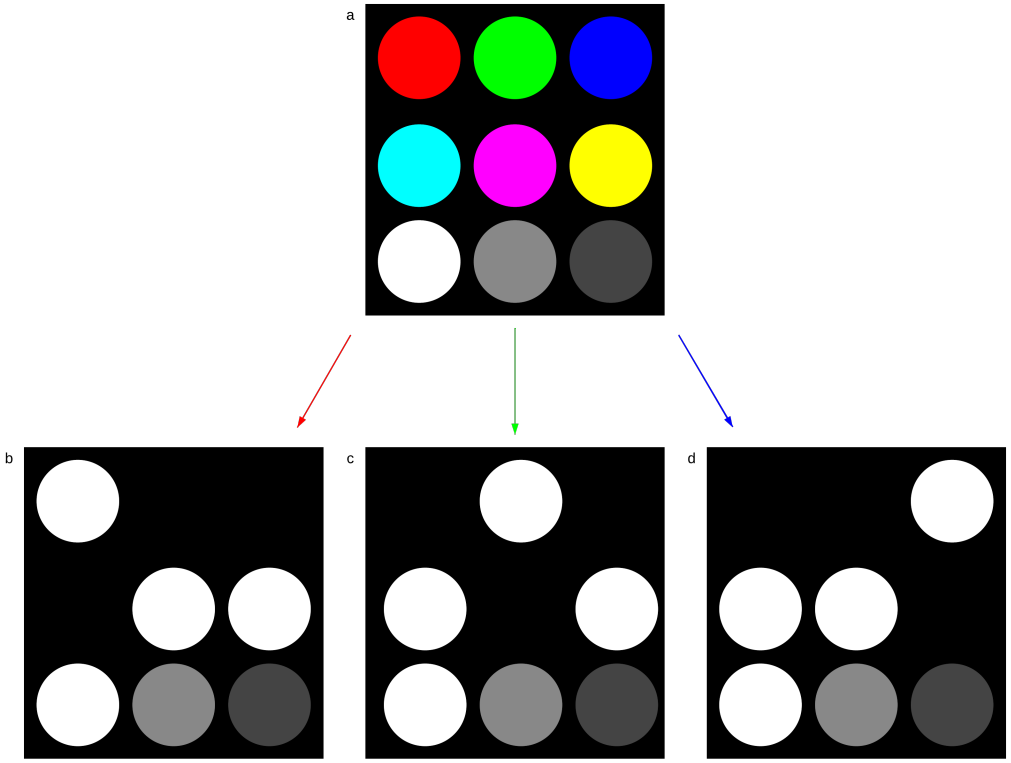
Na Figura 2, temos em (a) uma imagem colorida e visualizada no espaço RGB, e em (b), (c) e (d) temos, respectivamente, a visualização do componente vermelho, verde e azul.

2.2.2 Vizinhança entre pixels

Um pixel p possui 4 vizinhos de borda, horizontais e verticais, este conjunto de vizinhos é chamado *vizinhança-4* e é expresso por $N_4(p)$, além de um conjunto de 4 vizinhos diagonais, chamado de $N_D(p)$. Juntos, os conjuntos $N_4(p)$ e $N_D(p)$ são chamados de *vizinhança-8*, que é expresso por $N_8(p)$ (GONZALEZ; WOODS, 2009). Na Figura 3, o conjunto $N_4(p)$ está simbolizado por b_n, e o conjunto $N_D(p)$ está simbolizado por d_n.

Se um pixel q pertencer ao conjunto $N_4(p)$, é dito que os pixels p e q tem conectividade-4, e se q pertencer a $N_8(p)$, p e q tem conectividade-8.

Figura 2 – (a) Imagem colorida RGB; (b) componente R; (c) componente G; (d) componente B



Fonte: elaborado pelo autor

Figura 3 – Vizinhança 8 do pixel p

d_1	b_1	d_2
b_2	p	b_3
d_3	b_4	d_4

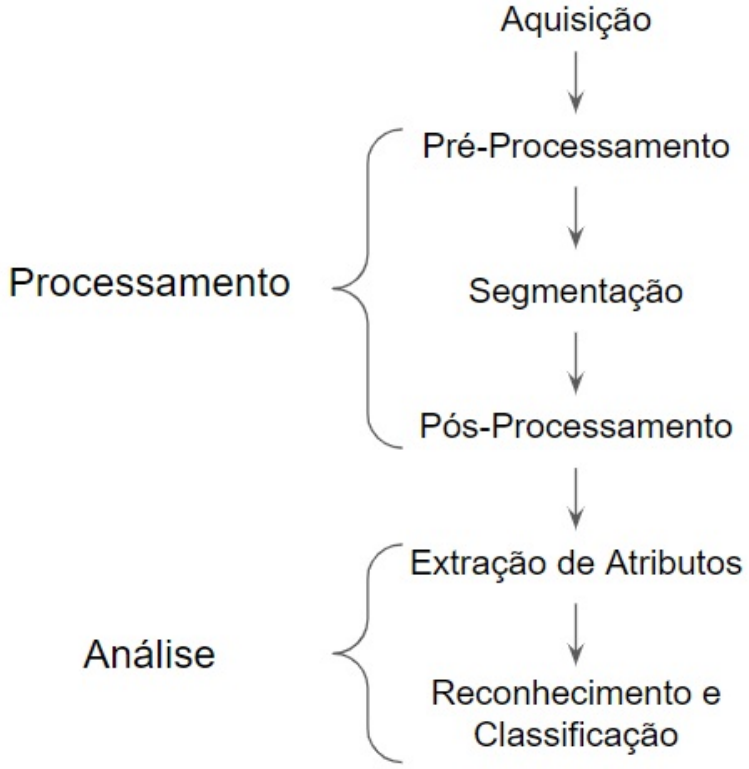
Fonte: elaborado pelo autor.

2.3 Processamento Digital de Imagem(PDI)

O termo PDI se refere a um conjunto de técnicas computacionais utilizadas para a manipulação, interpretação e análise de imagens digitais, o que permite processar imagens de forma rápida e precisa, e possibilitando realizar medidas que seriam impossíveis de serem realizadas manualmente (GOMES, 2001; IGLESIAS, 2008).

Os sistemas de PDI são programas de computador que executam, em imagens digitais, rotinas de procedimentos baseados em operações matemáticas (GOMES, 2001). Essas rotinas podem ser divididas em três etapas: aquisição de imagem digital, processamento digital de imagem e análise digital de imagem. Essas três etapas são divididas como mostrada na Figura 4.

Figura 4 – Etapas do Processamento de Imagem



Fonte: adaptado de OLIVEIRA (2004)

2.3.1 Pré-Processamento

Esta etapa é onde começa o preparo para a imagem ser processada, aplicando técnicas para melhorar a qualidade da imagem e segundo Wang *et al.* (2022), o pré-processamento de imagens pode melhorar a precisão da segmentação.

Alguns efeitos do pré-processamento são: realce de bordas, redução de ruído e ajuste de brilho e contraste. Para obter tais efeitos, uma das técnicas usadas são os filtros espaciais.

2.3.1.1 Filtros espaciais

GOMES (2001) define filtros espaciais como "operações em que o tom de cinza de um determinado pixel na imagem de saída é função não apenas de seu tom de cinza na imagem de entrada, mas também dos tons de cinza de seus pixels vizinhos nesta imagem".

O conjunto de pixels que serão usados na operação é selecionado a partir de uma máscara, ou *kernel*, em que seus elementos têm um peso associado, que afeta o valor do pixel da imagem original.

Realizar uma operação local em uma imagem digital é percorrer cada pixel da imagem com a máscara, atribuindo o valor resultante da operação no conjunto de pixels selecionados

pela máscara na imagem de saída. O pixel em que a operação está sendo realizada corresponde ao pixel central da máscara.

Dois exemplos de filtros espaciais são o filtro de média e o filtro de mediana, técnicas que causam um efeito de borramento na imagem, diminuindo seu contraste e suavizando as bordas presentes nela.

O filtro de média utiliza uma máscara para definir sua vizinhança e atribuir, ao pixel filtrado, o valor da média dos pixels na vizinhança. O valor de cada elemento na máscara é o peso dado ao pixel na imagem de entrada, e o escalar que aparece multiplicando a matriz é calculado a partir da soma dos pesos, e serve para calcular a média dos valores.

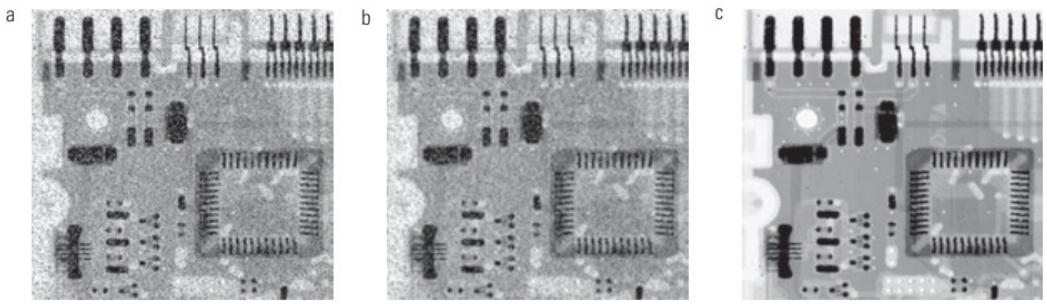
Para uma imagem de tamanho $m \times n$, o resultado de um filtro de média é dado pela equação 2.2, onde a = (m-1)/2, b = (n-1)/2, *k(s,t)* é a máscara, *f(x,y)* é a imagem de entrada e *g(x,y)* é a imagem de saída.

$$g(x,y) = \frac{\sum_{s=-a}^{a}\sum_{t=-b}^{b} f(x+s,y+t)k(s,t)}{\sum_{s=-a}^{a}\sum_{t=-b}^{b} k(s,t)}. \tag{2.2}$$

No filtro de mediana, o kernel é usado apenas para selecionar os pixels, que então são ordenados e tem sua mediana atribuída ao pixel de saída. "Os filtros de mediana são particularmente eficazes na presença de ruído implusivo, também chamado de ruído sal e pimenta, em razão de sua aparência, como pontos brancos e pretos sobrepostos em uma imagem"(GONZALEZ; WOODS, 2009).

A Figura 5 ilusta o uso dos filtros de média e mediana, em (a) temos uma imagem afetada por ruído, em (b) o resultado de um filtro de média de tamanho 3×3 e em (c) temos o resultado de um filtro de mediana de tamanho 3×3.

Figura 5 – Exemplos de aplicação dos filtros de média e mediana



Fonte: (GONZALEZ; WOODS, 2009).

2.3.2 Segmentação

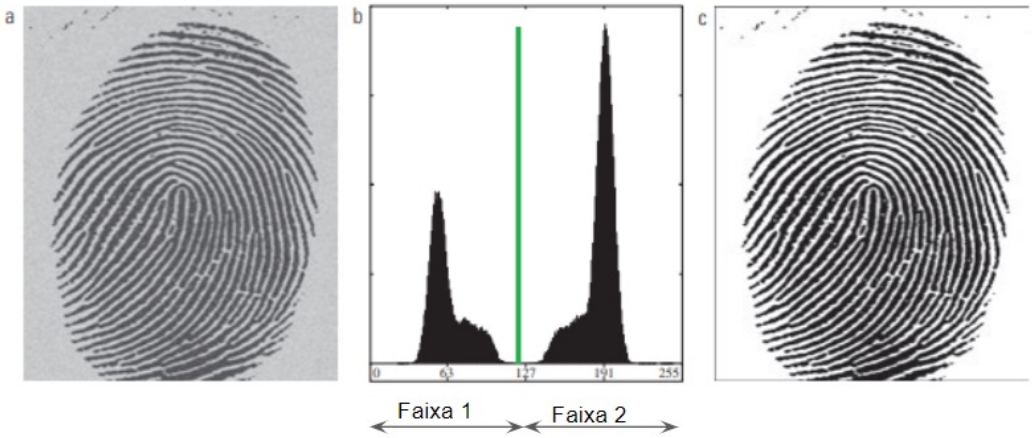
A segmentação tem como objetivo distinguir os pixels da imagem em diferentes grupos, de acordo com os objetos que cada grupo pertence (LONGFEI *et al.*, 2021). Para realizar esse agrupamento, existem diversas técnicas que podem ser implementadas, como detecção de borda, limiarização e segmentação baseada na região.

2.3.2.1 Limiarização

A limiarização, ou *thresholding*, utiliza valores de intensidade como faixas de corte, para agrupar os pixels da imagem em determinadas regiões.

O caso mais simples de limiarização é onde a imagem resultante é uma imagem binária, dividida em duas regiões. Uma das regiões representa os objetos da imagem, e a outra o seu fundo.

Figura 6 – exemplo de limiarização



Fonte: Adaptado Gonzalez e Woods (2009).

No exemplo da Figura 6, em *(a)* temos a imagem de uma digital, em *(b)* temos o seu histograma, com uma linha verde marcando o valor de intensidade usado para realizar a limiarização, e em *(c)* temos a imagem dividida em duas faixas de intensidade, com a digital sendo composta pelos pixels que estavam na faixa 1, representada pelos pixels pretos de valor 0, e o fundo sendo os pixels que estavam na faixa 2, que se tornaram brancos, de valor 1.

Um caso particular da limiarização é o método de OTSU (1979), um método iterativo que consiste em achar a faixa de corte que maximize a variância entre regiões, para isso é calculada a distribuição de frequência dos níveis de intensidade dos pixels da imagem e então é calculado a variância entre as possíveis faixas de corte, sendo selecionada a que resulte em

menor variância. Esse método é uma maneira automática de se fazer a limiarização de uma imagem, e é particularmente eficiente para imagens que apresentem uma grande distinção entre seus níveis de cor, geralmente representado por um vale no histograma, como visto na Figura 6 (b).

2.3.3 Pós-Processamento

Nem sempre os resultados da segmentação são suficientes para realizar a detecção dos objetos, e para isso o pós-processamento tem a finalidade de corrigir a imagem proveniente da segmentação. Uma das maneiras de ajustar a imagem é por meio de procedimentos como a separação, união ou eliminação de objetos, que podem ser realizados através de operações lógicas e de morfologia matemática (IGLESIAS, 2008).

2.3.3.1 Operações Morfológicas

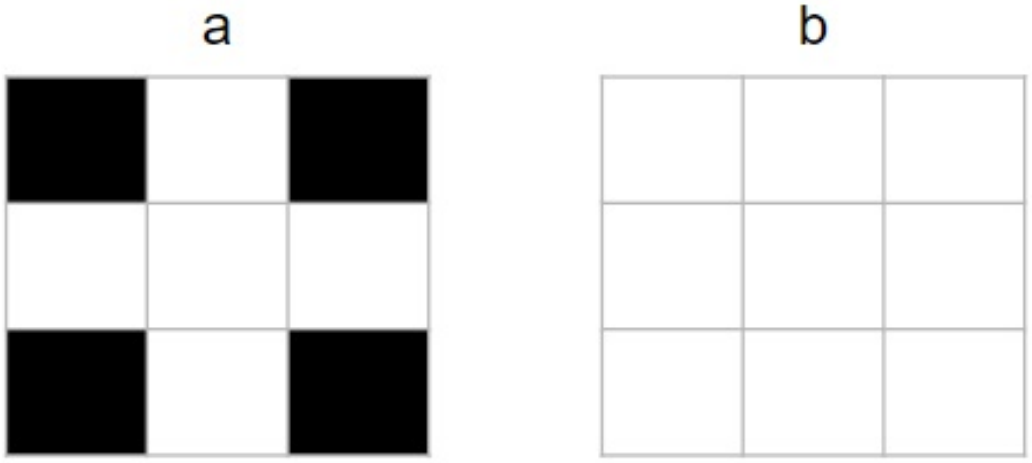
Um dos tipos de técnicas de pós-processamento são as operações morfológicas, baseadas na morfologia matemática. "Os conjuntos em morfologia matemática representam os objetos encontrados em uma imagem. Por exemplo, o conjunto de todos os pixels brancos em uma imagem binária é uma descrição morfológica completa da imagem"(GONZALEZ; WOODS, 2009).

Similares às operações locais, elas também possuem uma máscara para definir a vizinhança do pixel e que percorrerá a imagem, alterando o valor de seus pixels com base no resultado da operação. No caso das operações morfológicas, a máscara é chamada de elemento estruturante, como mostrados na Figura 7, em que os quadrados brancos representam os pixels que serão parte da vizinhança na operação morfológica.

As operações básicas são a erosão e a dilatação, que são operações com resultados opostos. Na operação de erosão, ao percorrer a imagem, se o pixel de entrada for branco e pelo menos um dos vizinhos definidos pelo elemento estruturante seja preto, o pixel é invertido na imagem de saída, isso faz com que objetos na imagem diminuam de tamanho, podendo até mesmo serem eliminados. Já na operação de dilatação, se o pixel de entrada for preto, e possuir pelo menos um vizinho branco, o pixel é invertido na saída, isso faz com que objetos na imagem diminuam de tamanho, podendo até mesmo terem sua região unida a outros objetos.

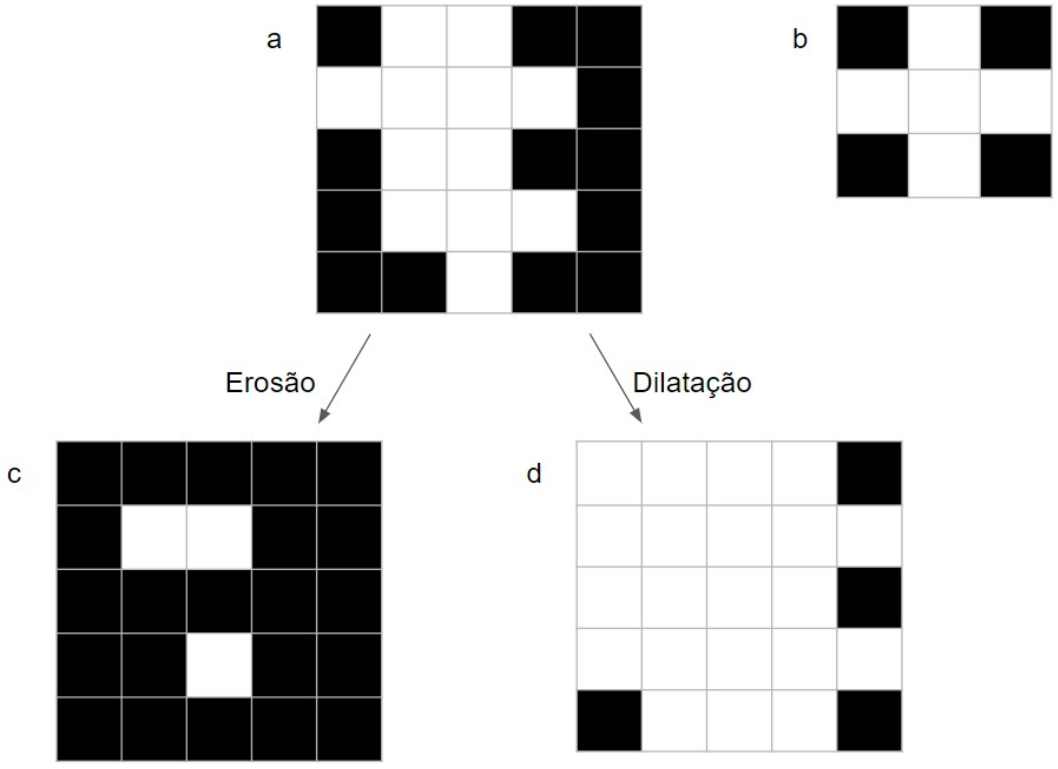
Na Figura 8, temos o exemplo do uso de um elemento estruturante de conectividade-4 nas operações de erosão e dilatação, sobre a matriz mostrada na Figura 8 (a).

Figura 7 – elementos estruturantes: *(a)* conectividade-4, *(b)* conectividade-8



Fonte: elaborado pelo autor.

Figura 8 – Exemplo de erosão e dilatação. *(a)* imagem original, *(b)* elemento estruturante, *(c)* imagem resultante da erosão, *(d)* imagem resultante da dilatação



Fonte: elaborado pelo autor.

2.3.4 Extração de Características

Neste ponto do processo, a imagem está pronta para ser analisada. Esta etapa consiste na realização de medidas na imagem, sendo obtidas características tanto da imagem quanto dos objetos presentes nela. Entre as características que podem ser obtidas, temos a contagem dos objetos, as medidas de tamanho e posição dos objetos.

A contagem de objetos em uma imagem é realizada analisando as regiões de pixels conectados, e de mesmo valor, presentes na imagem. Suzuki e Abe (1985) descreveram um algoritmo usado para achar o contorno de objetos na imagem, utilizando a contiguidade de pixels de mesma tonalidade em uma região, com uma conectividade-4 ou -8, e a partir destes contornos é possível identificar os objetos na imagem.

As medidas de tamanho constituem-se nas medidas geométricas básicas dos objetos, como sua área e perímetro GOMES (2001). Um método para achar estas informações do objeto é circunscrever este objeto com um retângulo, e então extrair as características deste retângulo, como sua largura e altura. Ainda utilizando o retângulo circunscrito, podemos calcular seu ponto central para achar a posição do objeto na imagem.

2.3.5 Reconhecimento e Classificação

Na etapa final do processo, os dados previamente extraídos são utilizados para classificar os objetos na imagem. De acordo com COSTA e JR. (2000), os problemas de classificação podem ser divididos em três tipos:

- **Critério imposto:** os critérios de classificação são impostos de acordo com cada problema, e então as características medidas são comparadas com estes critérios para serem classificadas.
- **Classificação supervisionada:** são fornecidos um ou mais conjuntos de dados sobre classes de objetos já conhecidos, que são usados como referência para a classificação.
- **Classificação não supervisionada:** não há conhecimento prévio das classes de objetos, e a classificação ocorre, geralmente, por meio de técnicas de agrupamento de dados.

Para solucionar problemas de classificação supervisionada e não-supervisionada, podem ser utilizadas técnicas de aprendizado de máquina para identificar e classificar os objetos baseados nos dados existentes.

Aprendizado de máquina é uma abordagem guiada a dados para a resolução de problemas, na qual padrões em um conjunto de dados são identificados e utilizados para ajudar na tomada de decisões (BRINK *et al.*, 2016).

As técnicas de aprendizado de máquina podem ser divididas em duas categorias, as de classificação supervisionada e as não supervisionadas. Na classificação supervisionada, temos como exemplo: máquina de vetores de suporte(SVM), k-vizinho mais próximo(KNN) e árvores de decisão. Na classificação não supervisionada, temos como exemplo: k-médias(K-Means).

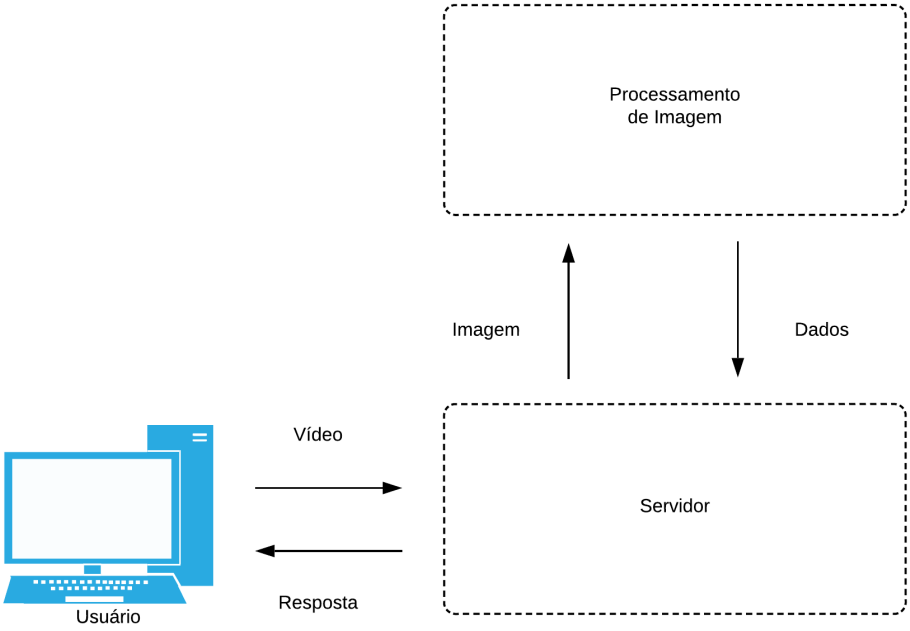
3 MATERIAIS E MÉTODOS

Este capítulo apresenta a proposta do sistema, as ferramentas usadas na sua implementação e a justificativa de suas escolhas, bem como testes realizados para a validação do sistema.

3.1 Proposta

O sistema proposto neste trabalho é dividido em duas partes, como mostrado na Figura 9.

Figura 9 – Arquitetura do Sistema



Fonte: elaborado pelo autor.

Esta estrutura permite que o usuário forneça, para o servidor, uma fonte de vídeo e alguns dados sobre o objeto analisado, a câmera que gravou o vídeo e a correia. Com estes dados, o servidor consegue calcular as informações necessárias para o processamento do vídeo, e assim conseguir detectar as torradas e identificar quais estão fora do padrão. Para este trabalho, o usuário deve fornecer um ângulo limite de rotação das torradas, que vai ser usado para classificar uma torrada como fora de padrão.

Na medida que o processamento das imagens do vídeo é feito, o servidor expõe, na interface de usuário, as informações sobre a quantidade de torradas detectadas, a quantidade de

torradas fora do padrão e a posição das torradas fora de padrão.

3.1.1 Servidor

Esta camada é onde o usuário consegue se comunicar com o sistema, sendo composta por 3 partes.

- **Entrada de Imagem:** o usuário seleciona a fonte de vídeo a ser fornecida para processamento;
- **Definição de parâmetros:** o usuário define algumas informações do vídeo e as características padrão do objeto a ser analisado, para que o processamento seja realizado apropriadamente;
- **Resposta do Processamento:** por meio de uma interface web, o usuário recebe uma resposta visual do sistema, com a contagem dos objetos achados, a quantidade de objetos fora do padrão, e uma marcação visual dos objetos fora de padrão.

Para a comunicação do sistema , o servidor terá três processos, um para servir a interface de usuário, onde o usuário escolherá a fonte de vídeo e fornecerá os parâmetros do sistema; uma para lidar com a entrada de vídeo escolhida pelo usuário, exibindo-a na interface de usuário e mandando as imagens para o processamento; e uma para executar o processamento da imagem.

O servidor também armazena, e atualiza, a quantidade de torradas que foram detectadas e a posição na correia das torradas que foram classificadas como fora de padrão, para exibir na interface de usuário.

3.1.2 Processamento da Imagem

Nesta camada, constituída das etapas descritas na seção 2.3, a imagem será recebida do servidor, processada, analisada e então uma resposta será devolvida para o servidor, contendo os dados obtidos no processamento.

Como o objetivo é a análise de apenas um tipo de objeto, as torradas, e em um mesmo vídeo as torradas tem a mesma tonalidade, pode ser processado apenas imagens monocromáticas, em que as torradas tenham uma tonalidade diferente da correia. Para isso, será escolhido um componente de algum espaço de cor em que as torradas tenham níveis de cinza distintos da correia e então a imagem colorida será processada apenas neste componente.

O processamento realizado é constituído de 5 etapas:

- **Pré-Processamento:** a imagem colorida é transformada em uma imagem em níveis de cinza, e são utilizados filtros espaciais para reduzir o ruído e melhorar o contraste entre as torradas e a correia;
- **Segmentação:** a imagem em níveis de cinza é limiarizada, resultando em uma imagem binária, com os objetos sendo representados por pixels de intensidade 1 e o fundo por pixels de intensidade 0;
- **Pós-Processamento:** são realizadas operações morfológicas na imagem binária para remover ruídos e corrigir buracos dentro da área dos objetos ;
- **Extração de dados:** é detectado o contorno dos objetos e um retângulo é circunscrito nestes contornos para ser calculado o tamanho e ângulo de rotação dos objetos;
- **Reconhecimento e Classificação:** pelo tamanho dos objetos na imagem, são reconhecidos quais são torradas, e estes são classificados como dentro ou fora do padrão, de acordo com o ângulo limite de rotação estabelecido usuário.

3.2 Implementação

O desenvolvimento e testes deste sistemas foram feitos em um notebook ASUS X510URR, e para sua criação, foi utilizado majoritariamente a linguagem *Python*[1], devido ao seu rico ecossistema de bibliotecas voltadas para computação numérica e processamento de imagens. As principais bibliotecas utilizadas para desenvolver este trabalho foram *NumPY*[2] e *OpenCV*[3]

NumPY é a biblioteca fundamental para computação científica em *Python*.

Ela é especializada em manipular matrizes e vetores, garantindo maneiras fáceis e rápidas de se trabalhar com estes tipos de dados.

A biblioteca *OpenCV*[4] fornece algoritmos de visão computacional e aprendizado de máquina. Ela é uma biblioteca feita usando *NumPY* como base, e foi escolhida para facilitar o trabalho com imagens e vídeos, além de possuir a implementação dos algoritmos de manipulação de imagem usados neste trabalho.

Os requerimentos do ambiente python utilizado no desenvolvimento se encontram no Apêndice A .

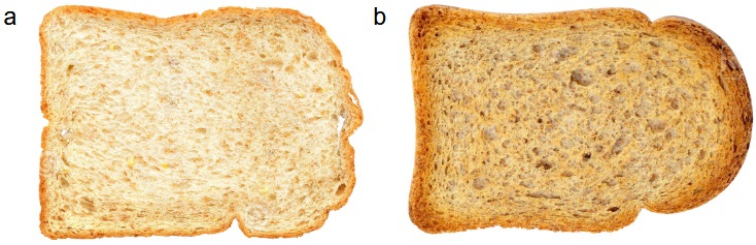
[1] https://www.python.org/
[2] https://numpy.org/
[3] https://docs.opencv.org/4.1.1/index.html
[4] https://docs.opencv.org/4.1.1/index.html

3.2.1 Criação dos vídeos

Devido a falta de imagens reais para serem analisadas, foi necessário criar um vídeo simulando a atuação de uma esteira industrial com torradas, para isso, foram selecionados dois tipos de torradas, como mostrado na figura 10 e duas tonalidades para a correia transportadora, como mostrado na figura 11, a seleção destas imagens foi feita com o intuito de testar o sistema em situações de diferentes contrastes.

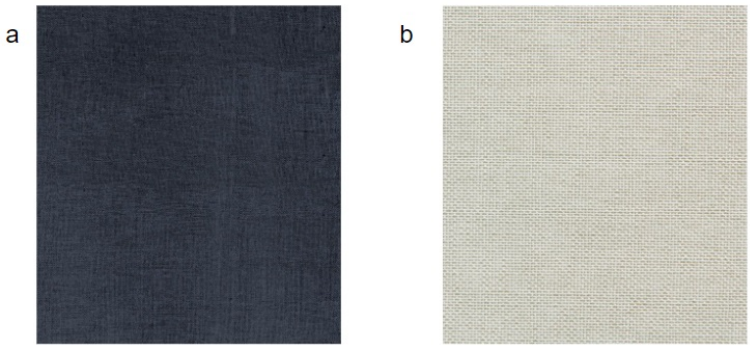
Foram feitos quatro vídeos, um para cada combinação de torrada com material de correia, e cada um contendo 50 torradas. Para cada um dos vídeos originais, foi gerado mais um vídeo, adicionando um ruído gaussiano em todos os *frame* dos vídeos, para simular uma imagem de baixa qualidade, totalizando oito vídeos, que serão usados para testar a qualidade do sistema. Para a adição do ruído, foi utilizada a biblioteca *skimage*[5].

Figura 10 – Torradas



Fonte: elaborado pelo autor.

Figura 11 – Materiais da correia



Fonte: elaborado pelo autor.

Na criação dos vídeos, foi assumido que a câmera captura a imagem perpendicularmente a correia, e foram definidos os seguintes parâmetros para uniformizar a qualidade deles:

[5] https://scikit-image.org/docs/0.16.x/

- **Câmera:** ângulo de filmagem(θ) = 90º, distância entre câmera e correia(d_c) = 50 cm, quadros por segundo(fps) = 30, ângulo de visão horizontal(θ_h) = 90º;
- **Imagem digital:** comprimento(w_p) = 1280 px, altura(h_p) = 720 px;
- **Objeto:** comprimento(w_o) = 8 cm, altura(h_o) = 6 cm;
- **Correia:** distância entre colunas de objetos = 4 cm, quantidade de objetos em uma coluna = 5, velocidade(v) = 8 cm/s.

Com essas informações é possível saber o comprimento real(w), em centímetros, que a imagem capturou, utilizando a fórmula:

$$w = 2 \cdot d_c \cdot tan(\frac{\theta}{2}). \tag{3.1}$$

e sabendo o comprimento em centímetros e o comprimento em pixels, é possível saber a razão(R) *pixel*/cm entre imagem digital e imagem real:

$$R = \frac{w_p}{w}. \tag{3.2}$$

Cada torrada foi gerada com um uma pequena variação em seus características, com sua altura e seu comprimento sendo adicionados de um valor aleatório entre -0.1 e 0.1 centímetros, e com seu ângulo de rotação sendo escolhido a partir de um valor aleatório de uma distribuição normal, que foi selecionada entre as três distribuições mostradas na tabela 1. Após a geração de cada torrada, suas características(altura, comprimento e ângulo de rotação) foram salvas para servirem de referência para os testes. Para rotacionar a imagem, foi utilizada a biblioteca *imutils*[6].

Tabela 1 – Distribuições usadas na rotação das torradas durante criação dos vídeos

distribuição	média	desvio padrão	chance de ocorrência
1	-10	4	10%
2	0	4	80%
3	10	4	10%

Os códigos usados na criação do vídeo se encontram no Apêndice B.

[6] https://github.com/PyImageSearch/imutils

3.2.2 Servidor

O servidor, criado utilizando as bibliotecas *Flask*[7], *websocket* [8], é responsável por executar o sistema de processamento de imagem e se comunicar com o usuário. *Flask* foi escolhido por ser uma biblioteca simples para criação se servidores, e *websocket* foi escolhido para proporcionar uma comunicação bidirecional entre a interface de usuário e o servidor.

O servidor funciona usando um sistema de *threads* e *queues*(filas), utilizando as bibliotecas *Threading*[9] e *Queue* [10] respectivamente. As threads foram usadas para possibilitar o processamento das imagens sem bloquear as ações do usuário na interface, e as filas foram usadas para serializar o envio das imagens do vídeo, assim a interface só envia uma nova imagem após receber a resposta da imagem anterior.

3.2.2.1 Entrada de Imagem

Na interface, o usuário seleciona uma fonte de vídeo para fornecer as imagens ao sistema, seja um arquivo de vídeo ou uma câmera que está fazendo a transmissão.

3.2.2.2 Parâmetros Necessários

Para o correto funcionamento do sistema, o usuário tem de fornecer as seguintes informações referentes a:

- **Câmera:** distância entre câmera e correia(d_c), quadros por segundo(fps), ângulo de visão horizontal(θ_h);
- **Imagem digital:** comprimento(w_p);
- **Torrada:** comprimento(w_o) = 8 cm, altura(h_o);
- **Correia:** velocidade(v).

Com estas informações, é possível calcular a razão entre pixels e centímetros da imagem, pela Equação 3.2, e assim saber qual a altura e comprimento, em pixels, que as torradas tem nas imagens.

Além disso, é necessário calcular quantos pixels são deslocados a cada *frame*, para que seja possível manter o registro das torradas mesmo após elas saírem da zona de filmagem da

[7] https://flask.palletsprojects.com/en/1.1.x/
[8] https://websockets.readthedocs.io/en/8.1/
[9] https://docs.python.org/3/library/threading.html
[10] https://docs.python.org/3/library/queue.html

câmera. O cálculo desse deslocamento(v_f) é dado por:

$$v_f = \frac{v * R}{fps} \tag{3.3}$$

3.2.2.3 Resposta do Processamento

Para mostrar a resposta do processamento ao usuário, foi criada uma interface web, que consiste de uma página *HTML*[11], estilizada com *CSS*[12], e que possui um script de *JavaScript*[13] para se comunicar com o servidor. A interface é responsável por mostrar ao usuário a quantidade de objetos detectados, a quantidade de objetos que estavam fora do padrão estabelecido pelo usuário, e por mostrar o vídeo que está sendo processado, dando destaque aos objetos fora de padrão que estão em tela.

Os códigos usados na criação do servidor se encontram no Apêndice C, e o código para a interface do usuário se encontra no Apêndice E.

3.2.3 Processamento da Imagem

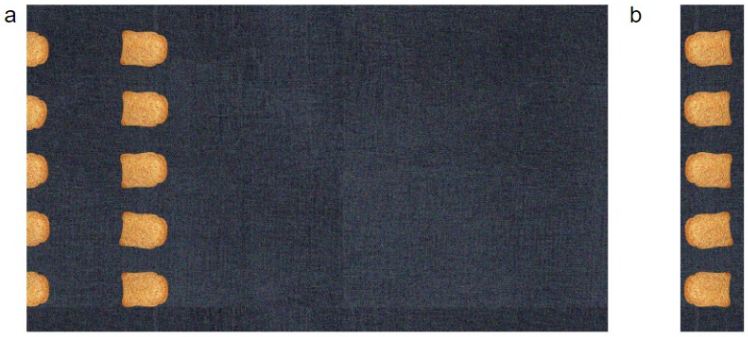
Esta parte do sistema é responsável por processar cada *frame* do vídeo fornecido pelo usuário, e então enviar para a interface o resultado da análise do *frame*. Como a correia tem uma movimentação unidirecional, é possível utilizar apenas uma seção perpendicular à direção de seu movimento para fazer o processamento das imagens. A diminuição do tamanho da imagem a ser processada foi uma decisão para diminuir os custos computacionais necessários para realizar o processamento das imagens, o que permite realizar as operações mais rapidamente e diminui a necessidade de recursos computacionais.

As imagens mostradas nesta seção são as imagens do vídeo adicionado de ruído, com a Torrada da Figura 10 (b) e o Material da Figura 10 (a).

Na Figura 12 (b), temos um exemplo de imagem utilizada no processamento, ela tem a mesma altura da imagem original, e comprimento igual a 130% do comprimento da torrada. Este valor para o comprimento desta regiaõ foi escolhido por ser o suficiente para garantir que uma torrada inteira consiga passar pela imagem e sempre tenha torradas em tela, pois se ficar não tiver torradas em tela, a limiarização de Otsu não terá bons resultados devido a falta de contraste na imagem.

[11] https://html.spec.whatwg.org/multipage/
[12] https://www.w3.org/Style/CSS/
[13] https://developer.mozilla.org/en-US/docs/Web/JavaScript

Figura 12 – (a) um *frame* do vídeo, (b) seção do *frame*



Fonte: elaborado pelo autor.

Os códigos usados para o processamento e análise das imagens se encontram no Apêndice D, com os códigos de pré-processamento, segmentação e pós-processamento código-fonte 8 - Processamento da Imagem, e os códigos da extração de atributos e reconhecimento das torradas estão no código-fonte 9 - Análise da Imagem.

3.2.3.1 Pré-Processamento

Como serão processadas apenas imagens monocromáticas, o primeiro objetivo do pré-processamento é transformar a imagem original em uma imagem com apenas um nível de intensidade. Para realizar essa transformação, foram analisados alguns espaços de cor e seus componentes, visando encontrar uma transformação que resulte em uma imagem com alto contraste entre a torrada e a correia, para facilitar a segmentação.

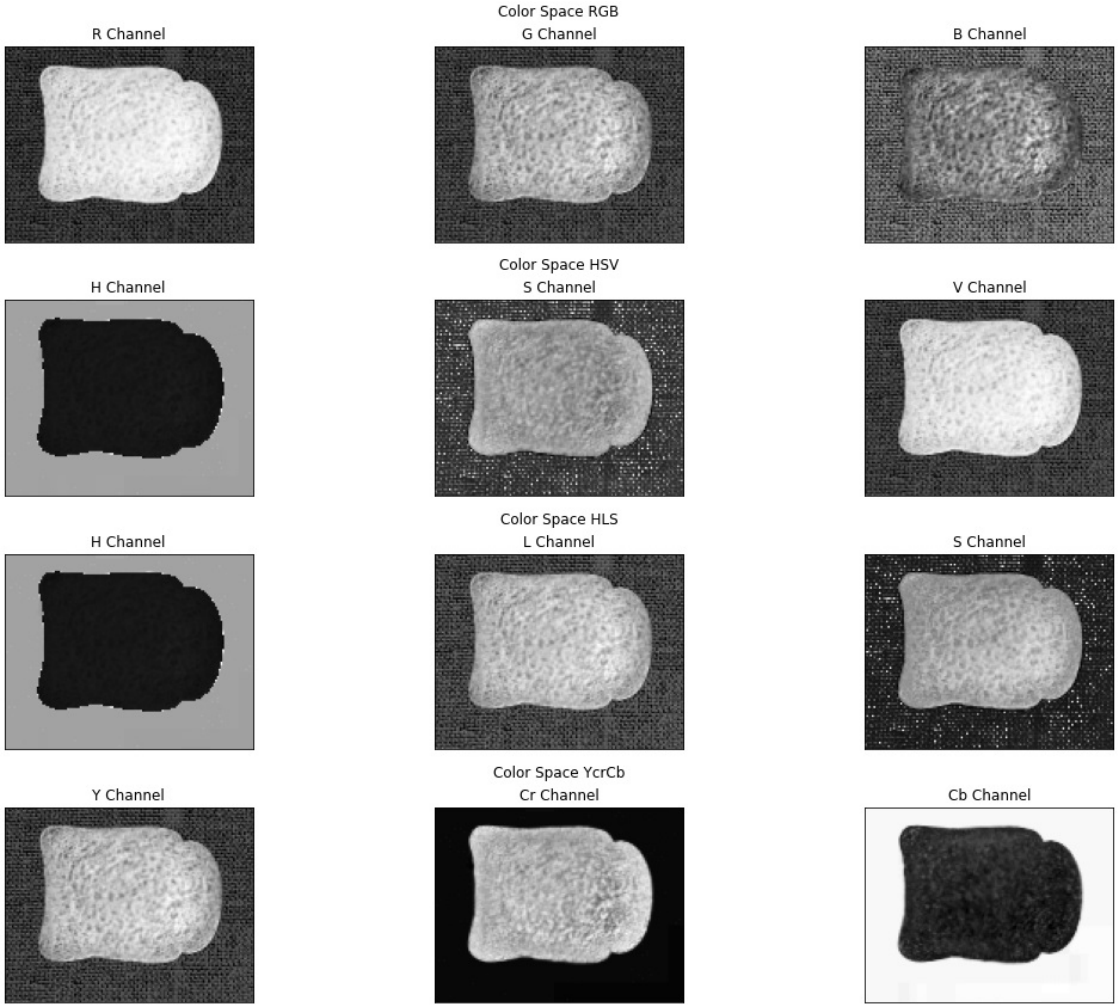
A decisão de qual espaço de cor usar e componente ser usado foi feita após uma análise visual das imagens e seus histogramas em cada componente de alguns espaços de cor, e um teste da segmentação gerada para cada componente.

No final da seleção, o espaço HLS(*Hue, Lightness, Saturation*), no seu componente *S*, foi o que apresentou melhores resultados para limiarização das torradas. A Figura 13 expõe a imagem monocromática das camadas de alguns espaços de cor testados.

Com a seção perpendicular selecionada e a imagem sendo convertida em uma monocromática, são realizadas duas operações locais, ambas para redução de ruído. A primeira é a utilização de um filtro de mediana, mencionado na seção 2.3.1.1, com um *kernel* quadrado de tamanho 3, para remover valores muito discrepantes em relação aos seus vizinhos. Em seguida, é utilizado um filtro de média, com um *kernel* quadrado de tamanho 5 e pesos de valor 1.

A figura 14 ilustra a imagem de entrada no componente *Saturation* do espaço de cor HSL, e a imagem saída desta etapa, após a realização das operações de filtro de mediana e média.

Figura 13 – Espaços de cor e seus componentes



Fonte: elaborado pelo autor.

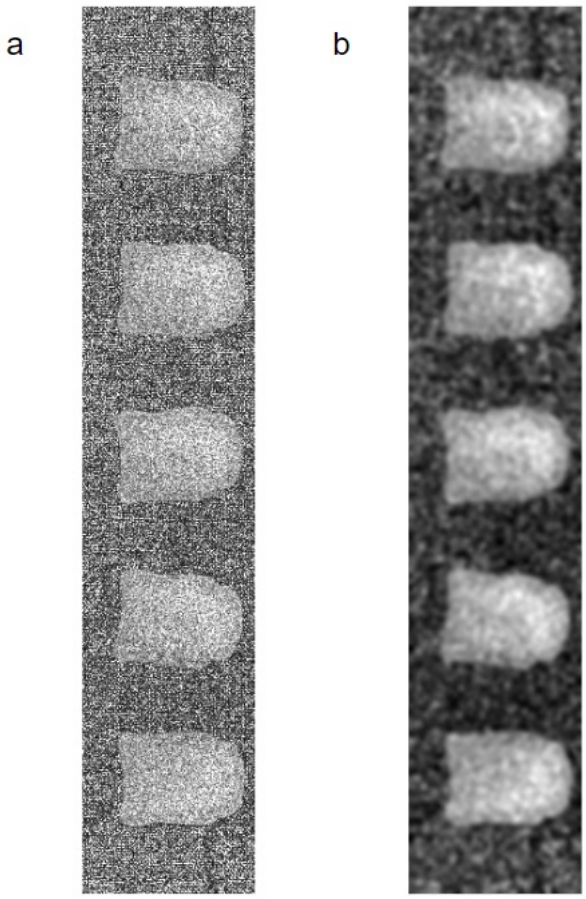
Pode-se ver que a utilização dos filtros fez com que a correia e as torradas ficassem com um maior contraste de tonalidade.

Na Figura 15, temos os histogramas das imagens de entrada e a de saída do pré-processamento, e na Figura 15 (b), pode-se ver que o pré-processamento foi efetivo em criar uma distinção entre as duas regiões da imagem, que é o resultado desejado para que a limiarização de Otsu obtenha um ótimo resultado.

3.2.3.2 Segmentação

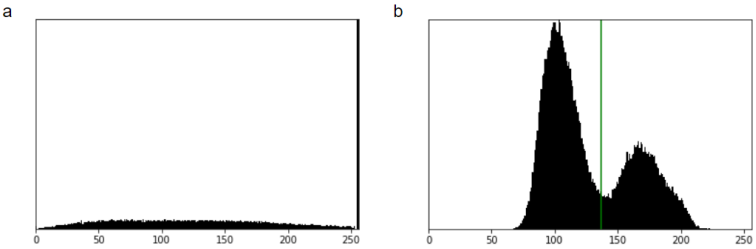
Nesta etapa, é realizado o processo de limiarização de Otsu, descrito na seção 2.3.2.1, que vai segmentar a imagem em duas regiões, uma representando os objetos presente nela, e a outra representando o seu fundo. Na Figura 15 (b), a linha verde representa o valor de limiarização achado pelo método de Otsu para esta imagem.

Figura 14 – Pré-Processamento (a) entrada, (b) saída



Fonte: elaborado pelo autor.

Figura 15 – Histogramas da imagem, (a) antes e (b) depois do pré-processamento



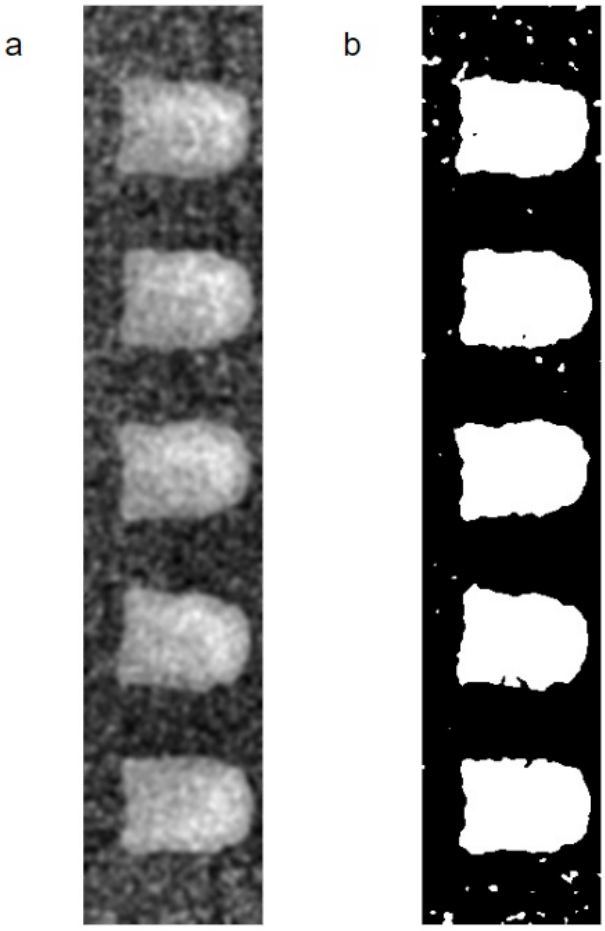
Fonte: elaborado pelo autor.

Na Figura 16 (b) temos o resultado da segmentação, uma imagem binária onde as regiões brancas tem valor 1 e é constituída dos objetos de interesse e alguns ruídos, enquanto a região preta tem valor 0 e representa o fundo da imagems.

3.2.3.3 Pós-Processamento

Para melhorar o resultado da segmentação, como completar torradas que ficaram divididas, e diminuir os ruídos ainda presentes na imagem, é necessário usar-se de técnicas de

Figura 16 – Segmentação (a) entrada, (b) saída



Fonte: elaborado pelo autor.

operação morfológica. Foram usadas duas técnicas, que combinam as operações de erosão e dilatação explicadas na seção 2.3.3.1, são elas as operações de abertura e fechamento.

A primeira delas a ser realizada é a operação de abertura, que é a combinação de uma operação de erosão seguida de uma de dilatação, e tem como efeito a remoção de pequenos ruídos na imagem, como os pontos brancos presentes na imagem de entrada, que podem ser vistos na Figura 17 (a). Foi utilizado um elemento estruturante similar ao mostrado na Figura 7 (b), mas com um tamanho de 5, para aumentar o tamanho dos ruídos afetados.

A segunda operação realizada é a de fechamento, que é a combinação de uma operação de dilatação seguida de uma erosão, e ela tem o efeito de fechar pequenos buracos na imagem, e que é útil para juntar pedaços do objeto quando ele acaba sendo separado na segmentação. Para esta operação também é utilizado um elemento estruturante similar ao mostrado na Figura 7 (b), porém com um tamanho 9, para aumentar o tamanho dos ruídos afetados.

Na figura 17 (b) temos a imagem resultante do pós-processamento, onde é possível

Figura 17 – Pós-Processamento (a) entrada, (b) saída

Fonte: elaborado pelo autor.

ver que a maioria dos pontos de ruídos foram eliminados e as torradas tiveram seu contorno e interior preenchidos.

3.2.3.4 Extração de Dados

Nesta etapa, os objetos na imagem são detectados e tem sua altura, comprimento, posição e ângulo de rotação extraídos. É utilizado a técnica de Suzuki e Abe (1985), usando a conectividade-8, para encontrar os contornos dos objetos presentes na imagem. Cada contorno achado é circunscrito por um retângulo, como exemplificado na Figura 18, que é usado para aproximar três características de interesse do objeto:

- Posição: o ponto central do retângulo é considerado a posição do objeto na imagem.

- Altura e Comprimento: o comprimento do objeto é o maior lado do retângulo, enquanto a altura é o menor.

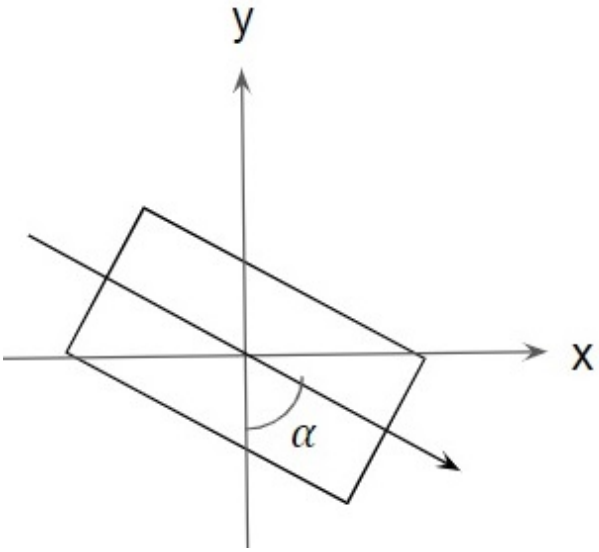
- Ângulo: o ângulo de rotação(α) do objeto é calculado a partir do ângulo que o eixo

do comprimento do retângulo faz com o eixo horizontal da imagem, como mostrado na Figura 19.

Figura 18 – Exemplo do retângulo circunscrito

Fonte: elaborado pelo autor.

Figura 19 – Ângulo do objeto



Fonte: elaborado pelo autor.

3.2.3.5 Reconhecimento e Classificação

Neste trabalho, a classificação das torradas foi feita por meio de critérios impostos, e foram usados três critérios para a classificação dos objetos, dois para classificar o objeto como torrada, e um para classificar a torrada como fora de padrão.

Para reconhecer um objeto como torrada, primeiro é checado o tamanho deste objeto, para saber se altura e largura estão variando até 5% do tamanho de torrada fornecido pelo usuário. Depois é checado a área, que é a quantidade de pixels branco, ocupada pelo objeto no retângulo circunscrito, pois as vezes os ruídos da imagem acabam ficando com um tamanho semelhante aos da torrada, e por testes empíricos, observou-se que as torradas ocupam uma área entre 50 e

60% do retângulo.

Passado nessas duas checagens, objeto será classificado como torrada, e para a ele é marcado como dentro do padrão se seu ângulo de rotação(α) for igual ou inferior ao fornecido pelo usuário(α_l), e fora de padrão caso contrário.

Como está sendo analisado um vídeo, e a mesma torrada aparece em diversos *frames*, é necessário ter o cuidado de registrar cada torrada apenas uma vez, e não a cada *frame* que ela aparecer. Para isso ser possível, é checado o centro de massa das torradas detectadas no *frame* atual e é feito uma comparação de sua distância para os centros de massa das torradas já armazenadas.

Por testes empíricos, foi determinado que uma torrada só será registrada se ela tiver seu centro de massa a uma distância maior que 4*v_f que os centro de massa das outras torradas já registradas.

3.3 Teste de qualidade

Com o objetivo de avaliar a qualidade do sistema, na criação dos 4 vídeos originais, as características, altura, comprimento e ângulo de rotação, de cada uma das torradas geradas foram salvas de forma ordenada em um arquivo, para serem usados como referência aos resultados obtidos pelo processamento. Os 8 vídeos gerados passaram pelo processamento de imagens do sistema e também tiveram as características das torradas detectadas salvas em um arquivo, para serem comparadas com os valores no arquivo de referência correspondente. Para cada medida, foi calculada a diferença entre o valor gerado e o obtido pelo processamento, e com estes valores de diferença, foi calculado a média e desvio padrão.

Os resultados do teste estão expostos na seção 4, onde cada combinação de torrada e material de correia tem duas tabelas, sendo uma para o vídeo sem ruído e a outra para o vídeo com ruído.

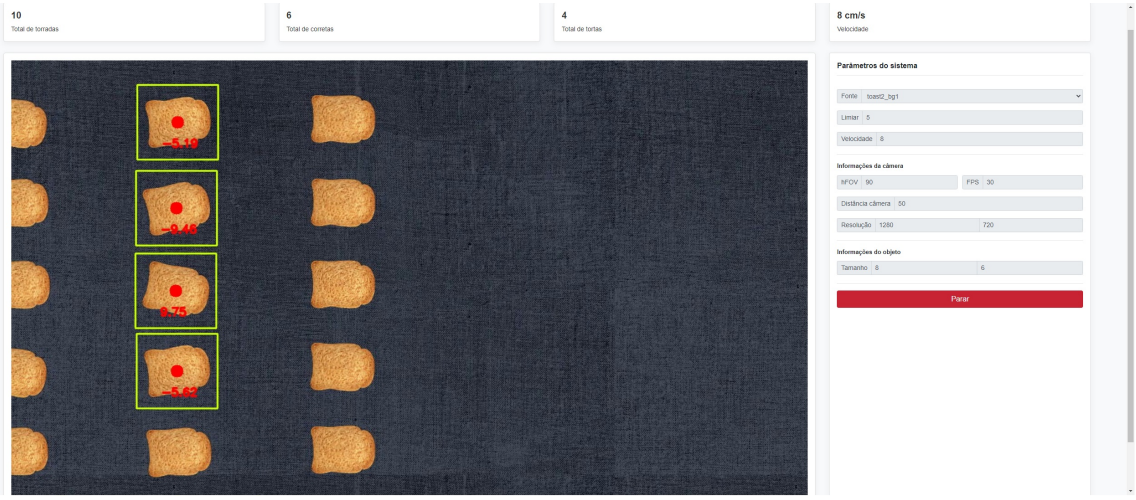
4 RESULTADOS

Nesta seção são apresentados os resultados obtidos no trabalho, o resultado do processamento mostrado na interface do usuário, e os resultados da precisão do sistema, testados com os vídeos criados de acordo com a seção 3.2.1.

4.1 Interface Web

A interface web mostra para o usuário o vídeo que está sendo transmitido pela câmera e a contagem das torradas, que inclui o total de torradas que passaram e a quantidade de torradas que estavam fora do padrão. Além disso, sobre o vídeo é colocado uma marcação sobre as torradas que estão fora de padrão, para auxiliar na visualização do funcionamento do sistema, onde é indicado sua posição e seu ângulo de rotação.

Figura 20 – Interface de Usuário



Fonte: elaborado pelo autor.

4.2 Teste de qualidade

Para todos os vídeos gerados, o sistema conseguiu identificar corretamente todas as torradas presentes, e para cada uma delas, foi calculada a diferença entre os valores medidos e os valores usados na sua criação. Tendo calculado este erro nos valores obtidos, utilizou-se seu valor absoluto para saber a média, desvio padrão e o valor máximo do erro das três características medidas, o ângulo, o comprimento e a altura das torradas.

4.2.1 Teste 1

O teste 1 foi realizado com os vídeos feitos com a Torrada da Figura 10 (a) e o Material da Figura 11 (a), que consiste de torradas claras e uma correia escura. As tabelas 2 e 3 indicam o desempenho do sistema, para o vídeo sem e com ruído, respectivamente.

Tabela 2 – Resultado do vídeo sem ruído do teste 1

parâmetro	média do erro	desvio padrão do erro	valor máximo do erro
â	1.26	0.63	2.93
comprimento(cm)	0.12	0.08	0.31
altura(cm)	0.19	0.05	0.28

Tabela 3 – Resultado do vídeo com ruído do teste 1

parâmetro	média do erro	desvio padrão do erro	valor máximo do erro
â	1.34	0.68	3.02
comprimento(cm)	0.08	0.06	0.23
altura(cm)	0.16	0.07	0.31

4.2.2 Teste 2

O teste 2 foi realizado com os vídeos feitos com Torrada da Figura 10 (a) e o Material da Figura 11 (b), que consiste de torradas claras e uma correia clara. As tabelas 4 e 5 indicam o desempenho do sistema, para o vídeo sem e com ruído, respectivamente.

Tabela 4 – Resultado do vídeo sem ruído do Teste2

parâmetro	média do erro	desvio padrão do erro	valor máximo do erro
â	1.46	0.45	2.67
comprimento(cm)	0.09	0.06	0.23
altura(cm)	0.15	0.07	0.31

Tabela 5 – Resultado do vídeo com ruído do Teste2

parâmetro	média do erro	desvio padrão do erro	valor máximo do erro
â	1.57	0.66	2.67
comprimento(cm)	0.08	0.06	0.31
altura(cm)	0.12	0.08	0.29

4.2.3 Teste 3

O teste 3 foi realizado com os vídeos feitos com Torrada da Figura 10 (b) e o Material da Figura 11 (a), que consiste de torradas escuras e uma correia escura. As tabelas 6 e 7 indicam o desempenho do sistema, para o vídeo sem e com ruído, respectivamente.

Tabela 6 – Resultado do vídeo sem ruído do teste 3

parâmetro	média do erro	desvio padrão do erro	valor máximo do erro
â	0.43	0.34	1.69
comprimento(cm)	0.19	0.07	0.33
altura(cm)	0.19	0.08	0.38

Tabela 7 – Resultado do vídeo com ruído do teste 3

parâmetro	média do erro	desvio padrão do erro	valor máximo do erro
â	0.43	0.31	1.00
comprimento(cm)	0.15	0.07	0.27
altura(cm)	0.15	0.08	0.38

4.2.4 Teste 4

O teste 4 foi realizado com os vídeos feitos com a Torrada da Figura 10 (b) e o Material da Figura 11 (b), que consiste de torradas escuras e uma correia clara. As tabelas 8 e 9 indicam o desempenho do sistema, para o vídeo sem e com ruído, respectivamente.

Tabela 8 – Resultado do vídeo sem ruído do teste 4

parâmetro	média do erro	desvio padrão do erro	valor máximo do erro
ângulo(grau)	0.27	0.23	0.93
comprimento(cm)	0.18	0.07	0.33
altura(cm)	0.18	0.07	0.35

Tabela 9 – Resultado do vídeo com ruído do teste 4

parâmetro	média do erro	desvio padrão do erro	valor máximo do erro
â	0.42	0.37	1.87
comprimento(cm)	0.16	0.08	0.35
altura(cm)	0.15	0.07	0.31

5 CONSIDERAÇÕES FINAIS

5.1 Conclusão

Neste trabalho, foi proposto um sistema de monitoramento de esteira industrial, com o intuito de automatizar a detecção de torradas e sinalizar as que estão fora dos padrões fornecidos pelo usuário, para possibilitar a remoção dos mesmos e assim evitar que estas torradas causem gargalos na linha de produção.

Dado os resultados apresentados na seção anterior, acredita-se que os objetivos propostos foram alcançados. Foi desenvolvido um sistema de processamento de imagens que conseguiu detectar todas as torradas presentes em todos os vídeos, e calcular o tamanho e o ângulo de rotação delas. O sistema também contou com uma interface de usuário para realizar a comunicação entre o usuário e o sistema, e informar ao usuário sobre dados obtidos no processamento, como a contagem de torradas fora do padrão e a posição delas na esteira.

Também foi possível avaliar a qualidade do sistema, comparando as informações obtidas para cada torrada com as informações originais das torradas, já que os vídeos usados foram criados artificialmente e as informações de cada torrada foram salvas para comparação.

E com os resultados vistos no capítulo anterior, pode-se inferir que o sistema teve um bom desempenho, conseguindo identificar todas as torradas presentes nas imagens dos vídeos e tendo um erro máximo de 3.02 na medição dos ângulos das torradas. Embora todas as torradas tenham sido detectadas, o sistema apresentou um melhor desempenho nos vídeos com as torradas mais escuras, onde teve os menores erros no cálculo do ângulo de rotação e tamanho das torradas.

Por fim, apesar do trabalho ter sido realizado com a problemática de uma linha de produção de torradas, sua metodologia de monitoramento pode ser utilizado de forma genérica para monitoração de objetos que estão se movendo em um único sentido, sendo feito as devidas alterações de parâmetros de entrada.

5.2 Limitações

Mesmo que o sistema atenda os objetivos propostos no trabalho e tenha obtido bons resultados, o desenvolvimento do sistema se deu em um ambiente extremamente controlado, sem ter variações na iluminação das imagens, nem imagens geradas com uma perspectiva diferente da vertical.

A aquisição de imagens que tenham variações destas condições é necessária para uma melhor validação do sistema, e a possível adição de novas capacidades para o mesmo.

5.3 Trabalhos Futuros

Há a oportunidade de estender o projeto para trabalhar com outros tipos de cenários, usando diferentes qualidades de iluminação; ângulos de gravação das imagen; e utilizar imagens obtidas a partir de gravações reais, além disso, o uso de técnicas de inteligência artificial, como redes neurais convolucionais, podem ser utilizadas no sistema para tornar mais robusto a capacidade de detecção e análise de objetos.

Uma outra possibilidade é uma integração com um sistema de internet das coisas, utilizando os dados obtidos para atuar na remoção automática dos objetos que estão fora do padrão.

REFERÊNCIAS

BRINK, H.; RICHARDS, J.; FETHEROLF, M. **Real-World Machine Learning**. Manning, 2016. ISBN 9781638357001. Disponível em: <https://books.google.com.br/books?id=zTczEAAAQBAJ>.

COSTA, L. D. F.; JR., R. M. C. **Shape analysis and classifcation - Theory and Practice**. [S.l.]: CRC Press LLC, 2000. v. 1rd.

COSTA, R. C. S. **Inspeção Automática de Laranjas Destinadas à Produção de Suco, Utilizando Técnicas de Processamento Digital de Imagens**. Dissertação (Graduação em Tecnologia Mecatrônica) — Centro Federal de Educação Tecnológica do Estado do Ceará, 2006.

FELIX, J.; CORTEZ, P.; HOLANDA, M. Automatic system for quantification and visualization of lung aeration on chest computed tomography images: The lung image system analysis - lisa. **Revista Brasileira de Engenharia Biomedica**, v. 26, p. 195–208, 01 2010.

FIRJAN SENAI. **5 motivos para investir na área de Automação Industrial**. 2020. Disponível em: <https://firjansenai.com.br/cursorio/blog/5-motivos-para-investir-na-area-de-automacao-industrial>. Acesso em: 18 jun. 2023.

FREEPIK. **Material Claro**. Disponível em: <https://www.freepik.com/free-photo/white-microfiber-fabric-background_4101429.htm>. Acesso em: 17 jun. 2023.

FREEPIK. **Material Escuro**. Disponível em: <https://www.freepik.com/free-photo/blue-painted-concrete-textured-background_17595813.htm>. Acesso em: 17 jun. 2023.

GOMES, O. D. F. M. **Processamento e Análise de Imagens Aplicados à Caracterização Automática de Materiais**. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, 2001.

GONZALEZ, R. C.; WOODS, R. E. **Processamento Digital de Imagens**. [S.l.]: Pearson Education, 2009. v. 3rd.

HASTING, G.; RUBIN, A. Colour spaces - a review of historic and modern colour models*. **African Vision and Eye Health**, v. 71, 12 2012.

IGLESIAS, J. C. Álvarez. **Uma Metodologia para Caracterização de Sínter de Minério de Ferro: Microscopia Digital e Análise de Imagens**. Tese (Doutorado) — Pontifícia Universidade Católica do Rio de Janeiro, 2008.

LONGFEI, Z.; ZHANG, L.; KONZ, N. Computer vision techniques in manufacturing. 12 2021.

MIGUEL GRINBERG. **Video Streaming with Flask**. 2023. Disponível em: <https://blog.miguelgrinberg.com/post/video-streaming-with-flask>. Acesso em: 17 jun. 2023.

OLIVEIRA, C. D. Q. **METODOLOGIA DE PROCESSAMENTO DIGITAL APLICADA À EXPLORAÇÃO DE IMAGENS RADIOLÓGICAS**. Dissertação (Mestrado) — INSTITUTO MILITAR DE ENGENHARIA, 2004.

OTSU, N. A. Threshold selection method from gray-level histograms. **IEEE Transactions on Systems, Man, and Cybernetics**, SMC-9, n. 1, p. 62–66, 1979.

PNGWING. **Torrada clara**. Disponível em: <https://www.pngwing.com/en/free-png-zarjf>. Acesso em: 17 jun. 2023.

PNGWING. **Torrada escura**. Disponível em: <https://www.pngwing.com/en/free-png-tlrmg>. Acesso em: 17 jun. 2023.

PORTAL DA INDÚSTRIA. **Indústria 4.0: Entenda seus conceitos e fundamentos**. 2023. Disponível em: <https://www.portaldaindustria.com.br/industria-de-a-z/industria-4-0/>. Acesso em: 17 jun. 2023.

RAPHAEL HOLZER. **OpenCV tutorial Documentation**. 2019. Disponível em: <https://opencv.org/about/>. Acesso em: 17 jun. 2023.

ROSER, C.; NAKANO, M.; TANAKA, M. Detecting shifting bottlenecks. p. 59–62, 06 2002.

SANTOS, T. T. e. a. Visão computacional aplicada na agricultura. **Portal Embrapa**, Cap 6, p. 19, 2020. Disponível em: <https://ainfo.cnptia.embrapa.br/digital/bitstream/item/217703/1/LV-Agricultura-digital-2020-cap6.pdf>.

SUZUKI, S.; ABE, K. Topological structural analysis of digitized binary images by border following. **COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING**, v. 30, n. 1, p. 32–46, 1985.

SZELISKI, R. **Computer Vision: Algorithms and Applications**. Springer International Publishing, 2022. (Texts in Computer Science). ISBN 9783030343729. Disponível em: <https://books.google.com.br/books?id=QptXEAAAQBAJ>.

TOTVS. **Linha de produção: principais gargalos e como resolvê-los**. 2018. Disponível em: <https://www.totvs.com/blog/gestao-industrial/linha-de-producao-gargalos/>. Acesso em: 24 jun. 2023.

WANG, L.; YU, L.; ZHU, J.; TANG, H.; GOU, F.; WU, J. Auxiliary segmentation method of osteosarcoma in mri images based on denoising and local enhancement. **Healthcare**, v. 10, n. 8, 2022. ISSN 2227-9032. Disponível em: <https://www.mdpi.com/2227-9032/10/8/1468>.

XIANGUO, L.; LIFANG, S.; ZIXU, M.; CAN, Z.; HANGQI, J. Laser-based on-line machine vision detection for longitudinal rip of conveyor belt. **Optik**, v. 168, p. 360–369, 2018. ISSN 0030-4026. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0030402618305461>.

APÊNDICE A – REQUERIMENTOS DO AMBIENTE PYTHON

Código-fonte 1 – Requerimentos do ambiente python

```
Click==7.0
cycler==0.10.0
decorator==4.4.1
Flask==1.1.1
imageio==2.6.1
imutils==0.5.3
itsdangerous==1.1.0
Jinja2==2.10.3
kiwisolver==1.1.0
MarkupSafe==1.1.1
matplotlib==3.1.1
networkx==2.4
numpy==1.17.3
opencv-python==4.1.1.26
Pillow==6.2.1
pyparsing==2.4.2
python-dateutil==2.8.1
PyWavelets==1.1.1
scikit-image==0.16.2
scipy==1.3.1
six==1.12.0
websockets==8.1
Werkzeug==0.16.0
```

APÊNDICE B – CÓDIGOS-FONTES UTILIZADOS PARA CRIAÇÃO DO VÍDEO

Código-fonte 2 – Classes utilizadas no Vídeo

```
1  import numpy as np
2  import cv2
3  import video_utils
4
5  class Frame():
6      noise_type = ""
7      def __init__(self, width, height, buffer_width,
           obj_width, line_distance, pixels_shift, bg_img):
8          self.affineMat = np.float32([
9                          [1, 0, 0],
10                         [0, 1, 0]
11                         ])
12         self.real_pixels_shift = pixels_shift
13         self.pixels_debt = 0
14         self.current_pixels_shifted = 0
15         self.bg_right_region_crop = 0
16         self.bg_img = bg_img
17         self.buffer_width = buffer_width
18         self.obj_insertion = int(obj_width*1.5)
19         self.line_distance = line_distance
20         self.objects_info =  ObjectInfo(len(line_distance),
               len(("angle", "width", "height")))
21         self.img = np.zeros([height, width + buffer_width ,
               3], dtype=np.uint8)
22         self.img, self.bg_right_region_crop = video_utils.
              add_background(self.img, self.bg_img, self.
              bg_right_region_crop ,
23
```

```
24
25          self.createObjectsAtBuffer()
26
27      def createObjectsAtBuffer(self):
28          self.listOfObjectsAtBuffer = []
29          for _ in self.line_distance:
30              obj = Object()
31              self.listOfObjectsAtBuffer.append(obj)
32          self.img = video_utils.add_objects(frame = self.img
                , objects= self.listOfObjectsAtBuffer, point_x =
                 self.obj_insertion, line_distance= self.
                line_distance)
33
34      def updateSpeed(self):
35          possible_shift = self.real_pixels_shift + self.
                pixels_debt
36          self.affineMat[0,2] = int(np.ceil(possible_shift))
37          self.pixels_debt = possible_shift - self.affineMat
                [0,2]
38
39      def update_pixels_count(self):
40          self.current_pixels_shifted += int(self.affineMat
                [0,2])
41      def shift_frame(self):
```

```
42          self.img = cv2.warpAffine(self.img, self.affineMat,
                (self.img.shape[1], self.img.shape[0]))
43      def update_frame(self):
44          self.shift_frame()
45          self.update_pixels_count()
46          self.updateSpeed()
47          if(self.current_pixels_shifted > self.buffer_width)
                :
48              self.img, self.bg_right_region_crop =
                    video_utils.add_background(self.img, self.
                    bg_img, self.bg_right_region_crop ,self.
                    current_pixels_shifted)
49              self.objects_info.updateInfo(self.
                    listOfObjectsAtBuffer)
50              self.current_pixels_shifted = 0
51              self.createObjectsAtBuffer()
52      def getImgROI(self, useNoise = False):
53          if(useNoise and Frame.noise_type):
54              return video_utils.addNoise(self.img[:,self.
                    buffer_width:,:], Frame.noise_type)
55          else:
56              return self.img[:,self.buffer_width:,:]
57
58  class Object():
59      img_variations = []
60      flip_prob = []
61      noise_position = 0
62      noise_size = 0
63      norm_params = 0
64      norm_weights = 0
65      width = 104
66      height = 78
```

```
67      def __init__(self):
68          self.toast_flip = np.random.choice(range(len(Object
                .img_variations)), p=Object.flip_prob)
69          self.img = Object.img_variations[self.toast_flip]
70          self.vertical_noise_size = round(np.random.uniform
                (-Object.noise_size, Object.noise_size, 1)[0])
71          self.horizontal_noise_size = round(np.random.
                uniform(-Object.noise_size, Object.noise_size,
                1)[0])
72          self.horizontal_noise_displacement = round(np.
                random.uniform(-Object.noise_position, Object.
                noise_position, 1)[0])
73          self.vertical_noise_displacement = round(np.random.
                uniform(-Object.noise_position, Object.
                noise_position, 1)[0])
74          self.rotation_degree = round(video_utils.
                mixture_gaussian(Object.norm_params, Object.
                norm_weights), 2)
75
76          self.width = int(Object.width + self.
                horizontal_noise_size)
77          self.height = int(Object.height + self.
                vertical_noise_size)
78
79          self.img = cv2.resize(Object.img_variations[self.
                toast_flip],
80                                (self.width,
81                                 self.height),
82                                interpolation=cv2.
                                    INTER_AREA)
83          self.img_rotated = video_utils.rotate_img(Object.
                img_variations[self.toast_flip], self.
```

```
            rotation_degree)
84          self.x_offset = int(self.horizontal_noise_size)
85          self.y_offset = int(self.vertical_noise_size)
86      def getPropsForArray(self):
87          return [ self.rotation_degree, self.width, self.
                height]
88      def getImg(self):
89          return self.img_rotated
90
91  class ObjectInfo():
92      def __init__(self, numOfItens = 5, numOfProps = 3):
93          self.numOfItens = numOfItens
94          self.numOfProps = numOfProps
95          self.info = np.zeros([self.numOfItens, 0, self.
                numOfProps])
96
97      def updateInfo(self, objects):
98          array = np.zeros([self.numOfItens, 1, self.
                numOfProps])
99          for i in range(self.numOfItens):
100             info = objects[i].getPropsForArray()
101             array[i,:,:] = info
102         self.info = np.append(self.info, array, 1)
103
104     def getInfo(self) -> np.array:
105         return self.info
106     def saveToFile(self, path = ''):
107         if(path is ''):
108             path = 'objectsInfo.txt'
109         array = self.info.reshape(self.numOfItens, -1)
110         np.savetxt(path, array)
111     def getFromFile(self, path, delimiter = None):
```

```
112         array =  np.loadtxt(path, delimiter)
113         self.info = array.reshape(self.numOfItens, int(
               array.shape[1]/ self.numOfProps), self.
               numOfProps)
```

Código-fonte 3 – Utils

```
1  import numpy as np
2  import cv2
3  import imutils
4  import skimage
5  def mixture_gaussian(norm_params, norm_weights):
6      mixture_id = np.random.choice(len(norm_weights), size
          =1, replace=True, p=norm_weights)[0]
7      return np.random.normal(*(norm_params[mixture_id]))
8
9  def addNoise(img, noise_type):
10     noisy = skimage.util.random_noise(img, mode=noise_type)
11     img = cv2.normalize(noisy, None, 0, 255, cv2.
          NORM_MINMAX, cv2.CV_8U)
12     return img
13
14 def rotate_img(img, degree):
15     return imutils.rotate_bound(img, degree)
16
17 def overlay_object_in_images(frame, img, y_i, x_f):
18     frame_height, frame_width, _ = frame.shape
19     img_height, img_width, num_of_channels = img.shape
20
21     frame_y_i = y_i
22     frame_y_f = min(frame_height, y_i + img_height)
23     img_y_i = 0
```

```
24      img_y_f = frame_y_f - y_i
25
26      frame_x_f =  x_f
27      frame_x_i = x_f - img_width if (x_f - img_width) > 0
            else 0
28      img_x_f = img_width
29      img_x_i = 0
30
31      if (num_of_channels == 4):
32          img_alpha = img[img_y_i:img_y_f, img_x_i:img_x_f,
                3] / 255.0
33          frame_alpha = (1.0 - img_alpha)
34          for channel in range(3):
35              frame[frame_y_i:frame_y_f, frame_x_i:frame_x_f,
                    channel] = (img_alpha*img[img_y_i:img_y_f,
                    img_x_i:img_x_f, channel] +
36                                          frame_alpha*frame[
                                                frame_y_i:
                                                frame_y_f,
                                                frame_x_i:
                                                frame_x_f,
                                                channel])
37      else:
38          frame[frame_y_i:frame_y_f, frame_x_i:frame_x_f, :]
                = img[img_y_i:img_y_f, img_x_i:img_x_f, :]
39      return frame
40
41  def add_background(frame, bg_img, bg_right_region_crop,
        bg_insertion_column):
42      frame_height, _, _ = frame.shape
43      bg_height, bg_width, _  = bg_img.shape
44      _bg_insertion_column = bg_insertion_column
```

```
45      _bg_right_region_crop = bg_right_region_crop
46      bg_left_region_crop = _bg_right_region_crop -
           _bg_insertion_column if(_bg_right_region_crop >
           _bg_insertion_column ) else 0
47      while _bg_insertion_column > 0:
48          for i in range(0,frame_height,bg_height):
49              frame = overlay_object_in_images(frame, bg_img
                   [:,bg_left_region_crop:_bg_right_region_crop
                   ], i, _bg_insertion_column)
50
51          _bg_insertion_column = _bg_insertion_column - (
               _bg_right_region_crop - bg_left_region_crop)
52          _bg_right_region_crop = bg_width if (
               bg_left_region_crop  == 0) else
               bg_left_region_crop
53          bg_left_region_crop = _bg_right_region_crop -
               _bg_insertion_column if(_bg_insertion_column <=
               _bg_right_region_crop) else 0
54
55      return frame, _bg_right_region_crop
56
57
58  def add_objects(frame, objects, point_x, line_distance) ->
       np.array:
59      for i, y in enumerate(line_distance):
60          x = objects[i].x_offset + point_x
61          y = objects[i].y_offset + y
62          overlay_object_in_images(frame, objects[i].getImg()
               , y, x)
63      return frame
```

Código-fonte 4 – Geração do Vídeo

```
import os
import cv2
import numpy as np
from video_classes import Object, Frame

fov = 90

fps = 30
video_width = 1280
video_height = 720
video_duration = 26

object_width = 8
object_height = 6

dist_camera_to_conveyor = 50
conveyor_speed = 8
objects_per_column = 5
h_dist_between_objects = 4

Frame.noise_type = "gaussian"
noise_object_position = (
    0.3
)
noise_object_size = 0.1
Object.norm_params = np.array(
    [[-10, 4], [0, 4], [10, 4]]
)
Object.norm_weights = [
    (1.0 / 10.0),
    (8.0 / 10.0),
    (1.0 / 10.0),
```

```
]
Object.flip_prob = [
    1 / 2,
    1 / 2,
]

_conveyor_width = int(
    round(2 * dist_camera_to_conveyor * np.tan((np.pi * (
        fov / 180)) / 2))
)
_pixels_per_centimeter = round(video_width /
    _conveyor_width)
Object.witdh = round(object_width * _pixels_per_centimeter)
Object.height = round(object_height *
    _pixels_per_centimeter)
_conveyor_speed_px = conveyor_speed *
    _pixels_per_centimeter

_horizontal_distance_between_columns_in_pixels = (
    object_width + h_dist_between_objects
) * _pixels_per_centimeter

_v_distance_between_lines = int(
    np.ceil((video_height + Object.height) / (
        objects_per_column + 1))
)

_line_distance = range(
    _v_distance_between_lines - Object.height, video_height
        , _v_distance_between_lines
)

```

```
59 _buffer_width =
      _horizontal_distance_between_columns_in_pixels + Object.
      witdh
60
61 _pixels_shift_per_frame = _conveyor_speed_px / fps
62 Object.noise_position = noise_object_position *
      _pixels_per_centimeter
63 Object.noise_size = noise_object_size *
      _pixels_per_centimeter
64
65 _toasts_assets = ['toast1', 'toast2']
66 object_name = _toasts_assets[0]
67 _object = cv2.imread('../assets/images/object/' +
      object_name + '.png', cv2.IMREAD_UNCHANGED)
68
69 _bg_assets = ['bg1', 'bg2' ]
70 background_name = _bg_assets[1]
71 _bg_pattern_image = cv2.imread('../assets/images/background
      /' + background_name + '.jpg')
72
73
74 _object_resized = cv2.resize(_object, (Object.witdh, Object
      .height), interpolation=cv2.INTER_AREA)
75 Object.img_variations = [
76     _object_resized,
77     cv2.flip(_object_resized, 0),
78 ]
79
80 fourcc = cv2.VideoWriter_fourcc(*"MP42")
81 _output_video_name = "./videos/" + object_name +  "_" +
      background_name
82 _output_image_name = _output_video_name + "_img.png"
```

```
83 _video_filename = _output_video_name + ".avi"
84 video = cv2.VideoWriter(
85     _video_filename, fourcc, float(fps), (video_width,
       video_height)
86 )
87
88
89 noiseName = "_"+Frame.noise_type
90 _output_imageWithNoise_name = _output_video_name +
     noiseName + "_img.png"
91 _videoWithNoise_directory = _output_video_name + noiseName
     + ".avi"
92
93 videoWithNoise = cv2.VideoWriter(
94     _videoWithNoise_directory, fourcc, float(fps), (
       video_width, video_height)
95 )
96
97 def print_progress_bar(iteration, total, prefix='', suffix=
     '', decimals=1, length=100, fill='*', print_end="\r"):
98     percent = ("{0:." + str(decimals) + "f}").format(100 *
       (iteration / float(total)))
99     filled_length = int(length * iteration // total)
100    bar = fill * filled_length + '-' * (length -
       filled_length)
101    print('\r%s |%s| %s%% %s' % (prefix, bar, percent,
       suffix), end=print_end)
102    if iteration == total:
103        print()
104
105 def save_parameters():
106    save_current_line = False
```

```
107     with open(_output_video_name + "_videoInfo.txt", "a")
            as f1:
108         for line in open("video_generator.py"):
109             if "#%#" in line and not save_current_line:
110                 save_current_line = True
111             elif "#%#" in line and save_current_line:
112                 f1.write(line)
113                 break
114             if save_current_line:
115                 f1.write(line)
116
117
118 save_parameters()
119 frame = Frame(width = video_width, height = video_height,
        buffer_width = _buffer_width,
120             obj_width = Object.witdh, line_distance =
                _line_distance,
121             pixels_shift = _pixels_shift_per_frame,
122             bg_img = _bg_pattern_image)
123
124 for f in range(fps * video_duration):
125
126     print_progress_bar(
127         f, fps * video_duration - 1, prefix="Progress:",
            suffix="Complete", length=70
128     )
129     frame.update_frame()
130     if(f==120):
131         imageAtFrame120 = cv2.imwrite(_output_image_name,
            frame.getImgROI())
132         imageWithNoiseAtFrame120 = cv2.imwrite(
            _output_imageWithNoise_name, frame.getImgROI(
```

```
            useNoise=True))
133
134     video.write(frame.getImgROI())
135     videoWithNoise.write(frame.getImgROI(useNoise=True))
136
137 frame.objects_info.saveToFile(_output_video_name+"
       _objectInfo.txt")
138 video.release()
139 videoWithNoise.release()
```

APÊNDICE C – CÓDIGOS-FONTES UTILIZADO PARA O SERVIDOR

Código-fonte 5 – Comunicação entre servidor e interface

```
import asyncio
import json
import logging
import socket

import cv2
import time
import threading
import numpy as np
from toast import Toast

from steps import imageAnalysis

from flask import Response
from flask import Flask
from flask import render_template

logging.basicConfig()

serverIP = socket.gethostbyname(socket.gethostname())
output_frame = None

lock = threading.Lock()
lock_state = threading.Lock()
app = Flask(__name__)

STATE = {
    "active": False,
    "video_src": "",
```

```
    "total_toasts_count": 0,
    "total_target_toasts_count": 0,
    "target_toasts_at_screen": 0,
    "velocity": 0,
    "input_video": [""],
}

USERS = set()

class VariablesManager:
    def __init__(self):
        self.lock = threading.Lock()

        self.fps = None
        self.fov = None
        self.width = None
        self.height = None
        self.camera_distance = None

        self.object_width = None
        self.object_height = None

        self.rotation_threshold = None
        self.roi_x_initial = None
        self.roi_x_final = None

        self._conveyor_width = None
        self._pixels_per_centimeter = None
        self._object_width_px = None
        self._object_height_px = None

```

```
61      def set_info(self, rth, fps, fov, width, height,
            camera_dist, object_w, object_h):
62          with self.lock:
63              self.rotation_threshold = float(rth)
64              self.fps = int(fps)
65              self.fov = int(fov)
66              self.width = int(width)
67              self.height = int(height)
68              self.camera_distance = float(camera_dist)
69              self.object_width = float(object_w)
70              self.object_height = float(object_h)
71
72              self._conveyor_width = int(
73                  round(2 * float(camera_dist) * np.tan((np.
                        pi / 180 * float(fov)) / 2))
74              )
75
76              self._pixels_per_centimeter = round(int(width)
                    / self._conveyor_width)
77              self._object_width_px = round(float(object_w) *
                     self._pixels_per_centimeter)
78              self._object_height_px = round(
79                  float(object_h) * self.
                        _pixels_per_centimeter
80              )
81              self.roi_x_initial = 0
82              self.roi_x_final = int(
83                  self.roi_x_initial + 1.2 * max(self.
                        _object_width_px, self._object_height_px
                        )
84              )
85      def get_info(self, attr):
```

```
86          with self.lock:
87              if hasattr(self, attr):
88                  return getattr(self, attr)
89              else:
90                  return None
91
92      def convert_velocity_to_cm_s(self, v):
93          if self.fps:
94              return v * (self.fps / self.
                    _pixels_per_centimeter)
95          else:
96              return 0
97
98  VM = VariablesManager()
99
100 def run(processed_data, original_image):
101     global output_frame
102
103     STATE["total_toasts_count"] = processed_data["
            total_toasts_count"]
104     STATE["total_target_toasts_count"] = processed_data["
            total_target_toasts_count"]
105     STATE["target_toasts_at_screen"] = processed_data["
            target_toasts_at_screen"]
106
107     for toast in processed_data["target_toasts"]:
108         centroid = toast.centroid
109         cv2.circle(original_image, (centroid[0], centroid
                [1]), 10, (0, 0, 255), -1)
110
111         cv2.rectangle(
112             original_image,
```

```
113             (centroid[0] - int(VM._object_width_px*1.1/2),
                    centroid[1] - int(VM._object_width_px*1.1/2)
                    ),
114             (centroid[0] + int(VM._object_width_px*1.1/2),
                    centroid[1] + int(VM._object_width_px*1.1/2)
                    ),
115             (0, 255, 200),
116             2,
117         )
118         cv2.putText(
119             original_image,
120             str(toast.angle),
121             (centroid[0] - 25, centroid[1] + 40),
122             cv2.FONT_HERSHEY_SIMPLEX,
123             0.6,
124             (0, 0, 255),
125             2,
126             cv2.LINE_AA,
127         )
128
129     with lock:
130         output_frame = original_image
131
132     loop = asyncio.new_event_loop()
133     asyncio.set_event_loop(loop)
134     loop.run_until_complete(notify_state())
135     loop.close()
136
137 def state_event():
138     return json.dumps({"type": "state", **STATE})
139
140 async def notify_state():
```

```
141     if USERS:
142         message = state_event()
143         await asyncio.wait([user.send(message) for user in
                USERS])
144
145 async def register(websocket):
146     USERS.add(websocket)
147
148 async def unregister(websocket):
149     USERS.remove(websocket)
150
151 async def counter(websocket, _):
152     global STATE
153
154     await register(websocket)
155     try:
156         await websocket.send(state_event())
157         async for message in websocket:
158             data = json.loads(message)
159             if data["active"]:
160                 with lock_state:
161                     STATE["active"] = True
162                     STATE["video_src"] = data["fonte"]
163                     VM.set_info(
164                         data["limiar"],
165                         data["fps"],
166                         data["fov"],
167                         data["video_resolution_width"],
168                         data["video_resolution_height"],
169                         data["dist"],
170                         data["obj_width"],
171                         data["obj_height"],
```

```
172                     )
173                     STATE["velocity"] = round((float(data["
                            velocity"])*VM.
                            _pixels_per_centimeter)/VM.fps,2)
174                     Toast.updatePixelsToShift(speed=STATE["
                            velocity"])
175
176                     imageAnalysis.reset_variaveis()
177                 await notify_state()
178             elif not data["active"]:
179                 with lock_state:
180                     STATE["active"] = False
181                     STATE["velocity"] = 0
182                 await notify_state()
183             else:
184                 logging.error("unsupported event: {}", data
                        )
185     finally:
186         await unregister(websocket)
187
188 def generate():
189     global output_frame, lock
190
191     while True:
192         time.sleep(0.005)
193         with lock:
194
195             if output_frame is None:
196                 continue
197
198             (flag, encodedImage) = cv2.imencode(".jpg",
                    output_frame)
```

```
199
200             if not flag:
201                 continue
202
203         yield (
204             b"--frame\r\n"
205             b"Content-Type: image/jpeg\r\n\r\n" + bytearray
                    (encodedImage) + b"\r\n"
206         )
207
208 @app.route("/")
209 def index():
210     return render_template("index.html", serverIP=serverIP)
211
212 @app.route("/video_feed")
213 def video_feed():
214     return Response(generate(), mimetype="multipart/x-mixed
        -replace; boundary=frame")
215
216 def launch_web_server():
217     app.run(host="0.0.0.0", port=80, debug=False, threaded=
        True, use_reloader=False)
```

Código-fonte 6 – Comunicação entre os processos de interface e PDI(interface)

```
1 import os
2 import cv2
3 import asyncio
4 import websockets
5 from queue import Queue
6 from threading import Thread
7
```

```
8  from steps import imageProcess, imageAnalysis
9  import interface
10
11 def process(roiImage):
12     postProcessedImage = imageProcess.run(roiImage)
13     data = imageAnalysis.run(postProcessedImage)
14     return data
15 originalFrameQueue = Queue()
16 roiFrameQueue = Queue()
17
18 imageProcessingQueue = Queue()
19
20 videoAssetsPath = "../assets/videos"
21
22 def process_data(processing_function, input_queue,
      output_queue, auxiliary_queue=None):
23     while True:
24         input_data = input_queue.get()
25
26         if auxiliary_queue:
27             auxiliary_input = auxiliary_queue.get()
28             if output_queue:
29                 output_data = processing_function(
                      input_data, auxiliary_input)
30                 output_queue.put(output_data)
31             else:
32                 processing_function(input_data,
                      auxiliary_input)
33         else:
34             if output_queue:
35                 output_data = processing_function(
                      input_data)
```

```
36                  output_queue.put(output_data)
37              else:
38                  processing_function(input_data)
39
40          input_queue.task_done()
41
42  web_serverThread = Thread(target=interface.
       launch_web_server)
43  web_serverThread.setDaemon(True)
44  web_serverThread.start()
45
46  imageProcessingThread = Thread(
47      target=process_data,
48      args=(process,
49      roiFrameQueue,
50      imageProcessingQueue),
51
52  )
53  imageProcessingThread.setDaemon(True)
54  imageProcessingThread.start()
55
56  interfaceThread = Thread(
57      target=process_data,
58      args=(interface.run,
59      imageProcessingQueue,
60      None,
61      originalFrameQueue),
62  )
63  interfaceThread.setDaemon(True)
64  interfaceThread.start()
65
66  videoSource = [""]
```

```
67
68 for file in os.listdir(videoAssetsPath):
69     if file.endswith(".avi"):
70         videoSource.append(file.split(".")[0])
71
72 interface.STATE["input_video"] = videoSource
73
74 def video_player():
75     while True:
76         with interface.lock_state:
77             active = interface.STATE["active"]
78             source_name = interface.STATE["video_src"]
79
80         if not active:
81             cv2.waitKey(15)
82             continue
83         else:
84
85             source_name = os.path.join(videoAssetsPath,
                source_name + ".avi")
86             input_video = cv2.VideoCapture(source_name)
87             roi_x_initial = interface.VM.get_info("
                roi_x_initial")
88             roi_x_final = interface.VM.get_info("
                roi_x_final")
89
90             while input_video.isOpened():
91                 with interface.lock_state:
92                     if not interface.STATE["active"]:
93                         break
94
95                 ret, frame = input_video.read()
```

```
96
97                  if ret:
98                      roi = frame[:, roi_x_initial:
                           roi_x_final]
99                      roiFrameQueue.put(roi)
100                     originalFrameQueue.put(frame)
101                 else:
102                     with interface.lock_state:
103                         interface.STATE["active"] = False
104
105                         loop = asyncio.new_event_loop()
106                         asyncio.set_event_loop(loop)
107                         loop.run_until_complete(interface.
                               notify_state())
108                         loop.close()
109                     break
110
111                 if cv2.waitKey(33) & 0xFF == ord("q"):
112                     with interface.lock_state:
113                         interface.STATE["active"] = False
114
115                         loop = asyncio.new_event_loop()
116                         asyncio.set_event_loop(loop)
117                         loop.run_until_complete(interface.
                               notify_state())
118                         loop.close()
119                     break
120
121             input_video.release()
122
123 video_playerThread = Thread(target=video_player)
124 video_playerThread.setDaemon(True)
```

```
125 video_playerThread.start()
126
127 start_server = websockets.serve(interface.counter, "",
       6789)
128 asyncio.get_event_loop().run_until_complete(start_server)
129 asyncio.get_event_loop().run_forever()
```

APÊNDICE D – CÓDIGOS-FONTES UTILIZADOS PARA O PROCESSAMENTO DIGITAL DE IMAGEM

Código-fonte 7 – Classe da Torrada(Toast)

```
import numpy as np
class Toast:
    speed_debt = 0
    pixelsToShift = 0
    def updatePixelsToShift(speed:float) -> None:
        supposedShift = speed + Toast.speed_debt
        Toast.pixelsToShift = int(np.ceil(supposedShift))
        Toast.speed_debt = supposedShift - Toast.
           pixelsToShift

        return None
    def __init__(self, box=0, angle=0, centroid=[0,0], alvo
       =False):
        self.box = box
        self.angle = angle
        self.centroid = centroid
        self.alvo = alvo

    def isTarget(self):
        return self.alvo
```

Código-fonte 8 – Processamento da Imagem(imageProcess)

```
import numpy as np
import cv2

def preProcessing(image):
    hlsImg = cv2.cvtColor(image, cv2.COLOR_BGR2HLS)
```

```
    saturationImg = hlsImg[:, :, 2]

    medianBlurKernel = 3
    medianBlurImg = cv2.medianBlur(saturationImg,
       medianBlurKernel)

    blurKernel = 5
    preProcessedImage = cv2.blur(medianBlurImg, (blurKernel
       , blurKernel))

    return preProcessedImage

def segmentation(preProcessedImage):
    _, segmentedImage = cv2.threshold(preProcessedImage, 0,
        1, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    return segmentedImage

def postProcessing(segmentedImage):
    kernelOpen = np.ones((5,5),np.uint8)
    kernelClose = np.ones((10,10),np.uint8)
    postProcessedImage = cv2.morphologyEx(segmentedImage.
       copy(), cv2.MORPH_OPEN, kernelOpen)
    postProcessedImage = cv2.morphologyEx(
       postProcessedImage, cv2.MORPH_CLOSE, kernelClose)
    return postProcessedImage

def run(roiImage):
    preProcessedImage = preProcessing(roiImage)
    segmentedImage = segmentation(preProcessedImage)
    postProcessedImage = postProcessing(segmentedImage)
    return postProcessedImage
```

Código-fonte 9 – Análise da Imagem(imageAnalysis)

```
import numpy as np
import math
import cv2
import interface
from toast import Toast
from typing import List

list_toasts_at_screen:List[Toast] = []

total_target_toasts_count = 0

total_toasts_count = 0

target_toasts_at_screen = 0

def reset_variaveis():
    global list_toasts_at_screen, total_target_toasts_count
       , total_toasts_count, target_toasts_at_screen
    list_toasts_at_screen = []
    total_target_toasts_count = 0
    total_toasts_count = 0
    target_toasts_at_screen = 0

def adjustAngle(box, centro, angle):
    adjustedAngle = 0
    if box[0][0] > centro[0]:

        adjustedAngle = 90 + angle
    else:
        adjustedAngle = angle
    return round(adjustedAngle, 2)
```

```
31
32 def hasAToastDensity(img, cx, cy, width):
33     acceptedDensity = 0.5
34     y_i = int(cy-(width*1.1/2))
35     y_f = int(cy+(width*1.1/2))
36     x_i = int(cx-(width*1.1/2))
37     x_f = int(cx+(width*1.1/2))
38     toastArea = img[y_i:y_f,x_i:x_f]
39     w,h = toastArea.shape
40     area = (w*h)
41     if(area == 0):
42         return False
43     toastDensity = toastArea.sum()/(w*h)
44     return toastDensity>acceptedDensity
45
46 def isNewToast(centroids, toasts:List[Toast]):
47     isNew = True
48     distance_limit = 4*interface.STATE['velocity']
49     for toast in toasts:
50         centroid = toast.centroid
51
52         dist = math.sqrt(
53             (centroid[0] - centroids[0]) ** 2
54             + (centroid[1] - centroids[1]) ** 2
55         )
56
57         if dist < distance_limit:
58             isNew = False
59             break
60
61     return isNew
62
```

```
63 def update_toasts(list_toasts_at_screen:List[Toast]):
64     if len(list_toasts_at_screen) == 0:
65         return list_toasts_at_screen
66 
67     Toast.updatePixelsToShift(speed= interface.STATE['
       velocity'])
68     pixelsToShift = Toast.pixelsToShift
69     for toast in list_toasts_at_screen:
70         centroid = toast.centroid
71 
72         centroid[0] = centroid[0] + pixelsToShift
73 
74         if centroid[0] >= interface.VM.get_info("width"):
75             list_toasts_at_screen.remove(toast)
76         else:
77             toast.centroid = centroid
78 
79     return list_toasts_at_screen
80 
81 def isAnAcceptableMeasure(idealMeasure, obtainedMeasure) ->
    bool:
82     if(1.05* idealMeasure > obtainedMeasure > 0.95*
       idealMeasure):
83         return True
84     else:
85         return False
86 
87 def findToasts(img, alfa, prev_toasts):
88     roiOffset = interface.VM.get_info("roi_x_initial")
89     idealWidth = interface.VM.get_info("_object_width_px")
90     idealHeight = interface.VM.get_info("_object_height_px"
       )
```

```
91
92      global total_target_toasts_count
93      global total_toasts_count
94
95      contours, _ = cv2.findContours(img, cv2.RETR_TREE, cv2.
            CHAIN_APPROX_SIMPLE)
96
97      num_toasts = len(contours)
98
99      for i in range(num_toasts):
100
101         rect = cv2.minAreaRect(contours[i])
102         box = cv2.boxPoints(rect)
103         box = np.int64(box)
104         cX = int(box[:,0].sum()/4)
105         cY = int(box[:,1].sum()/4)
106         a = float(rect[2])
107         angle = adjustAngle(box, [cX, cY], a)
108
109         object_width, object_height = max(rect[1][0], rect
                [1][1]), min(rect[1][0], rect[1][1])
110
111         if (isAnAcceptableMeasure(idealWidth,object_width)
                and isAnAcceptableMeasure(idealHeight,
                object_height)):
112
113             isToastDensity = hasAToastDensity(img, cX, cY,
                    object_width)
114             isNew = isNewToast([cX + roiOffset, cY],
                    prev_toasts)
115
116             if isToastDensity and isNew:
```

```
117                 total_toasts_count += 1
118
119                 centroid = [cX + roiOffset, cY]
120
121                 box[:, 0] = box[:, 0] + roiOffset
122
123                 toast = Toast(box, angle, centroid)
124                 if abs(angle) >= alfa:
125                     total_target_toasts_count += 1
126                     toast = Toast(box, angle, centroid,
                           True)
127                 else:
128                     toast = Toast(box, angle, centroid,
                           False)
129
130                 prev_toasts.append(toast)
131
132     return prev_toasts
133
134 def run(postProcessedImage):
135     global list_toasts_at_screen
136     global total_target_toasts_count
137     global target_toasts_at_screen
138     global total_toasts_count
139     alfa = interface.VM.get_info("rotation_threshold")
140     all_toasts = update_toasts(list_toasts_at_screen)
141     all_toasts = findToasts(postProcessedImage, alfa,
           all_toasts)
142
143     target_toasts = []
144     target_toasts_at_screen = 0
145     for toast in all_toasts:
```

```
146         if toast.isTarget():
147             target_toasts_at_screen += 1
148             target_toasts.append(toast)
149 
150     return {
151         "target_toasts": target_toasts,
152         "total_toasts_count": total_toasts_count,
153         "total_target_toasts_count":
              total_target_toasts_count,
154         "target_toasts_at_screen": target_toasts_at_screen,
155     }
```

APÊNDICE E – CÓDIGO-FONTE UTILIZADO A INTERFACE DE USUÁRIO

Código-fonte 10 – Página da Interface de Usuário

```
<html lang="en">

<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-
    scale=1, shrink-to-fit=no">
<meta name="description" content="">
<meta name="author" content="">
<link rel="icon" href="{{ url_for('static', filename='img/
    toast.ico') }}">

<link href="{{ url_for('static', filename='css/bootstrap.
    min.css') }}" rel="stylesheet">
<link href="{{ url_for('static', filename='css/style.css')
    }}" rel="stylesheet">
</head>

<body>
<main role="main">
<div class="album py-5 bg-light">
<div class="container-fluid">
<div class="row">
<div class="col-lg-3 col-md-6 col-sm-6 col-xs-12 mb-4">
<div class="card box-shadow">
<div class="d-flex justify-content-around">
<div class="website-infos">
<h2><span class="counter total_toasts_count"></span></h2>
<p>Total de torradas</p>
</div>
```

```
26 </div>
27 </div>
28 </div>
29 <div class="col-lg-3 col-md-6 col-sm-6 col-xs-12 mb-4">
30 <div class="card box-shadow">
31 <div class="d-flex justify-content-around">
32 <div class="website-infos">
33 <h2><span class="counter total_ok_toasts_count"></span></h2
   >
34 <p>Total de corretas</p>
35 </div>
36 </div>
37 </div>
38 </div>
39 <div class="col-lg-3 col-md-6 col-sm-6 col-xs-12 mb-4">
40 <div class="card box-shadow">
41 <div class="d-flex justify-content-around">
42 <div class="website-infos">
43 <h2><span class="counter total_target_toasts_count"></span>
   </h2>
44 <p>Total de tortas</p>
45 </div>
46
47 </div>
48 </div>
49 </div>
50 <div class="col-lg-3 col-md-6 col-sm-6 col-xs-12 mb-4">
51 <div class="card box-shadow">
52 <div class="d-flex justify-content-around">
53 <div class="website-infos">
54 <h2><span class="counter velocity">0 cm/s</span></h2>
55 <p>Velocidade</p>
```

```
</div>
</div>
</div>
</div>
</div>
<div class="row" id="items_container">
<div class="col-xl-9 col-lg-12 col-md-12 col-sm-12 col-xs-12 mb-4">
<div class="card box-shadow">
<img src="{{ url_for('video_feed') }}" style="width: 100%; height: auto;">
</div>
</div>
<div class="col-xl-3 col-lg-12 col-md-12 col-sm-12 col-xs-12 mb-4">
<div class="h-100">
<div class="h-100 card box-shadow">
<div>
<strong>Parametros do sistema</strong>
</div>
<hr style="margin: 8px 0 30px 0">
<form action="#" onsubmit="iniciar();return false;">
<fieldset id="fieldset" disabled>
<div class="form-row">
    <div class="form-group col-xl-12 col-md-6">
        <div class="input-group input-group-sm">
            <div class="input-group-prepend">
                <span class="input-group-text" id="video_input2">Fonte</span>
            </div>
            <select class="form-control video_input" id="video_input">
```

```
83                  <option>Camera real</option>
84                  <option>Simulacao 1</option>
85                  <option>Simulacao 2</option>
86                  <option>Simulacao 3</option>
87                  <option>Simulacao 4</option>
88              </select>
89          </div>
90      </div>
91      <div class="form-group col-xl-12 col-md-6">
92          <div class="input-group input-group-sm">
93              <div class="input-group-prepend">
94                  <span class="input-group-text" id="limiar2"
                        >Limiar</span>
95              </div>
96              <input type="text" class="form-control" id="
                    limiar_input"
97                  aria-label="Small" aria-describedby="
                        inputGroup-sizing-sm"
98                  placeholder="angulo (grau)" required>
99          </div>
100     </div>
101     <div class="form-group col-xl-12 col-md-6">
102         <div class="input-group input-group-sm">
103             <div class="input-group-prepend">
104                 <span class="input-group-text"
105                     id="velocidade2">Velocidade</span>
106             </div>
107             <input type="text" class="form-control" id="
                    velocidade_input"
108                 aria-label="Small" aria-describedby="
                        inputGroup-sizing-sm"
109                 placeholder="cm/s" required>
```

```
110         </div>
111     </div>
112 </div>
113 <hr style="margin: 2px 0 15px 0">
114 <strong style="font-size: 13px  ;">Informacoes da camera</
    strong>
115 <div class="form-row" style="margin-top: 10px;">
116     <div class="form-group col-md-6">
117         <div class="input-group input-group-sm">
118             <div class="input-group-prepend">
119                 <span class="input-group-text" id="fov2">
                        hFOV</span>
120             </div>
121             <input type="text" class="form-control" id="
                    fov_input"
122                 placeholder="horizontal field of view"
                        required>
123         </div>
124     </div>
125     <div class="form-group col-md-6">
126         <div class="input-group input-group-sm">
127             <div class="input-group-prepend">
128                 <span class="input-group-text" id="fps2">
                        FPS</span>
129             </div>
130             <input type="text" class="form-control" id="
                    fps_input"
131                 placeholder="frames per second" required>
132         </div>
133     </div>
134     <div class="form-group col-md-12">
135         <div class="input-group input-group-sm">
```

```
136             <div class="input-group-prepend">
137                 <span class="input-group-text" id="dist2">
                        Distancia
138                     camera</span>
139             </div>
140             <input type="text" class="form-control" id="
                    dist_input"
141                 placeholder="distancia (cm)" required>
142         </div>
143     </div>
144     <div class="form-group col-md-12">
145         <div class="input-group input-group-sm">
146             <div class="input-group-prepend">
147                 <span class="input-group-text" id="resol">
                        Resolucao</span>
148             </div>
149             <input type="text" class="form-control"
150                 id="video_resolution_width_input"
                        placeholder="width (px)"
151                 required>
152             <input type="text" class="form-control"
153                 id="video_resolution_height_input"
                        placeholder="height (px)"
154                 required>
155         </div>
156     </div>
157 </div>
158 <hr style="margin: 2px 0 15px 0">
159 <strong style="font-size: 13px;">Informacoes do objeto</
    strong>
160 <div class="form-row" style="margin-top: 10px;">
161     <div class="form-group col-md-12">
```

```
162         <div class="input-group input-group-sm">
163             <div class="input-group-prepend">
164                 <span class="input-group-text" id="tam">
                       Tamanho</span>
165             </div>
166             <input type="text" class="form-control" id="
                   obj_width_input"
167                 placeholder="width (cm)" required>
168             <input type="text" class="form-control" id="
                   obj_height_input"
169                 placeholder="height (cm)" required>
170         </div>
171     </div>
172 </div>
173 <hr style="margin: 2px 0 15px 0">
174 </fieldset>
175 <button type="submit" id="btn_start" style="width: 100%;"
176 class="btn btn-success">Iniciar</button>
177 <button type="button" onclick="parar()" id="btn_stop" style
   ="width: 100%;"
178 class="btn btn-danger">Parar</button>
179 </form>
180 </div>
181 </div>
182 </div>
183 </div>
184 </div>
185 </div>
186 </main>
187
188 <script src="{{ url_for('static', filename='js/bootstrap.
   min.js') }}"></script>
```

```
189 <script src="{{ url_for('static', filename='js/jquery
       -3.4.1.min.js') }}"></script>
190 <script type="text/javascript">
191 var total_toasts_count = document.querySelector('.
       total_toasts_count'),
192 total_ok_toasts_count = document.querySelector('.
       total_ok_toasts_count'),
193 total_target_toasts_count = document.querySelector('.
       total_target_toasts_count'),
194 velocity = document.querySelector(".velocity"),
195 video_inputs = document.querySelector('.video_input'),
196 websocket = new WebSocket("ws://{{serverIP}}:6789/");
197
198 websocket.onmessage = function (event) {
199 data = JSON.parse(event.data);
200 switch (data.type) {
201 case 'state':
202 console.log(data)
203 total_toasts_count.textContent = data.total_toasts_count;
204 total_ok_toasts_count.textContent = data.total_toasts_count
        - data.total_target_toasts_count;
205 total_target_toasts_count.textContent = data.
       total_target_toasts_count;
206
207 var displayOptions = "";
208
209 data.input_video.forEach(src => {
210 displayOptions += "<option>" + src + "</option>";
211 });
212 video_inputs.innerHTML = displayOptions;
213
214 if (data.video_src) {
```

```
215 video_inputs.selectedIndex = data.input_video.indexOf(data.
    video_src);
216 }
217
218 document.getElementById("fieldset").disabled = data.active;
219
220 if (data.active) {
221 document.getElementById("btn_start").style.display = "none"
    ;
222 document.getElementById("btn_stop").style.display = "block"
    ;
223 } else {
224 document.getElementById("btn_start").style.display = "block
    ";
225 document.getElementById("btn_stop").style.display = "none";
226 }
227 break;
228 default:
229 console.error("unsupported event", data);
230 }
231 };
232
233 var fonte = document.getElementById("video_input")
234 var limiar = document.getElementById("limiar_input")
235 var velocidade = document.getElementById("velocidade_input"
    )
236 var fov = document.getElementById("fov_input")
237 var fps = document.getElementById("fps_input")
238 var dist = document.getElementById("dist_input")
239 var obj_width = document.getElementById("obj_width_input")
240 var obj_height = document.getElementById("obj_height_input"
    )
```

```
241 var video_resolution_width = document.getElementById("
      video_resolution_width_input")
242 var video_resolution_height = document.getElementById("
      video_resolution_height_input")
243
244 function iniciar() {
245 velocity.textContent = velocidade.value + " cm/s";
246
247 obj = {
248 active: true,
249 fonte: fonte.value,
250 limiar: limiar.value,
251 fov: fov.value,
252 velocity: velocidade.value,
253 fps: fps.value,
254 dist: dist.value,
255 obj_width: obj_width.value,
256 obj_height: obj_height.value,
257 video_resolution_width: video_resolution_width.value,
258 video_resolution_height: video_resolution_height.value
259 }
260 websocket.send(JSON.stringify(obj));
261 return false;
262 }
263
264 function parar() {
265 websocket.send(JSON.stringify({ active: false }));
266 return false;
267 }
268 </script>
269 </body>
270
```

```
271 </html>
```