

TI0147 - FUNDAMENTOS DE PROCESSAMENTO DIGITAL DE IMAGENS

Toast Tracker

Fortaleza

2019

Sumário

1	INTRODUÇÃO	2
1.1	Estudo do Problema	2
1.2	Esquema geral do projeto	3
2	AQUISIÇÃO DE IMAGEM	4
2.1	Detalhes de implementação	5
3	PRÉ-PROCESSAMENTO DOS VÍDEOS	6
3.1	Objetivos	6
3.2	Metodologia	6
3.3	Técnicas utilizadas	6
3.4	Algoritmo	7
3.5	Resultados	8
4	PROCESSAMENTO DOS VÍDEOS	9
4.1	Objetivo	9
4.2	Metodologia	9
5	INTERFACE DA APLICAÇÃO	10
5.1	Objetivos	10
5.2	Front-End: Apresentação Visual da Interface	10
5.3	Back-End: Processamento Interno da Interface	11
6	CONCLUSÃO	14
	REFERÊNCIAS	15
	ANEXO A – LISTAGEM DAS EQUIPES	16

1 Introdução

O presente documento detalha o desenvolvimento e conclusão do projeto [nome do projeto], que tem como objetivo detectar falhas de padrão no posicionamento de torradas em uma esteira de linha de produção. Esse projeto foi desenvolvido como atividade final da disciplina “TI0147 - Fundamentos de Processamento Digital de Imagens”, ofertada pelo Departamento de Teleinformática da Universidade Federal do Ceará (UFC-DETi), turma 2019.2, no qual houve participação de toda a turma e gerenciamento do aluno Natanael Moreira de Lemos. No Apêndice A encontra-se uma listagem detalhada de cada equipe, com seus participantes e atividades desenvolvidas.

O objetivo geral do projeto consiste em desenvolver uma aplicação computacional capaz de detectar as torradas que estão fora do padrão de enquadramento e as classificá-las como adequadas ou inadequadas[ver com o pessoal da interface a classificação], em uma simulação que imita uma situação real de um esteira de linha de produção, dadas as suas dimensões físicas. O sistema deveria[como o sistema deveria se comportar?]. O projeto tem justificativa prática e é de interesse comercial, sistemas como o desenvolvido neste trabalho podem ser utilizados na indústria, no setor de qualidade de uma fábrica para otimização da linha de produção. Tal tecnologia substituiria a prática utilizada atualmente, na qual observadores humanos fazem a inspeção de qualidade, o que torna a possibilidade de erros humanos recorrente.

A implementação final utilizou linguagem Python, biblioteca OpenCV, framework Flash, Html, Css e JS.

1.1 Estudo do Problema

Com o objetivo de realizar experimentos e validar a qualidade da aplicação desenvolvida, foi desenvolvido um vídeo simulando uma esteira de produção de torradas.

Em seguida, iniciou-se o estudo do problema e identificação das necessidades do projeto. Inicialmente, notou-se a necessidade de realizar um tratamento nos frames do vídeo, afim de adequá-los para a etapa de processamento de vídeo.

Analizando individualmente as imagens disponíveis, foram observadas as primeiras necessidades referentes à qualidade das imagens obtidas e objetivos gerais do projeto. As necessidades foram definidas como:

1. Realce e correção de degradação dos frames do vídeo;
2. Delimitação de cada torrada em cada frame;

3. Centro de massa e angulação das torradas;
4. Detecção das torradas fora da angulação padrão;

1.2 Esquema geral do projeto

Para que se torne algo de uso prático para usuários não especializados, foi desenvolvida uma interface gráfica. Essa interface deveria ser responsável por executar os métodos de processamento de vídeo necessários e exibir os resultados estáticos ao operador final. Houve a necessidade de aplicar práticas de Pré-Processamento para efetivamente transformar o vídeo original em um vídeo de que exibia apenas os contornos das torradas preenchidos, para serem utilizados no sistema de detecção de padrão. Essa etapa deveria, então, tratar o vídeo realce e correção de degradação das imagens adquiridas, a fim da adequação do produto à utilização na etapa de construção do algoritmo de processamento.

Para tornar a produção ágil e atacar cada problema individualmente, a equipe do projeto foi dividida em quatro equipes que deveriam realizar o desenvolvimento independentemente, completando cada requisito do projeto:

1. Aquisição de imagens;
2. Pré-processamento de vídeo;
3. Processamento de vídeo;
4. Interface de usuário;

Para manter a independência de cada equipe, bem como garantir a integralização do produto final foi definido um pipeline de execução do projeto. Consistiu em fazer cada etapa receber, processar e produzir arquivos para, assim, garantir o andamento do projeto. Dessa forma as equipes poderiam usar arquivos de testes sem necessidade das outras etapas já estarem prontas. O fluxo da pipeline de execução é retratado na Figura 1.

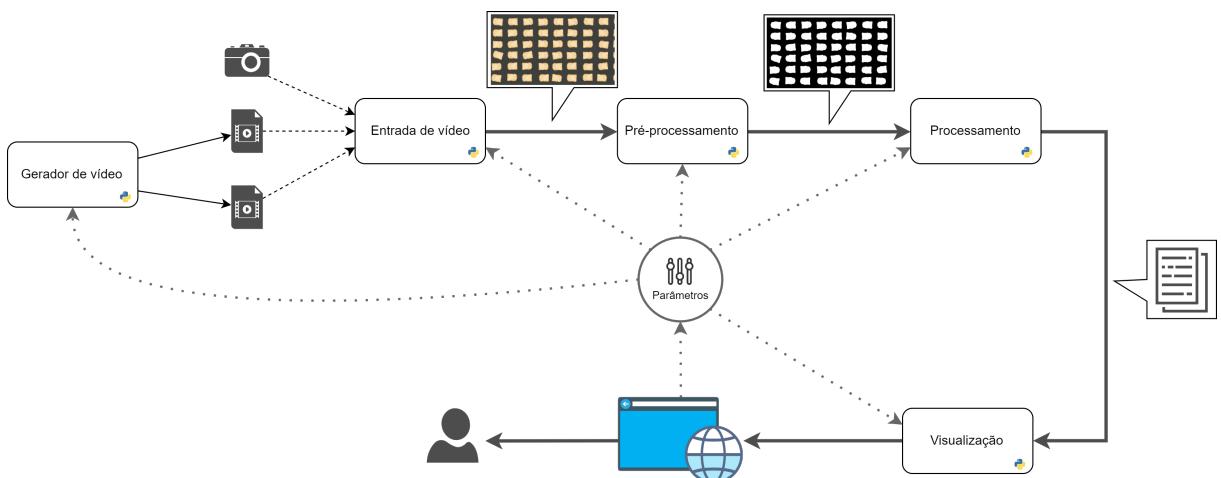


Figura 1 – Pipeline de execução do projeto

2 Aquisição de Imagem

A aquisição das imagens consiste na primeira etapa do projeto e pode ser feita de duas formas principais: gravação do ambiente real ou por meio de simulações. Embora a primeira opção pareça ser a melhor, ela apresenta algumas limitações. Primeiramente limitações de acesso as fábricas para realizar as gravações e em seguida, limitações na variedade de configurações diferentes, já que não é viável gravar esteiras com diferentes texturas, tipos de torradas, velocidades e diversas outras configurações que podem ser encontradas na industria. Desenvolver um sistema pensando em toda essa diversidade de possibilidades permite que ele possa ser facilmente adaptado as mais diversas situações. Pensando por esse ponto de vista, o uso de simulações se mostra vantajoso por proporcionar a criação de um grande banco de vídeos simulando as mais diversas aplicações do projeto.

Neste projeto, optamos por realizar a aquisição das imagens através de simulações. Um sistema de geração de vídeos foi desenvolvido para permitir a criação de material com base em dezesseis parâmetros, que vão desda textura da esteira utilizada e sua velocidade, até a adição de ruídos ao vídeo para aproxima-lo ainda mais da realidade. Esses parâmetros podem ser agrupados em quatro classes de informações: câmera, objetos (torradas nesse caso), esteira e ruídos.

Na primeira classe encontram-se informações referentes a câmera utilizada na simulação, como por exemplo, sua resolução, campo de visão (FOV), distância da câmera até a esteira, taxa de frames por segundo (FPS) e duração do vídeo que será criado. A segunda classe agrupa informações referentes aos objetos (só é aceito um tipo de objeto por vídeo) que passarão na esteira, dentre essas informações temos: dimensões média dos objetos em centímetros, quantidade de objetos por coluna na esteira (objetos são igualmente espaçados na vertical), espaçamento entre os objetos ao longo da esteira (espaço na horizontal entre as colunas) e por fim, é necessário fornecer também uma imagem da torrada que passará na esteira. Continuando, na terceira classe é preciso fornecer a velocidade da esteira em centímetros por segundo e uma imagem ilustrando a textura da esteira, essa imagem será replicada e suas replicas colocadas lado a lado para dar origem a esteira. Por fim e não menos importante, a classe dos ruídos ajuda a fazer uma simulação mais próxima da realidade adicionando ruídos uniformes na posição dos objetos na esteira e também no seus tamanhos, além de ruídos gaussianos tri-modais na rotação dos objetos e cinco diferentes tipos de ruídos que podem ser adicionados no vídeo como um todo, são eles: *gaussian*, *poisson*, *salt*, *pepper*, *sp*, *speckle*.

Exemplos de diferentes configurações possíveis são apresentados na Figura 2.

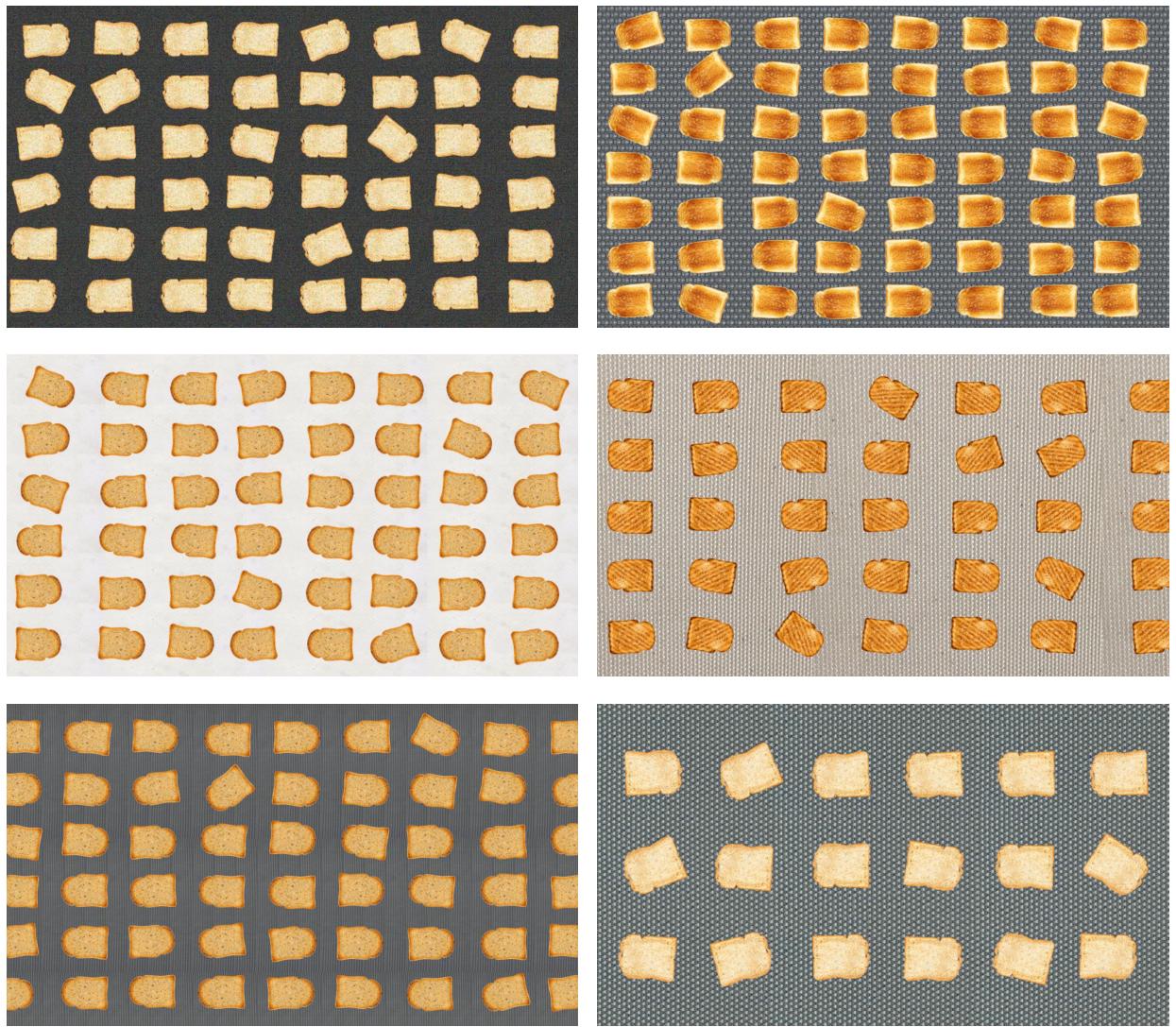


Figura 2 – Exemplos de configurações que podem ser geradas

2.1 Detalhes de implementação

A criação do vídeo é feita frame a frame. Em cada frame, primeiramente é adicionado a textura da esteira como plano de fundo do frame. Em seguida, os objetos são distribuídos sobre o frame com base nos parâmetros fornecidos. Eles são adicionados já com os respectivos ruídos de posicionamento, rotação e tamanho. Por fim, algum dentre os cinco tipos de ruídos disponíveis pode ser adicionado ao frame como um todo. vale destacar ainda que a cada frame os objetos que já estavam presentes no frame anterior são deslocados com base na velocidade da esteira. Além disso, o plano de fundo também é deslocado segundo a mesma velocidade a fim de dar mais realismo a simulação. Finalmente, o vídeo gerado é salvo no formato *.avi juntamente com um arquivo *.txt que contém todos os parâmetros usados para a criação dele.

3 Pré-Processamento dos Vídeos

3.1 Objetivos

Após a aquisição das imagens é realizada a etapa do pré-processamento. O objetivo da atividade de pré-processamento consiste no realce e correção de degradação das imagens adquiridas, a fim da adequação do produto à utilização na etapa de construção do algoritmo de processamento. Fundamentalmente, o propósito desta atividade é então subjetivo às necessidades do algoritmo aplicado nas imagens em fases subsequentes do trabalho.

3.2 Metodologia

Com base nos requisitos levantados para as imagens de saída, operou-se nas imagens obtidas com métodos especificados na linguagem python, nominalmente:

- Equalização de histograma;
- Ajuste gamma;
- Thresholding;
- Filtragem.

As operações acima foram divididas entre os membros da equipe para a manipulação das imagens adquiridas. Subgrupos foram formados e responsabilizados pela execução de cada procedimento. Para o grupo de ajuste de contraste, foi de atribuição aplicar e investigar possíveis melhorias decorrentes da manipulação das imagens via aplicação de ajuste gamma e equalização de histograma. Para a equipe de análise de ruído, foi dada a tarefa de examinar potenciais vantagens em aplicar processos de remoção de ruído às imagens. Finalmente, para a equipe de Thresholding, ficou a responsabilidade de binarizar as imagens e entregar o output da atividade.

3.3 Técnicas utilizadas

A forma como a imagem é manipulada no pré-processamento é específica para cada aplicação, pois as técnicas utilizadas são orientadas pelo escopo do projeto. A partir de uma análise visual das imagens que foram geradas e dos requisitos levantados para o projeto foram decididos realces à serem realizados.

Inicialmente é feito a transformação da imagem para tons de cinza com o objetivo de facilitar o processamento e torná-lo mais veloz.

A equalização de histograma tem o objetivo de melhorar o contraste da imagem, é reduzida as diferenças de intensidade entre os elementos da imagem e os detalhes são realçados. Para isso, os valores de intensidade da imagem são redistribuídos para que se possua um histograma uniforme.

Ajuste de gamma é uma transformação de potência onde é buscado o ajuste quanto ao brilho. Em imagens muito esbranquiçadas ou muito escuras o ajuste de gamma realça detalhes que anteriormente não estavam visíveis.

A limiarização ou thresholding tem como resultado uma imagem com os valores de intensidade em branco ou preto de todos os valores na faixa de interesse e, e o restando da imagem com o valor de intensidade oposto (branco ou preto).

O tipo de filtragem utilizada na imagem foi a mediana. Este filtro é responsável por retirar principalmente ruídos do tipo sal e pimenta. Se trata de uma máscara de tamanho pré definido que percorre por toda a imagem alterando um pixel por vez para o valor da mediana da intensidade dos seus vizinhos.

3.4 Algoritmo

Inicialmente, recebe-se a imagem. Em seguida, reduz-se ruídos notáveis na mesma. Posteriormente, a imagem é transformada para escala de cinza e então o processo de limiarização é aplicado.

Por fim, se ainda houver ruído residual, o mesmo é eliminado e a imagem resultante é retornada. Considere por exemplo a imagem abaixo, numa seção do código onde a imagem em escala de cinza - `image[:, :, 1]` - é limiarizada utilizando método da biblioteca opencv com posterior filtragem de mediana, a fim de eliminação de ruído do tipo sal e pimenta. Figura 3.

```
ret, thresh1 = cv.threshold(image[:, :, 1], 160, 255, cv.THRESH_BINARY)
med = median(thresh1, disk(5))
return med
```

Figura 3 – Algoritmo de filtragem mediana

A fim de robustez no tratamento das imagens de entrada, também foi proposto código de suporte - a ser usado em caso de imagens com parâmetros distintos -, nominalmente. Figura 4.

Onde se utiliza uma transformada wavelet para remoção de potencial ruído presente nas imagens tratadas. Adicionalmente, tem-se semelhantes funções para alargamento de contraste, para o caso de imagens de baixo contraste as quais tornaria o processo de

```

fps = reader.get_meta_data()['fps']

writer = imageio.get_writer('denoise_imageio.avi', fps=fps)

idx = 0

for im in reader:
    writer.append_data(denoise_wavelet(im, multichannel=True,
                                       rescale_sigma=True)[:, :, 1])
    print(idx)
    idx+=1
writer.close()

```

Figura 4 – Algoritmo do código de suporte

limiarização não bem definido para as bordas, tornando o processo de delimitação do objeto menos eficiente. Figura 5.

```

# Contrast stretching
print(i)
i = i+1
img_eq = exposure.equalize_hist(image)
p2, p98 = np.percentile(img_eq, (2, 98))

img_rescale = exposure.rescale_intensity(img_eq, in_range=(p2, p98))
gamma_corrected = exposure.adjust_gamma(img_rescale, 2)
writer.append_data(gamma_corrected)

```

Figura 5 – Código de alargamento de contraste

Onde se utiliza ajuste gamma e equalização do histograma para obtenção de uma imagem com melhor contraste.

3.5 Resultados

A avaliação das imagens obtidas para a aplicação do algoritmo de detecção de objetos fica a cargo da equipe de algoritmo - levando em consideração se as imagens são ou não próprias à aplicação do procedimento - e após devidos testes, foi constatado que a metodologia proposta nesta seção do trabalho de fato foi adequada a uso posterior.

4 Processamento dos Vídeos

4.1 Objetivo

Após o pré-processamento do vídeo, é realizado o processamento das imagens. Esta etapa tem como objetivo localizar o movimento das torradas no vídeo pré-processado, a partir dos frames em uma região delimitada para o processamento. Localizada cada torrada, é preciso, então, determinar quais delas estão fora do padrão previamente estabelecido. Com isso, deve-se manter um registro desses processamentos para cada torrada, a fim de manter informações a respeito da localização de cada uma delas que passou pela área de processamento.

4.2 Metodologia

Para realizar o processamento descrito acima, foram usados métodos implementados na linguagem Python, usando alguns módulos da linguagem. São eles:

- Numpy: usado para lidar com arrays multidimensionais;
- Math e Fraction: bibliotecas para cálculos matemáticos;
- CV2: biblioteca com métodos do OpenCV, usada para processamento de imagens

Primeiramente, são definidas variáveis necessárias para o processamento, tais como as dimensões da imagem, a região delimitada para o processamento e a velocidade da esteira, além da angulação “alfa” limite das torradas.

Com isso, são localizadas as torradas na região estabelecida, com base no contorno de cada uma na imagem, utilizando a função `cv2.findContours()`. Além disso, a função `cv2.boxPoints()` é utilizada para localizar os quatro vértices do retângulo onde a torrada está circunscrita.

Finalmente, para determinar a inclinação da torrada, a função `cv2.minAreaRect()` é utilizada para determinar uma região mínima para esse contorno, retornando três parâmetros. O primeiro corresponde a um ponto (x, y) equivalente ao vértice superior esquerdo do retângulo, e o segundo, à sua largura e altura. O terceiro parâmetro, então, retorna o ângulo de inclinação do retângulo, o qual, nesse caso, corresponde à inclinação da torrada.

5 Interface da Aplicação

5.1 Objetivos

O principal objetivo da Interface é uma interação com o usuário final de forma prática e transparente. A aplicação da interface consiste de um servidor de streaming de vídeos desenvolvido utilizando o framework Flask e linguagem Phyton. A interface apresenta o vídeo em tempo real, disponibilizando dados úteis ao usuário e uma seção que permite a configuração dos parâmetros utilizados pelo sistema.

5.2 Front-End: Apresentação Visual da Interface

Na criação do front-end, que é onde ocorre a interação visual do usuário final, foram utilizadas quatro ferramentas de desenvolvimento Web: HTML(Hypertext Markup Language), CSS(Cascading Style Sheets), JS(Java Script) e Bootstrap (framework utilizado para rápidas personalizações utilizando HTML, CSS e JS). Essas três ferramentas lidam com a formatação, estilização e tratamentos de eventos da aplicação visual de interface.

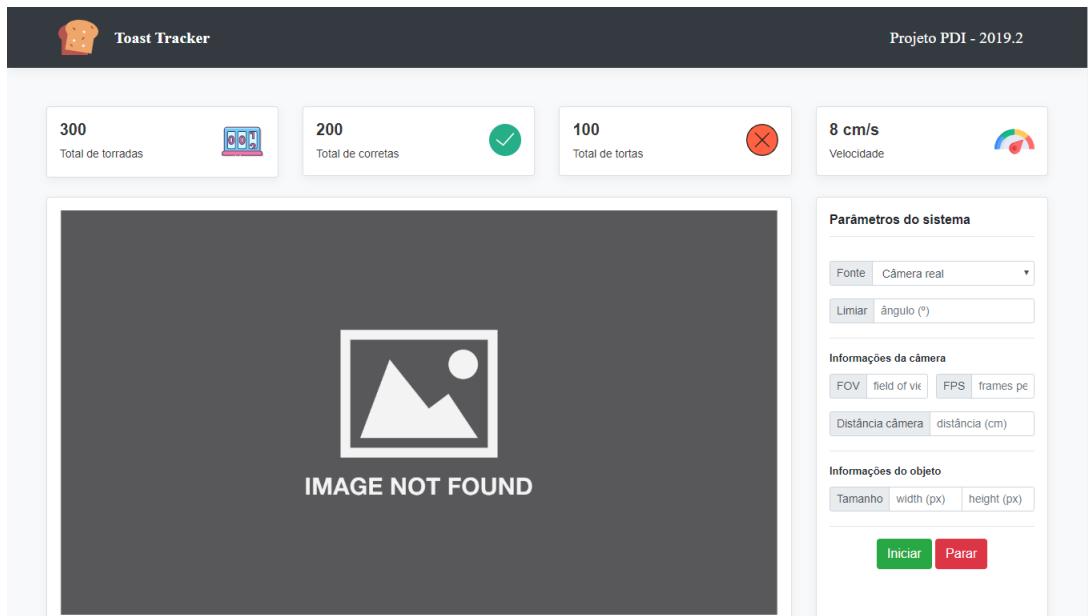


Figura 6 – Apresentação Visual da Interface

O ambiente foi pensando para o usuário não ter dúvidas na utilização. Em uma interface simples conta com tês elementos principais: uma área superior que reúne dados importantes ao usuário (Total de torradas, Total de torradas corretas, Total de tortas e Velocidade da esteira), uma barra lateral que permite parametrização do sistema junto com os botões de "Iniciar" e "Parar" o processo e, o espaço destinado ao streaming do vídeo

em tempo real. Dessa forma a ideia se torna simples, o usuário configura os parâmetros na aba lateral nos campos indicados e inicia o streaming no botão "Iniciar", o vídeo irá ser processado e os dados sobre as torradas serão exibidos nos retângulos localizados na parte superior.

5.3 Back-End: Processamento Interno da Interface

O backend da interface foi feito utilizando o microframework Flask que utiliza a linguagem python. No servidor é utilizado opencv para abrir o vídeo transmitir dentro da interface html, sendo um frame de cada vez. Antes de enviar o frame é feito um pós-processamento com os dados recebidos e são aplicados os contornos nas torradas que possuem inclinação maior que o threshold definido.

A aplicação conta com uma função principal "process" responsável por receber todas as informações vindas do script de processamento e da imagem original, adicionando os objetos visuais resultados do processamento (contornos e indicadores) e disponibilizando o streaming de video por meio da variável "output-frame". Além de guardar os dados processados a serem enviados ao frontend da aplicação via websockets. Segue abaixo imagem da função:

```
def process(processed_data, original_image):
    global output_frame
    STATE['count_tor_tot'] = processed_data['count_tor_tot']
    STATE['count_tor_alvo'] = processed_data['count_tor_alvo']
    STATE['count_tor_alvo_tela'] = processed_data['count_tor_alvo_tela']

    for t in processed_data['tor_alvo']:
        cm = (int(t[0]), int(t[1]))
        cv2.circle(original_image, cm, 10, (0, 0, 255), -1)
        cv2.putText(original_image, str(t[2]), (cm[0]-20, cm[1]-20), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 2, cv2.LINE_AA)
        cv2.rectangle(original_image, (cm[0]-20, cm[1]-20), (cm[0]+20, cm[1]+20), (0, 255, 200), 2)

    cv2.line(original_image, (100, 0), (100, 720), (0, 255, 0), 2)
    cv2.line(original_image, (250, 0), (250, 720), (0, 255, 0), 2)
    # cv2.imshow('output', original_image)
    # cv2.waitKey(1)
    with lock:
        output_frame = original_image.copy()

    loop = asyncio.new_event_loop()
    asyncio.set_event_loop(loop)
    loop.run_until_complete(notify_state)
    loop.close()
```

Figura 7 – Função "process"

O websocket é responsável por monitorar e atualizar as informações do processamento no frontend da aplicação, ou seja, toda vez que for constatada uma mudança nas informações, o websocket se encarregará de enviar as informações atualizadas. Segue abaixo imagem do trecho do código do websocket:

```
# ##### WEBSOCKET #####
def state_event():
    return json.dumps({"type": "state", **STATE})

async def notify_state():
    if USERS: # asyncio.wait doesn't accept an empty list
        message = state_event()
        await asyncio.wait([user.send(message) for user in USERS])

async def register(websocket):
    USERS.add(websocket)

async def unregister(websocket):
    USERS.remove(websocket)
|
async def counter(websocket, path):
    # register(websocket) sends user_event() to websocket
    await register(websocket)
    try:
        await websocket.send(state_event())
        async for message in websocket:
            data = json.loads(message)
            if data["action"] == "minus":
                STATE["value"] -= 1
                await notify_state()
            elif data["action"] == "plus":
                STATE["value"] += 1
                await notify_state()
            else:
                logging.error("unsupported event: {}", data)
    finally:
        await unregister(websocket)
```

Figura 8 – Web socket

O webserver é responsável pela disponibilização dos frames de vídeo, ou seja, o streaming do vídeo em tempo real, possibilitando que a mesma seja processada pelos demais algoritmos. Segue abaixo imagem do trecho do código do webserver:

```
# ##### WEB SERVER #####
def generate():
    # grab global references to the output frame and lock variables
    global output_frame, lock

    # Loop over frames from the output stream
    while True:
        # wait until the lock is acquired
        with lock:
            # check if the output frame is available, otherwise skip
            # the iteration of the loop
            if output_frame is None:
                continue
            # encode the frame in JPEG format
            (flag, encodedImage) = cv2.imencode(".jpg", output_frame)

            # ensure the frame was successfully encoded
            if not flag:
                continue

            # yield the output frame in the byte format
            yield(b"--frame\r\n" b'Content-Type: image/jpeg\r\n\r\n' +
                  bytearray(encodedImage) + b'\r\n')
```

Figura 9 – Web socket

6 Conclusão

Ao final da execução do projeto, implementação e testes, foi possível concluir a eficiência do sistema, atendendo a todos os requisitos de funcionamento e interface. Gerando resultados satisfatórios na análise da disposição das torradas na esteira de produção.

Para trabalhos futuros, a experimentação das técnicas em uma situação real seria o próximo passo para o aperfeiçoamento qualitativo do sistema e melhor atingir os objetivos. Os resultados seriam melhores sabendo a velocidade, posição e limitações reais da situação.

Referências

- [1] Gonzalez C., Rafael Processamento digital de imagens: 3 ed. São Paulo: Pearson Prentice Hall, 2010.
- [2] Processamento de Histogramas. Disponível em: <<http://www.facom.ufu.br/baces/gsi058/Aula04 -ProcessamentoHistogramas.pdf>> Acesso em: 10 nov. 2019.
- [3] Tutorial Thresholding . Disponível em: <https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html> Acesso em 13 nov 2019.
- [4] HAN, Dianyuan; HUANG, Xinyuan. A Tree Image Segmentation Method Based on 2-D OTSU in HSI color space. IEEE. China, p. 41-44. abr. 2010.

ANEXO A – Listagem das Equipes

EQUIPE DE GERÊNCIA

Tarefas: orientar as tarefas das demais equipes; fiscalizar as atividades executadas;

Líder: NATANAEL MOREIRA DE LEMOS (398447)

Membros:

Nome	Matrícula
AIRTON SILVA MESQUITA FILHO	408338

EQUIPE DE AQUISIÇÃO DE IMAGENS

Tarefas: aquisição das imagens usadas no projeto;

Líder: MATHEUS FERREIRA LESSA (374947)

EQUIPE DE PRÉ-PROCESSAMENTO

Tarefas: transformar vídeo em imagens de frames; realce (realce por contraste, realce de borda, etc.); realce correção de degradação das imagens adquiridas;

Líder: DANIEL FILHO COELHO RAMOS (374921)

Membros:

Nome	Matrícula
LEVIR CÉSAR RIBEIRO LEMOS	400555
FRANCISCO DAVID MOREIRA DE SOUSA	400496
HUGO LINHARES CRUZ TABOSA BARROSO	378602
JOSÉ MARCÍLIO DE SOUSA	385199
LUCA ISRAEL DE MOURA CRUZ	374942
FELIPE ARAÚJO MAGALHÃES	378599

EQUIPE DE ALGORITMO

Tarefas: contar torradas em sequência de imagens; classificar objetos por bounding-box; gerar imagens com objetos devidamente identificados; **Líder:** RAMIRO CAMPOS DE CASTRO (400723)

Membros:

Nome	Matrícula
LUAN PEREIRA DE LIMA BRASIL	397743
PEDRO CERCELINO MATOS	399325
LAIS BRANDÃO GADELHA	378606
ALAN OLIVEIRA MAIA	397280
PEDRO IVO GALLÃO BRITO	402268
LUCAS ESTEVES ROCHA	404708

EQUIPE DE INTERFACE

Tarefas: desenvolver interface de usuário; garantir portabilidade e usabilidade;

Líder: EDUARDO VIANA DE ABREU (399016)

Membros:

Nome	Matrícula
EMERSON MARQUES ARAUJO	398425
RHANIEL MAGALHÃES XAVIER	385215
ROMULO FERREIRA MOURA MAIA	378612
GLAUTON CARDOSO SANTOS	404201
MARIO VICTOR R. SALES	408848
MATEUS CÉSAR PINHEIRO LANDIM	385208