UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Vinícius Silva Ferreira

Otimização do Processamento de Imagens de Ovos de Galinha com Interface Gráfica Interativa e Paralelismo em Python

Uberlândia, Brasil

2025

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Vinícius Silva Ferreira

Otimização do Processamento de Imagens de Ovos de Galinha com Interface Gráfica Interativa e Paralelismo em Python

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Mauricio Cunha Escarpinati

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2025

Vinícius Silva Ferreira

Otimização do Processamento de Imagens de Ovos de Galinha com Interface Gráfica Interativa e Paralelismo em Python

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 29 de novembro de 2025:

Mauricio Cunha Escarpinati
Orientador

Daniel Duarte Abdala
Professor

Diego Nunes Molinos
Professor

Uberlândia, Brasil
2025

Resumo

Este trabalho propõe a reestruturação de um sistema de processamento digital de imagens voltado à análise morfológica de ovos de galinha, com foco em desempenho computacional e usabilidade. A proposta abrange a substituição do modelo sequencial por execução paralela com a biblioteca *concurrent.futures*, adotando a classe *ProcessPoolExecutor* para distribuir o processamento das subimagens entre múltiplos núcleos de CPU. O sistema também passou por uma reformulação completa da interface gráfica, utilizando a biblioteca Tkinter para tornar a interação mais intuitiva por meio de botões visuais e menus organizados. O projeto foi estruturado com uma arquitetura modular, separando a lógica da interface gráfica da lógica de processamento, o que facilitou a manutenção do código e possibilitou a aplicação eficiente do paralelismo em tarefas do tipo CPU-bound. A metodologia adotada envolveu a segmentação assistida de imagens contendo 30 ovos, além da extração automatizada de medidas morfológicas, com geração de relatórios e imagens anotadas organizadas em diretórios padronizados. Testes comparativos realizados em dois processadores com arquiteturas distintas (Intel Core i5-6200U e i3-N305) apresentaram reduções expressivas no tempo de execução, validando a eficácia das otimizações implementadas. Os resultados obtidos demonstram que a solução proposta oferece ganhos significativos em desempenho e usabilidade, sendo adequada para cenários que exigem análise em lote de imagens com maior eficiência computacional.

Palavras-chave: Processamento de imagens, Paralelismo, interface gráfica, ovos, Python, Tkinter.

Abstract

This work proposes the restructuring of a digital image processing system aimed at the morphological analysis of chicken eggs, with a focus on computational performance and usability. The proposal includes replacing the sequential model with parallel execution using the concurrent.futures library, specifically adopting the ProcessPoolExecutor class to distribute the processing of subimages across multiple CPU cores. The system also underwent a complete redesign of its graphical interface, using the Tkinter library to make user interaction more intuitive through visual buttons and organized menus. The project was structured with a modular architecture, separating the graphical interface logic from the processing logic, which facilitated code maintenance and enabled the efficient application of parallelism to CPU-bound tasks. The adopted methodology involved assisted segmentation of images containing 30 eggs, as well as the automated extraction of morphological measurements, with the generation of reports and annotated images organized into standardized directories. Comparative tests conducted on two processors with different architectures (Intel Core i5-6200U and i3-N305) showed significant reductions in execution time, validating the effectiveness of the implemented optimizations. The results demonstrate that the proposed solution delivers significant gains in performance and usability, making it suitable for scenarios that require batch image analysis with greater computational efficiency.

Keywords: *image processing, parallelism, graphical interface, eggs, Python, Tkinter.*

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

GUI	Graphical User Interface (Interface Gráfica de Usuário)
GIL	Global Interpreter Lock
OpenCV	Open Source Computer Vision Library
IHC	Interação Humano-Computador
I/O	Input/Output (Entrada/Saída)
API	Application Programming Interface
HSV	Hue (matiz), Saturation (saturação) e Value (valor)

Sumário

1 Introdução

A computação exerce influência decisiva em diversos campos da sociedade. Com o avanço de algoritmos e hardwares de alto desempenho, emergiram novos desafios de execução paralela e comunicação em larga escala. Esses desafios demandam soluções inovadoras em modelos de programação, movimentação de dados e pilhas de software para sustentar a evolução da área de Ciência da Computação (GONNORD et al., 2022).

A busca por melhorias em algoritmos e sistemas de processamento de dados é constante, a programação paralela e o uso de múltiplos processos destacam-se como estratégias para otimizar o tempo de processamento, a renderização e outras rotinas de processamento de imagens, sobretudo em cenas complexas ou com alta qualidade, continuam exigindo elevado poder de computação, o que motiva abordagens paralelas e aceleração em hardware. Essas necessidades reforçam soluções que combinam modelos de programação adequados e stacks otimizadas para CPU/GPU (SZELISKI, 2022).

A importância da visão computacional é evidente em aplicações como detecção de objetos, reconhecimento de imagens e reconhecimento facial. Entretanto, o crescente volume de dados e a maior complexidade das cenas impõem limites de eficiência e precisão às abordagens tradicionais, motivando técnicas e pipelines mais otimizados e, quando possível, paralelizados (SZELISKI, 2022)

Este trabalho constitui uma parte integrante de uma pesquisa de Pós-Graduação atualmente em andamento na Faculdade de Computação da Universidade Federal de Uberlândia. O foco desta pesquisa é a elaboração de um algoritmo destinado à identificação de ovos de galinha utilizando técnicas de processamento de imagem. Uma das principais preocupações abordadas neste contexto é a lentidão observada no tempo de processamento do método empregado no código desenvolvido. Este desafio representa um aspecto significativo a ser abordado na busca por melhorias na eficácia do algoritmo em questão.

Assim, como uma solução para mitigar a lentidão do algoritmo, proposto aplicação de técnica de processamento paralelo, acreditando que isso poderá significativamente aprimorar o desempenho do código. De maneira simplificada, a execução paralela é basicamente dividir uma tarefa pesada em várias subtarefas menores e dispará-las ao mesmo tempo em múltiplos núcleos de CPU, reduzindo o tempo total de execução quando lidamos com grandes volumes de dados ou operações complexas.

1.1 Motivação e contextualização

A indústria avícola desempenha papel fundamental no suprimento de ovos de galinha diante do aumento da demanda por proteína animal. Entretanto, os métodos manuais tradicionalmente empregados para avaliação de qualidade, como inspeções visuais de tamanho e cor, são lentos, demandam mão de obra especializada e, em muitos casos, são destrutivos, o que eleva custos operacionais(RHO; CHO, 2024).

Além disso, para atender a essa necessidade, a utilização de técnicas de processamento paralelo ganha visibilidade, visto que pode contribuir significativamente para reduzir o tempo de processamento. Assim, busca-se não apenas aprimorar a eficiência do código desenvolvido, mas também possibilitar o processamento de uma quantidade maior de ovos de galinha em menos tempo, tornando o processo de identificação de ovos mais ágil e econômico.

1.2 Objetivos

O objetivo deste trabalho é otimizar o processamento de imagens de ovos de galinha por meio da aplicação de paralelismo e melhorias de código, visando reduzir o tempo de execução e preservar a qualidade dos resultados.

Para concretizar esse propósito, o estudo busca, em nível macro:

Entender como funciona o processamento atual e descobrir onde estão os pontos que mais atrapalham o desempenho.

Pensar em estratégias de paralelização que façam sentido para o tipo de tarefa, especialmente considerando que cada imagem tem 30 ovos.

Implementar essas mudanças usando técnicas de execução paralela, deixando o sistema mais ágil quando precisar lidar com muitas imagens.

Por fim, comparar o desempenho antes e depois das melhorias com métricas quantitativas (tempo médio e redução percentual).

1.3 Organização do trabalho

Esse trabalho está estruturado da seguinte forma: no Capítulo 1, apresenta-se a introdução ao tema, destacando a motivação, os objetivos e a justificativa para a realização do estudo. O Capítulo 2 é dedicado à revisão bibliográfica, na qual são abordados os principais conceitos relacionados à execução paralela, à interação humano-computador e às bibliotecas utilizadas no desenvolvimento da solução proposta.

O Capítulo 3 contempla a análise de trabalhos correlatos, com o objetivo de contex-

tualizar a proposta no cenário de pesquisas existentes e identificar contribuições similares ou complementares. No Capítulo 4, é descrita a metodologia empregada para o desenvolvimento da aplicação, detalhando o funcionamento do código original, as etapas de refatoração com paralelismo, o pré-processamento das imagens, a segmentação assistida e a extração de medidas morfológicas.

No Capítulo 5, são apresentados os resultados obtidos com a implementação do sistema, incluindo a análise comparativa de desempenho entre a versão sequencial e a versão paralelizada, além da avaliação da nova interface gráfica. Por fim, o Capítulo 6 traz as conclusões do trabalho e sugestões para pesquisas futuras que visem aprimorar e expandir a solução desenvolvida.

2 REVISÃO BIBLIOGRÁFICA

2.1 Arquitetura de Execução: Processos e Threads

A eficiência computacional em sistemas modernos depende diretamente do modo como tarefas são organizadas e executadas. Dois conceitos fundamentais nesse contexto são processos e *threads*. Um processo é uma entidade independente com seu próprio espaço de memória e recursos do sistema, enquanto uma *thread* é uma linha de execução dentro do processo, compartilhando o mesmo espaço de endereçamento e recursos (TANENBAUM; BOS, 2022).

O modelo de *thread* é uma linha de execução dentro de um processo, compartilhando o mesmo espaço de memória e recursos. O compartilhamento reduz custo de criação e troca de contexto e facilita a execução concorrente de tarefas, porém reintroduz riscos de condições de corrida e erros de sincronização, exigindo mecanismos como locks, semáforos e monitores. (HEROUX et al., 2022).

Na prática, sistemas de alto desempenho combinam múltiplos processos (para isolamento, escalabilidade entre núcleos/soquetes ou contornar restrições de runtime) e múltiplas threads (para paralelismo leve dentro do processo), ajustando a arquitetura ao perfil da aplicação e às restrições do ambiente de execução. (HEROUX et al., 2022).

2.2 Modelos Computacionais: Concorrência e Paralelismo

Concorrência e paralelismo são termos complementares, mas distintos, dentro da computação moderna. Concorrência refere-se à decomposição de um problema em múltiplas tarefas que progridem de forma intercalada no tempo, geralmente em um único núcleo de CPU. Já o paralelismo busca a execução simultânea dessas tarefas, distribuindo-as entre múltiplos núcleos ou processadores (NGUYEN, 2018).

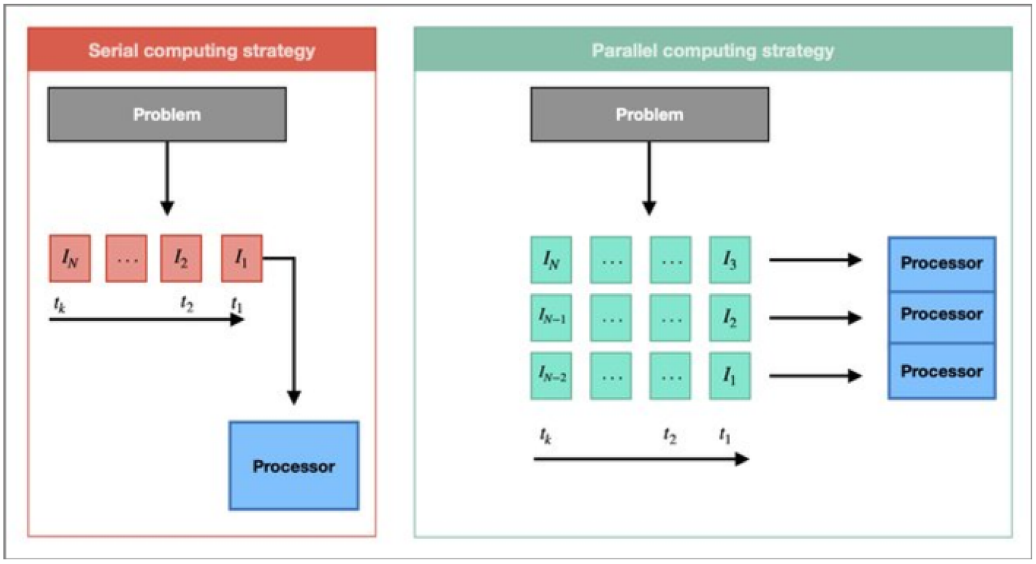


Figura 1 – Comparação entre estratégias de computação serial e paralela. Adaptado de (BENSAKHRIA, 2023).

Na figura 1 é apresentada a distinção entre computação serial e paralela. Enquanto na abordagem serial as instruções são executadas em sequência por um único processador, na estratégia paralela essas instruções podem ser distribuídas e executadas simultaneamente em diferentes unidades de processamento, reduzindo significativamente o tempo total de execução da tarefa.

Aplicações interativas e sistemas que demandam responsividade se beneficiam da concorrência, mesmo sem múltiplos núcleos. Por outro lado, algoritmos intensivos em cálculo, como processamento de imagens ou simulações científicas, extraem vantagens substanciais do paralelismo real. A escolha entre um modelo ou outro depende do tipo de carga de trabalho, da arquitetura do sistema e do grau de independência entre as tarefas.

2.3 Estratégias de Execução: Multiprocessamento e Multithreading

A implementação prática da concorrência e do paralelismo pode ser feita por meio de duas abordagens principais: multiprocessamento e multithreading. O multiprocessamento cria processos independentes que rodam em paralelo, cada um com seu próprio espaço de memória e interpretador Python. Essa abordagem é ideal para tarefas CPU-bound, pois permite verdadeira execução simultânea, especialmente em sistemas com múltiplos núcleos (NGUYEN, 2018).

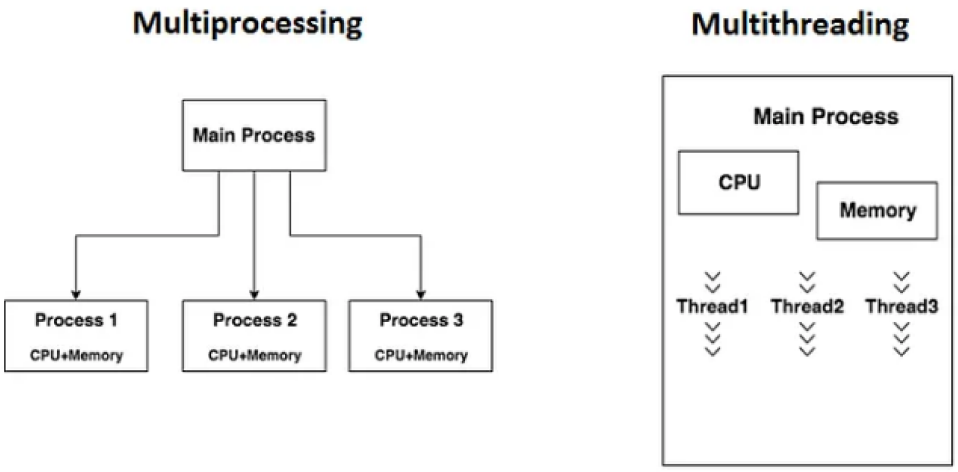


Figura 2 – Diferença entre multiprocessamento (processos independentes) e multithreading (threads compartilhando recursos) (Nouer Uz Zaman, 2023).

As diferenças entre as arquiteturas de execução podem ser observadas na figura 2. No multiprocessamento, cada processo possui sua própria CPU e memória isoladas, enquanto no multithreading múltiplas threads compartilham o mesmo espaço de memória e CPU dentro do processo principal.

Por sua vez, o *multithreading* cria várias threads dentro do mesmo processo, permitindo a divisão lógica de tarefas com comunicação rápida e leve entre elas. Essa abordagem é útil para tarefas *I/O-bound* ou para operações que exigem resposta rápida e contínua. No entanto, no Python, o multithreading encontra um obstáculo significativo: o Global Interpreter Lock (GIL).

2.4 Limitador do Paralelismo Real: Global Interpreter Lock (GIL)

O Global Interpreter Lock, é um recurso interno da principal implementação do Python (CPython) que impede que mais de uma thread execute código Python ao mesmo tempo. Mesmo em computadores com múltiplos núcleos, funciona como um "bloqueio central", liberando o acesso à CPU para apenas uma thread por vez, o que limita o ganho de desempenho em aplicações multithreading (Python Software Foundation, 2024b).

Na prática, o GIL cria um gargalo que impede o ganho de desempenho em tarefas intensivas de CPU. Em experimentos clássicos como contadores concorrentes ou laços de cálculo intensivo, múltiplas threads acabam executando de forma serializada, revezando o uso do GIL e resultando em tempos de execução semelhantes ou piores em comparação com a execução sequencial (NGUYEN, 2018).

O funcionamento do GIL em gráficos informa que, mesmo com duas threads e dois núcleos, apenas uma thread é executada por vez, enquanto a outra permanece bloqueada.

Isso inviabiliza o paralelismo real em aplicações *multithread* puras com Python (TANENBAUM; BOS, 2022). Esse comportamento é representado de forma esquemática na figura 3, embora múltiplas threads estejam disponíveis e diversos núcleos existam, o GIL garante que apenas uma thread por vez acesse o interpretador, enquanto as demais aguardam, criando um gargalo de execução.

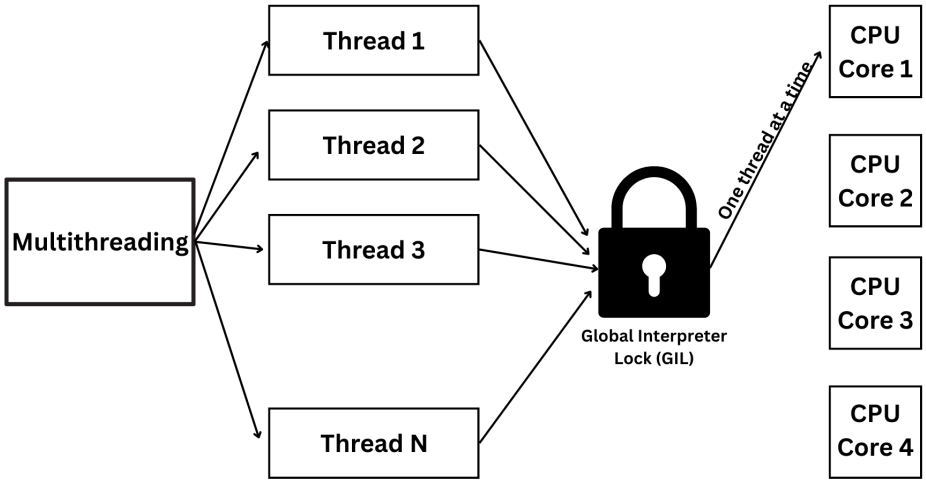


Figura 3 – Funcionamento do GIL em Python: uma única thread executa por vez, mesmo com múltiplos núcleos disponíveis. Adaptado de (Codecademy Team, 2025)

Como alternativa, este trabalho adotou o multiprocessamento com *ProcessPoolExecutor*, técnica que cria subprocessos independentes, cada um com sua própria instância do interpretador e livre das restrições do GIL. Isso permitiu a execução paralela real das tarefas de análise de imagem, com ganhos significativos de desempenho medidos empiricamente.

Além disso, no ecossistema Python, bibliotecas como NumPy, SciPy e OpenCV atenuam o impacto do GIL ao concentrar o trabalho pesado em operações vetorizadas, reduzindo a sobrecarga do interpretador. Para tarefas CPU-bound, o paralelismo efetivo é obtido com processos independentes via *multiprocessing* e *ProcessPoolExecutor*, em *I/O-bound*, o uso de *threading* ou *asyncio* permite sobreposição de latência (Python Software Foundation, 2024a).

2.5 Paralelismo no Processamento de Imagens

O paralelismo computacional tem sido amplamente estudado como uma solução eficaz para otimizar o desempenho de programas que executam tarefas repetitivas e independentes. Em Python, o módulo multiprocessing se destaca por permitir a execução de múltiplos processos em paralelo, aproveitando melhor os múltiplos núcleos do processador

ao contornar a limitação do Global Interpreter Lock (GIL) do interpretador CPython (Python Software Foundation, 2024a).

(STRECK, 2018) aplicaram essa abordagem para automatizar execuções de modelos agrícolas, como o PhenoGlad. O estudo foi motivado pelo alto tempo de processamento de simulações executadas de forma sequencial, o que causava subutilização do hardware disponível. A partir dessa limitação, os autores exploraram o uso do objeto Pool do módulo *multiprocessing*, que permite distribuir tarefas entre os núcleos de forma eficiente.

O ponto central da proposta foi a identificação de que cerca de 98% do tempo de execução era consumido pela chamada dos modelos, o que evidenciava a oportunidade de paralelização. Como cada simulação era independente e não exigia troca de dados entre processos, a paralelização pôde ser aplicada sem a necessidade de sincronização ou controle de regiões críticas.

Os autores realizaram experimentos variando o número de processos no pool e constataram melhorias significativas na performance, especialmente quando o número de processos não ultrapassava a quantidade de núcleos físicos do processador. Quando isso ocorria, observou-se uma redução na eficiência, causada pela sobrecarga do sistema operacional na gerência de processos um comportamento comum em arquiteturas com múltiplos núcleos. Na figura 4 estão sintetizados os resultados dos experimentos com 2, 4, 6 e 8 processos no pool do multiprocessing, evidenciando os tempos médios de execução.

Percebe-se que a estratégia paralela trouxe ganhos significativos, reduzindo mais da metade do tempo já a partir da execução com dois processos. Entretanto, a redução de tempo tende a se estabilizar quando a quantidade de processos se aproxima do número de núcleos físicos disponíveis. Esse efeito é ilustrado na Figura 5, que apresenta o gráfico de speedup alcançado. Observa-se que, ao utilizar oito processos, ocorre uma pequena queda no speedup em comparação aos cenários em que a quantidade de processos não ultrapassa o total de núcleos do processador.

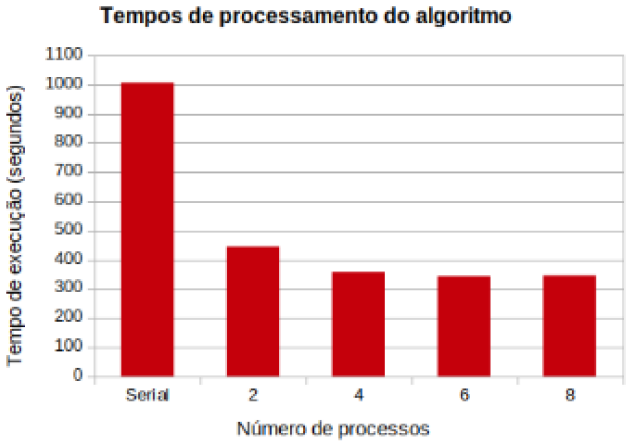


Figura 4 – Tempo médio de execução com diferentes números de processos (STRECK, 2018)

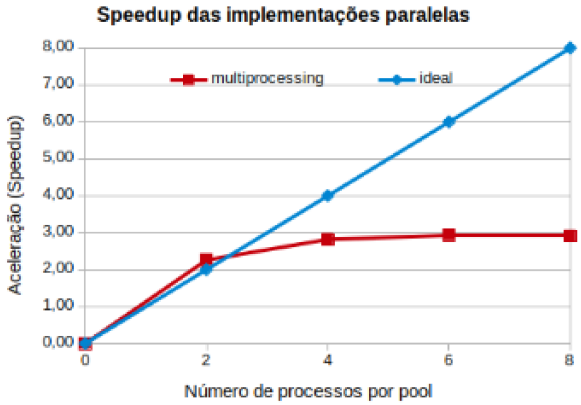


Figura 5 – Speedup obtido com a implementação paralela (STRECK, 2018)

A fundamentação apresentada por (STRECK, 2018) sustenta o uso do *multiprocessing* como base para projetos que necessitam otimizar execuções paralelas, como o presente trabalho, que trata da análise de subimagens de ovos. O uso de *ProcessPoolExecutor* neste projeto segue o mesmo princípio, explorando a independência entre as unidades de processamento e buscando ganho de desempenho por meio da execução simultânea.

No presente projeto, esse conceito foi adaptado à análise de imagens de ovos, em que o processamento de cada ovo é realizado de forma independente. Inspirado nesse estudo, o uso de *ProcessPoolExecutor* permitiu distribuir o processamento das subimagens entre diferentes processos. Essa abordagem proporcionou redução significativa do tempo de execução e melhor aproveitamento do hardware, mantendo a consistência dos resultados gerados.

2.6 Interação Humano-Computador (IHC)

A Interação Humano-Computador (IHC) é uma área da computação voltada ao estudo e à prática de projetar interfaces que promovam a comunicação eficaz entre os usuários e os sistemas computacionais. Em projetos que envolvem tarefas técnicas, como o processamento de imagens, a IHC é fundamental para garantir que os usuários possam operar os sistemas com facilidade e compreender os resultados apresentados.

Segundo (NIELSEN, 2010), a IHC busca alinhar os recursos computacionais com as necessidades cognitivas e operacionais do usuário, facilitando a execução de tarefas por meio de elementos como menus interativos, botões de ação e feedback visual. Essa perspectiva é especialmente relevante quando se considera que sistemas tecnicamente eficientes podem falhar na adoção se não forem utilizáveis.

2.6.1 Aplicação de Princípios de Usabilidade na Interface

A aplicação dos princípios de usabilidade em interfaces tem sido amplamente abordada na literatura de Interação Humano-Computador. (NIELSEN, 2010) destaca heurísticas fundamentais, como a visibilidade do estado do sistema, a correspondência entre o sistema e o mundo real, o controle pelo usuário e a consistência. Esses princípios visam facilitar a interação e minimizar a ocorrência de erros durante o uso do sistema.

(SHNEIDERMAN et al., 2017) também reforça a importância de elementos como menus organizados, botões claramente rotulados e a oferta de feedback imediato ao usuário. Tais elementos, segundo o autor, são essenciais para promover uma experiência de uso intuitiva e eficiente, principalmente em sistemas técnicos ou especializados.

No sistema analisado neste trabalho, observou-se que a interface, desenvolvida com a biblioteca Tkinter, incorpora esses princípios ao apresentar menus estruturados por funcionalidade, botões com rótulos explicativos e mensagens visuais que confirmam as ações do usuário. Essa abordagem proporciona uma redução na curva de aprendizado e contribui para a autonomia no uso da ferramenta, aspectos que vão ao encontro do que defendem (NIELSEN, 2010) e (SHNEIDERMAN et al., 2017).

Portanto, ainda que o foco deste projeto tenha sido a otimização do código, a análise da interface desenvolvida evidencia a aplicação coerente de princípios clássicos da usabilidade, reforçando a importância da IHC como parte integrante de sistemas computacionais eficazes.

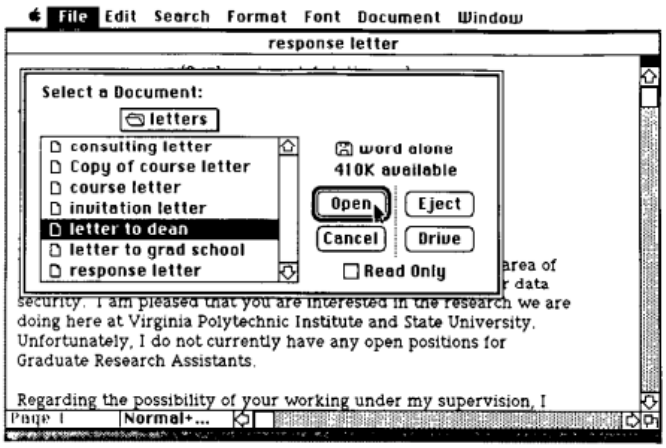


Figure 3. Example of a dialogue box that exhibits multi-thread dialogue.

Figura 6 – Exemplo de caixa de diálogo com múltiplas opções e feedback visual. (SHNEIDERMAN, 1997)

A Figura 6 apresenta um exemplo clássico de interface que utiliza caixas de diálogo com múltiplas opções, botões rotulados com clareza e elementos de feedback visual, todos alinhados aos princípios fundamentais da IHC. Esse modelo reforça a importância de uma organização lógica das opções e da apresentação de ações de forma explícita ao usuário, tal como foi considerado na construção do menu interativo deste trabalho.

No presente trabalho, foi desenvolvido um menu gráfico interativo com a biblioteca Tkinter, cuja implementação considerou aspectos essenciais de usabilidade. Entre os princípios aplicados, destacam-se menus organizados por função, botões com rótulos claros e autoexplicativos, além de mensagens visuais que oferecem feedback ao usuário a cada ação realizada. Essa estrutura reduz a curva de aprendizado e favorece a tomada de decisão durante o uso do sistema. A interface prioriza a clareza dos comandos, a minimização de erros e a simplicidade na navegação, de acordo com os princípios de usabilidade definidos por (NIELSEN, 2010), como visibilidade do estado do sistema, correspondência entre o sistema e o mundo real, e controle e liberdade do usuário.

Além da facilidade de uso, a interface também contribui para o controle preciso do processo de análise, permitindo que o usuário visualize o progresso, selecione as imagens de entrada e acompanhe os resultados em tempo real. Esses elementos refletem a importância da IHC não apenas como um complemento estético, mas como parte central na funcionalidade e no sucesso do sistema como um todo.

2.7 Tecnologias utilizadas

Nesta seção, são descritas as bibliotecas em Python utilizadas no desenvolvimento do projeto, com ênfase em como contribuíram para a otimização do código, tornando mais

eficientes as etapas de manipulação, transformação e análise das imagens, além de reduzir a complexidade das operações implementadas.

2.7.1 Linguagem em Python

A linguagem Python se consolidou como uma das favoritas entre desenvolvedores e pesquisadores, especialmente por sua simplicidade e pelas bibliotecas robustas que facilitam desde a criação de interfaces até a análise de dados e visualizações gráficas (TULCHAK; RCHUK, 2016).

Python possui uma extensa coleção de bibliotecas. Com o tempo, essas bibliotecas cresceram significativamente e suas capacidades se expandiram dramaticamente. Com apoio da comunidade global de colaboradores, estão em constante aperfeiçoamento, o que significa que limitações anteriormente problemáticas nas funções e estratégias de otimização não são mais um problema (SRINATH, 2017).

Dentro deste projeto, utiliza-se algumas das principais bibliotecas para simplificar e agilizar os processos relacionados ao processamento de imagens e à construção da interface gráfica. Dentre essas bibliotecas, destacam-se: OpenCV, para manipulação e análise de imagens; Tkinter, para o desenvolvimento da interface gráfica; *Scikit-image*, que oferece suporte a operações avançadas de processamento de imagem; e a biblioteca *concurrent.futures*, responsável pela implementação de paralelismo por meio do uso de executores, melhorando o desempenho e reduzindo o tempo de processamento.

2.7.2 OpenCV

OpenCV é uma biblioteca bastante utilizada em projetos que envolvem visão computacional. Por ser de código aberto, oferece uma ampla variedade de recursos para processar, analisar e manipular imagens e vídeos, permitindo desde operações básicas, como leitura e filtragem, até detecção de objetos e reconhecimento de padrões (MORDVINTSEV; ABID, 2014).

2.7.3 Tkinter

A criação de Interfaces Gráficas de Usuário (GUIs) representa um componente fundamental no contexto da programação. No universo Python, a biblioteca Tkinter é uma das mais acessíveis para construir interfaces gráficas. Ela permite criar janelas, botões e menus de forma simples, facilitando o desenvolvimento de sistemas mais interativos e fáceis de usar (AMOS, 2020).

2.7.4 *Scikit-image*

O *scikit-image* é uma biblioteca em Python dedicada ao processamento de imagens digitais. Essa biblioteca fornece uma ampla gama de ferramentas e algoritmos para a manipulação, análise e processamento de imagens, tornando-a uma escolha popular para pesquisadores e desenvolvedores que trabalham com visão computacional e processamento de imagens (WALT et al., 2014).

2.7.5 *concurrent.futures*

O módulo *concurrent.futures*, já incluído no Python por padrão, oferece uma forma prática de executar tarefas em paralelo. Ele permite rodar partes do código simultaneamente, seja com threads ou com processos, facilitando o uso de múltiplos núcleos para melhorar o desempenho. Entre suas principais classes está a *ProcessPoolExecutor*, projetada especificamente para facilitar a distribuição de tarefas entre múltiplos processos, em vez de threads, tornando possível a execução simultânea real em múltiplos núcleos do processador (FOUNDATION, 2024).

3 Trabalhos Correlatos

Esta seção apresentará uma série de trabalhos que utilizam diferentes interfaces de programação paralela em ambientes multi-core, enfatizando sua importância e destacando semelhanças com o projeto atual.

3.1 Otimização do Processamento de Imagens Médicas Utilizando a Computação Paralela

No estudo feito por Priscila Saito em (SAITO, 2007) teve como objetivo principal demonstrar a viabilidade na otimização do tempo de processamento de imagens médicas, especificamente por meio da programação paralela e distribuída. A abordagem utilizada se baseou em sistemas distribuídos e na biblioteca mpiJava para passagem de mensagens.

Para atingir os objetivos propostos, foram estudados em detalhes os algoritmos de suavização (filtro mediana) e de segmentação (detecção de bordas). Após um domínio aprofundado desses algoritmos, foram implementadas versões sequenciais em Java. Em seguida, as versões paralelas foram desenvolvidas para permitir a avaliação de desempenho. A implementação paralela foi realizada utilizando programação paralela e distribuída, aproveitando as capacidades da biblioteca mpiJava.

Os testes realizados com os algoritmos mostraram que, em certas situações, aplicar programação paralela e distribuída realmente ajuda a reduzir bastante o tempo necessário para processar imagens médicas. Durante os experimentos, foram avaliados vários fatores, como o tempo de execução, quantidade de máquinas envolvidas, número de processos em paralelo, além do tamanho das máscaras e das imagens analisadas. Os resultados deixaram claro que essa abordagem traz ganhos relevantes de desempenho. Inclusive, em alguns casos, o sistema ficou bem mais rápido do que o esperado — o que reforça a vantajem do uso dessa técnica em aplicações dessa área.

Na conclusão, a utilização da programação paralela e distribuída, empregando a biblioteca mpiJava, mostrou-se eficiente na otimização do tempo de processamento de imagens médicas. Os algoritmos de suavização e segmentação, quando implementados de forma paralela, proporcionaram ganhos expressivos, evidenciando a importância dessa abordagem na busca por eficiência em aplicações que lidam com imagens médicas.

3.2 Parallel Image Restoration on Parallel and Distributed Computers

No estudo conduzido por Alessandro Bevilacqua *et al* em (BEVILACQUA; Loli Piccolomini, 2000), apresenta-se um modelo de aplicativo executado em programação paralela para recuperação de imagens. A restauração de imagens é um desafio relevante em diversas áreas, como a medicina, onde imagens podem sofrer degradação ao serem reproduzidas ou gravadas.

O algoritmo utilizado baseia-se no paralelismo de dados, na decomposição adaptativa de imagens e no método de regularização de Tikhonov. A execução foi realizada em um cluster de 6 *workstations* conectadas por uma rede *Ethernet* em um Cray T3E com 128 processadores.

Os resultados mostraram que, ao utilizar subproblemas, o número de tarefas em cada processador e o balanceamento de carga de trabalho, obteve-se menor tempo de execução total ao dividir em 24 subdomínios.

3.3 A Data and Task Parallel Image Processing Environment

No estudo de Cristina Nicolescu *et al* em (NICOLESCU; JONKER, 2002), que se baseia em *algorithmic skeletons*, incluindo *multi-baseline stereo vision*, foram realizados testes utilizando tarefas em unidades menores paralelas no processamento de imagens. O experimento ocorreu em um sistema computacional distribuído composto por um cluster Pentium Pro/200 MHz com 64Mb de RAM, rodando LinuxOs.

A conclusão do experimento destacou que a abordagem de paralelismo de tarefas tornou-se mais eficiente a partir de 16 processadores, indicando que a combinação de paralelismo de dados e tarefas é mais eficaz do que o uso exclusivo do paralelismo de dados.

3.4 Protótipo de Interação Humano-Computador para Processamento da Língua Natural em LLMs

(MAURER; ZAMBERLAN; COMPUTAÇÃO,) propôs o desenvolvimento de um protótipo de Interação Humano-Computador voltado para o Processamento de Linguagem Natural (PLN), utilizando modelos de linguagem de grande escala (LLMs) e integrando funcionalidades como reconhecimento de voz (STT) e síntese de fala (TTS). O diferencial da proposta foi a ênfase na execução offline, com o objetivo de preservar a privacidade e reduzir a dependência de conectividade com a internet.

Para atender a esses objetivos, o autor projetou uma interface interativa baseada em linha de comando e recursos gráficos que facilitassem a comunicação entre o usuário e o modelo de IA local. A IHC foi tratada como elemento central do sistema, sendo aplicada por meio de feedback visual, personalização da experiência, detecção de problemas e adaptação da resposta. O sistema também oferecia respostas em voz e texto, tornando a interação mais fluida e acessível.

A avaliação do sistema foi realizada com base em interações comparativas entre o protótipo offline e o ChatGPT Voice. Foram avaliados critérios como tempo de resposta, manutenção de contexto e adequação das respostas geradas. Apesar do tempo de resposta maior do protótipo, os resultados demonstraram boa capacidade de interação e adequação das funcionalidades propostas.

Esse trabalho relaciona-se com o presente projeto ao destacar a importância da Interação Humano-Computador na mediação entre usuário e sistemas inteligentes, com foco na interface como meio facilitador para execução de tarefas complexas. Ainda que os contextos de aplicação sejam diferentes (PLN vs. processamento de imagens), ambos compartilham a preocupação em proporcionar uma experiência de uso eficiente e compreensível ao usuário final.

3.5 Desenvolvimento de um Sistema de Gerenciamento de Tarefas com Ênfase na Experiência do Usuário

(CEZAR, 2023) desenvolveu um sistema de gerenciamento de tarefas com foco na experiência do usuário, buscando aplicar os princípios de Interação Humano-Computador (IHC) para facilitar o uso da aplicação por usuários com diferentes níveis de familiaridade com tecnologia. O sistema priorizou aspectos como usabilidade, acessibilidade e feedback visual, integrando elementos como menus intuitivos, botões de fácil entendimento e respostas claras às ações do usuário.

O autor fundamentou seu projeto com base em referências clássicas da IHC, destacando a importância de projetar interfaces que considerem as capacidades cognitivas dos usuários e promovam uma interação eficiente, eficaz e agradável. A interface desenvolvida foi validada por meio de testes com usuários reais, os quais contribuíram com sugestões que permitiram ajustes na disposição dos elementos e nos fluxos de interação.

Esse trabalho se relaciona ao presente projeto na medida em que ambos demonstram como a atenção aos princípios de usabilidade e ao design centrado no usuário contribui para o sucesso de sistemas computacionais. Embora as áreas de aplicação sejam distintas, a abordagem centrada na IHC evidencia um ponto comum entre as propostas.

4 Materiais e Métodos

Este trabalho adotou uma abordagem experimental orientada à reestruturação e otimização de um fluxo de processamento digital de imagens aplicado à análise de ovos de galinha. O ponto de partida foi um sistema funcional legado, desenvolvido por terceiros, cuja lógica foi cuidadosamente estudada, adaptada e aprimorada por meio da aplicação de técnicas de paralelismo com a linguagem Python.

4.1 Código Original (Sequencial)

A versão inicial do sistema consistia em um código sequencial que realizava, de forma linear, as etapas de recorte de subimagens, segmentação de contornos, cálculo de métricas morfológicas e geração de relatórios visuais e tabulares. Essa implementação utilizava bibliotecas consolidadas como OpenCV, NumPy e SciPy. No entanto, ao processar conjuntos com elevado número de ovos, o tempo de execução se tornava excessivo, limitando a aplicabilidade do sistema em cenários que exigem desempenho. Na Figura 7 é apresentado um trecho do código original sequencial utilizado como base para a refatoração.

```python
for egg in posEggs:
    (gr, ll, col, lin, larg, alt) = egg

    lIni = int(lin - round(alt * 0.075))
    lFin = int(lin + round(alt * 1.075))
    cIni = int(col - round(larg * 0.075))
    cFin = int(col + round(larg * 1.075))

    if (lIni < lMin):
        lMin = lIni
    if (cIni < cMin):
        cMin = cIni
    if (lFin > lMax):
        lMax = lFin
    if (cFin > cMax):
        cMax = cFin

    recImage = imgProcess[lIni:lFin, cIni:cFin]

    egg_num = str(nFile)
    egg_folder_fit_plot_path = Path( *args: plot_results_path, egg_num)
    check_and_create_directory_if_not_exist(egg_folder_fit_plot_path)

    resultado = process(recImage, factor: 1, pixFactor)

    # print(f"\n\nRESULTADO Processamento da imagem :\n {resultado}\n\n")
    imgProc, a, b, c, d, v, area, pAi, pAf, pBi, pBf = resultado
    cSEx = int(min(pAi[0], pAf[0], pBi[0], pBf[0]))
    cSEy = int(min(pAi[1], pAf[1], pBi[1], pBf[1]))

    cIDx = int(max(pAi[0], pAf[0], pBi[0], pBf[0]))
    cIDy = int(max(pAi[1], pAf[1], pBi[1], pBf[1]))

    rotateImage = recImage[cSEx: cIDx, cSEy: cIDy]

    try:
        termo1 = math.acos((b / 2) / (d)) * (d ** 2 / math.sqrt(d ** 2 - (b / 2) ** 2))
        termo2 = math.acos((b / 2) / c) * (c ** 2 / math.sqrt(c ** 2 - (b / 2) ** 2))
        vFormulaArea = 2 * math.pi * (b / 2) ** 2 + math.pi * (b / 2) * (termo1 + termo2)
    except ValueError:
        vFormulaArea = 1
    try:
        vFormulaVolume = (b / 2) ** 2 * ((2 * math.pi) / 3) * (d + c)
    except ValueError:
        vFormulaVolume = 1

    processedImageName = Path( *args: processed_path, f'{nFile}.png')
    rotateProcessedImageName = Path( *args: processed_path, f'rotate_{nFile}.png')

    leituraArquivo = f'{processedImageName},{a},{b},{c},{d},{v},{area},{vFormulaArea},{vFormulaVolume},{pixFactor}, {dFactor}\n'
    arqRelatorio.write(leituraArquivo)
    nFile += 1

    cv2.imwrite(str(processedImageName), imgProc)
    cv2.imwrite(str(rotateProcessedImageName), rotateImage)
```

Figura 7 – Trecho do código original sequencial utilizado como base para a refatoração, imagem tirada da versão original.

Fonte: o autor (2025). Código adaptado e analisado neste trabalho.

Ainda que esse código original não tenha sido desenvolvido neste presente trabalho, ele foi integralmente analisado, executado e documentado com o objetivo de compreender seu funcionamento interno, identificar gargalos computacionais e estabelecer as bases para a proposta de refatoração e otimização.

4.2 Refatoração com Paralelismo

Com base na análise do código legado, foram realizadas modificações estruturais relevantes, orientadas por princípios de modularização, clareza lógica e eficiência computacional. A principal mudança feita no projeto foi trocar a execução sequencial por uma abordagem paralela, utilizando a biblioteca concurrent.futures. E a classe *Process-*

PoolExecutor permite distribuir tarefas entre diferentes processos, aproveitando melhor os núcleos do processador e reduzindo o tempo total de execução.

A estratégia de paralelização foi implementada com o encapsulamento das operações intensivas em processamento (CPU-bound) — como segmentação de imagem, ajuste polinomial e cálculo volumétrico — na função cpu_process_egg(), projetada para ser executada de forma isolada em subprocessos independentes. Em contrapartida, tarefas associadas à entrada e saída de dados (I/O-bound), como a gravação de arquivos de imagem, planilhas e gráficos, permaneceram centralizadas na thread principal, com o intuito de preservar a integridade dos dados e evitar conflitos de concorrência.

Os resultados obtidos encontram-se detalhados no Capítulo 5, acompanhados de gráficos e tabelas que evidenciam a expressiva melhoria no tempo de resposta do sistema.

As subseções a seguir descrevem cada uma das etapas metodológicas do sistema, organizadas conforme sua execução prática no projeto.

O processo metodológico seguiu a seguinte sequência estruturada:

1. Aquisição das Imagens
2. Pré-processamento
3. Segmentação assistida
4. Desenvolvimento da solução
5. Processamento paralelo e extração de medidas
6. Geração e organização dos resultados

4.3 Aquisição das Imagens

As imagens utilizadas foram selecionadas manualmente a partir do armazenamento local, utilizando uma interface gráfica construída em Tkinter. A seleção ocorre por meio de diálogo interativo, garantindo flexibilidade ao usuário. Também é oferecida a alternativa de captura por URL, embora esta não tenha sido o foco dos testes realizados.

4.4 Pré-processamento

Assim que a imagem é carregada, ela é automaticamente redimensionada para se ajustar a uma resolução padrão, preservando a proporção original. O sistema também permite que o usuário realize uma calibração manual de escala, desenhando uma linha de referência na imagem, o que possibilita a conversão de pixels para centímetros (pixFactor), fundamental para os cálculos geométricos realizados posteriormente.

4.5 Segmentação assistida

O sistema permite que o usuário desenhe linhas verticais e horizontais com o mouse sobre a imagem exibida. Essas linhas atuam como delimitadores visuais e auxiliam na organização da análise, dividindo a imagem original em sub-regiões que correspondem a cada ovo individual.

Essa forma de interação contribui para separar adequadamente as regiões de interesse, facilitando a aplicação das etapas automáticas de segmentação. Todas as imagens utilizadas nos testes seguem um padrão semelhante: bandejas contendo 30 ovos organizados em linhas e colunas, formando uma grade visual regular. Esse padrão pode ser observado na Figura 8, apresentado na seção 4.7.

4.6 Desenvolvimento da solução

A solução foi dividida em dois módulos principais: um responsável pela interface com o usuário (main.py) e outro dedicado ao processamento das imagens (Processamento.py). As funcionalidades foram organizadas em botões gráficos, substituindo comandos de teclado, o que tornou a interface mais amigável e fácil de operar.

No estágio de análise das imagens, o sistema realiza a conversão para o espaço de cores HSV, aplica filtro de mediana para redução de ruído e identifica regiões conectadas compatíveis com o formato esperado dos ovos. A identificação é feita com base em critérios de proporção e área ocupada; em seguida, cada ovo é isolado em uma subimagem para processamento em paralelo. Essa separação permite explorar o paralelismo e acelerar o processamento como um todo, inclusive em máquinas com menos recursos.

4.7 Segmentação e extração de medidas

A principal melhoria em relação a implementações anteriores está no uso de paralelismo com *ProcessPoolExecutor*, que permite que os ovos sejam processados simultaneamente. Cada subimagem presente na figura 8 corresponde a um ovo que é enviada para um subprocesso independente, que realiza o processamento morfológico.

Durante esse processamento, são extraídos automaticamente os eixos A (maior comprimento), B (largura perpendicular), além dos segmentos C e D, que representam distâncias auxiliares a partir da interseção dos eixos principais. Essas informações são utilizadas na anotação das imagens e no relatório final.

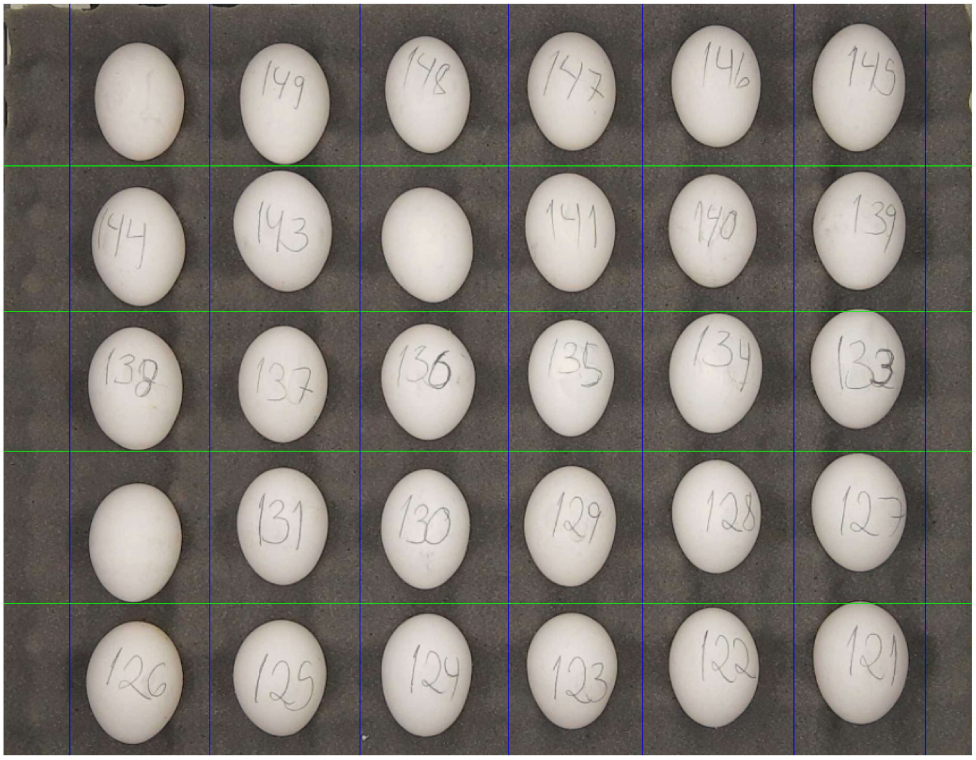


Figura 8 – Cartela contendo 30 ovos numerados manualmente, utilizada nos experimentos de segmentação e análise morfológica. As linhas sobre a imagem indicam os delimitadores visuais aplicados para auxiliar na separação das subimagens.

Fonte: o autor (2025)

5 Resultados e Discussão

O desenvolvimento da solução foi conduzido com o objetivo de otimizar o sistema existente para análise de imagens contendo ovos, aprimorando aspectos como desempenho, modularidade e usabilidade. Para isso, foram realizadas alterações estruturais no código-fonte, implementando processamento paralelo e reestruturando a interface gráfica. As seções a seguir detalham as principais etapas da implementação.

5.1 Arquitetura Modular do Sistema

A solução foi estruturada de forma modular em dois arquivos principais: main.py, responsável pela interface com o usuário e controle de fluxo, e Processamento.py, que executa o processamento computacional intensivo. Essa separação favorece a manutenção, a reutilização de componentes e a aplicação de técnicas de paralelismo com isolação de estado.

5.2 Organização do código principal (main.py)

O arquivo main.py desempenha um papel central no sistema, sendo responsável pela construção da interface gráfica, gerenciamento do fluxo de execução e controle da interação do usuário com a imagem carregada. Por meio da biblioteca Tkinter, a aplicação oferece menus gráficos e botões que permitem carregar imagens, acionar o processamento, alternar entre modos de visualização e encerrar a aplicação de forma controlada.

Além disso, uma das funções-chave do main.py é a *captureImage()*, responsável por carregar a imagem de entrada e iniciar o processo de interação gráfica com o usuário. Ela realiza a leitura da imagem, aplica redimensionamento com base nas dimensões ajustadas pelo sistema, inicializa o loop de exibição do OpenCV e ativa o menu interativo com os botões disponíveis. No Algoritmo 1 está descrita a função `captureImage()`, responsável pela captura e exibição da imagem com interação gráfica.

Algorithm 1 Função `captureImage(source)`: captura e exibe imagem com interação

```
def captureImage(source):
    #Captura uma imagem de um arquivo local ou de uma URL e abre
       o menu interativo.
    global fullImage, baseImage, originalImage, totalLines,
       totalColumns, rFactor, nomeArquivo
    global opencv_running, elementLines, tempLines,
       calibrationLine, menu_active

    elementLines.clear()
    tempLines.clear()
    calibrationLine.clear()
    menu_active = True  # Ativa o menu interativo somente apos
       abrir uma imagem

    if source == 1:
        nomeArquivo = dlg.askopenfilename()
        if nomeArquivo:
            fullImage = cv2.imread(nomeArquivo)
            status_message.set("Imagem carregada de arquivo")
    else:
        status_message.set("Fonte n o reconhecida")
        return

    if fullImage is not None:
        totalLines, totalColumns, rFactor = Processamento.
           adjustImageDimension(fullImage)
        down_points = (totalColumns, totalLines)
        # Cria a imagem base limpa (nunca modificada)
        baseImage = cv2.resize(fullImage, down_points,
           interpolation=cv2.INTER_LINEAR)
        # originalImage n o ser  alterada; ela pode ser usada
           apenas para refer ncia
        originalImage = baseImage.copy()
        status_message.set("Imagem carregada com sucesso")
        threading.Thread(target=run_opencv_loop, daemon=True).
           start()
        show_interactive_menu() # Exibe o menu interativo junto
           com a imagem
```

5.3 Interface Gráfica Interativa

A interface gráfica foi desenvolvida com a biblioteca Tkinter, substituindo comandos de teclado por botões interativos, o que torna o sistema mais acessível a usuários não técnicos. A usabilidade foi considerada através da organização funcional dos menus, feedback visual imediato e rótulos autoexplicativos. Isso contribuiu para uma experiência de uso mais fluida e intuitiva. A tela inicial do menu interativo pode ser vista na Figura

9. O menu principal é exibido na função *main_menu()*, que organiza a janela e associa os botões às respectivas ações.

Carregar Imagem de Arquivo – permite selecionar uma imagem local;

Capturar Imagem via URL – permite informar uma URL de imagem externa

Alternar Modo de Vídeo – alterna para modo de captura contínua (se aplicável)

Sair – encerra a execução da aplicação.

Figura 9 – Tela inicial do menu interativo, exibindo os principais botões de interação desenvolvidos em Tkinter.

Fonte: o autor (2025)

```python
def main_menu():
    global status_message
    root = Tk()
    root.title("Menu Interativo - Processamento de Imagem")
    root.geometry("375x500")  # Define um tamanho fixo adequado para monitores modernos

    status_message = StringVar()
    status_message.set("Pronto")

    menu_bar = Menu(root)
    help_menu = Menu(menu_bar, tearoff=0)
    help_menu.add_command(label="Comandos Disponíveis", command=show_help)
    help_menu.add_command(label="Tela Cheia", command=lambda: toggle_fullscreen(root))
    menu_bar.add_cascade(label="Ajuda", menu=help_menu)
    root.config(menu=menu_bar)

    frame = Frame(root, bg="#e3f2fd")
    frame.pack(fill="both", expand=True, padx=20, pady=20)

    Button(frame, text="📂 Carregar Imagem de Arquivo", command=lambda: captureImage(1), height=2, width=40,
           bg="#bbdefb", fg="#0d47a1", activebackground="#90caf9", activeforeground="#0d47a1",
           font=("Arial", 12, "bold")).pack(pady=10)
    Button(frame, text="🌐 Capturar Imagem via URL", command=lambda: captureImage(0), height=2, width=40, bg="#c8e6c9",
           fg="#1b5e20", activebackground="#a5d6a7", activeforeground="#1b5e20", font=("Arial", 12, "bold")).pack(
        pady=10)
    Button(frame, text="📽️ Alternar Modo de Vídeo", command=toggle_video_mode, height=2, width=40, bg="#ffe0b2",
           fg="#e65100", activebackground="#ffcc80", activeforeground="#e65100", font=("Arial", 12, "bold")).pack(
        pady=10)
    Button(frame, text="❌ Sair",
           command=lambda: threading.Thread(target=exit_program, args=(root,), daemon=True).start(), height=2, width=40,
           bg="#ffcdbc", fg="#b71c1c", activebackground="#ffab91", activeforeground="#b71c1c",
           font=("Arial", 12, "bold")).pack(pady=10)

    status_label = Label(root, textvariable=status_message, bg="#e3f2fd", fg="#0d47a1", font=("Arial", 10))
    status_label.pack(side="bottom", fill="x", pady=10)

    root.mainloop()
```

Figura 10 – Código da função *main_menu()* responsável por organizar a interface gráfica interativa com botões definidos em Tkinter.

Fonte: o autor (2025)

A função main_menu(), responsável por organizar a interface gráfica interativa, está representada na Figura 10. Essa reformulação da interface foi essencial para tornar o sistema mais amigável e funcional, permitindo que o usuário avance pelas etapas sem necessidade de comandos técnicos.

5.4 Janela de ações da imagem carregada

A Figura 11 apresenta os botões exibidos após o carregamento da imagem, uma nova janela é exibida com botões adicionais que permitem ações diretas sobre a imagem. Essas ações incluem iniciar o processamento, apagar a última linha traçada, salvar a imagem atual e encerrar a aplicação. Esses botões tornam a interação com a imagem mais prática e reforçam a acessibilidade do sistema para usuários finais.

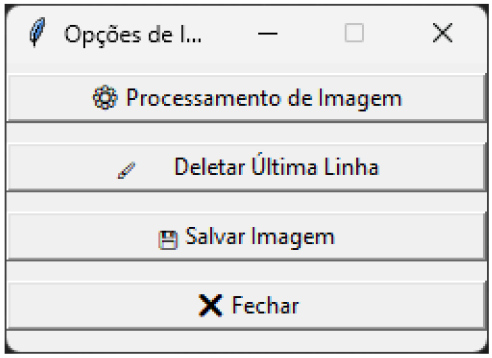


Figura 11 – Janela de ações após o carregamento da imagem, permitindo controle direto sobre o processamento.

Fonte: o autor (2025)

Cada botão é vinculado a uma função responsável por executar a ação correspondente. Na figura 12 está representada a definição dos botões da interface gráfica implementados em Tkinter, onde são especificadas as funções associadas às ações de processamento, exclusão de linhas, salvamento de imagens e encerramento da aplicação

```
Button(interactive_menu, text="⚙ Processamento de Imagem", command=processamento_action).pack(pady=5, fill='x')
Button(interactive_menu, text="🖉 Deletar Última Linha", command=delete_last_line).pack(pady=5, fill='x')
Button(interactive_menu, text="💾 Salvar Imagem", command=save_image).pack(pady=5, fill='x')
Button(interactive_menu, text="❌ Fechar", command=close_everything).pack(pady=5, fill='x')
```

Figura 12 – Definição dos botões da interface gráfica em Tkinter.

Fonte: o autor (2025)

5.5 Função update_image_cache()

A função *update_image_cache()* foi criada para centralizar a lógica de redimensionamento e exibição da imagem carregada. Ela calcula a escala da imagem para exibição sem distorções e atualiza variáveis globais que serão usadas por outras funções que dependem da resolução real da imagem. Esse controle é essencial para que os cliques do usuário no modo de segmentação sejam precisos e coordenados corretamente com o conteúdo visualizado.

5.6 Aplicação do Paralelismo com ProcessPoolExecutor

A função cpu_process_egg(), definida no arquivo Processamento.py, concentra a lógica de processamento computacional intensivo de cada ovo. Essa função é executada em paralelo por meio de múltiplos subprocessos utilizando a classe *ProcessPoolExecutor*, a qual realiza operações como recorte das subimagens, cálculo dos eixos morfológicos (A, B, C e D), rotação e anotação visual.

A chamada paralela dessa função ocorre dentro de subImagens(), enquanto as demais etapas do fluxo como o salvamento de gráficos, imagens e planilhas, permanecem na thread principal (não paralelizada). Essa divisão entre tarefas CPU-bound e I/O-bound foi crucial para obter ganhos reais de desempenho, evitando conflitos de concorrência e sobrecarga desnecessária. A execução paralela das subimagens foi implementada por meio da função *cpu_process_egg()*, distribuída em múltiplos subprocessos com a classe *ProcessPoolExecutor*. Esse procedimento está descrito no Algoritmo 2.

Algorithm 2 Execução paralela das subimagens com `ProcessPoolExecutor`

```
# Define o n mero de processos paralelos (CPU cores - 1)
maxw = max(1, multiprocessing.cpu_count() - 1)

# Cria o executor com m ltiplos subprocessos
with ProcessPoolExecutor(max_workers=maxw) as exe:

    # Distribui as tarefas de processamento dos ovos
    results = list(exe.map(cpu_process_egg, jobs))
```

A escolha pela expressão max(1, multiprocessing.cpu_count() - 1) visa evitar o uso total dos núcleos disponíveis. Essa prática reserva ao menos um núcleo para o sistema operacional e para tarefas de segundo plano, prevenindo sobrecarga e mantendo a responsividade geral do ambiente durante a execução paralela do algoritmo. Essa abordagem é especialmente importante em sistemas com menor capacidade computacional ou sob carga elevada.

Em termos técnicos, a execução paralela é aplicada exclusivamente às tarefas intensivas de CPU, deixando as operações de entrada e saída (como escrita de arquivos) fora do escopo de paralelismo, o que segue as boas práticas de otimização para workloads mistos.

Essa abordagem é explorada em profundidade na seção 5.8, onde se observam os impactos práticos da paralelização nas diferentes arquiteturas testadas.

5.7 Comparação com estrutura anterior

Em relação a versão anterior do sistema, a implementação atual demonstra avanços significativos no que tange à organização lógica, modularidade e desempenho computacional. Originalmente, o código apresentava-se de forma monolítica, com grande parte da lógica de processamento concentrada em blocos únicos, o que dificultava sua manutenção, extensão e reutilização.

A versão refatorada introduziu uma arquitetura modular, na qual cada componente funcional do sistema foi encapsulado em funções específicas. Exemplos notáveis

dessa modularização incluem os métodos main_menu(), responsável pela construção da interface gráfica principal; cpu_process_egg(), que executa o processamento paralelo dos ovos; e update_image_cache(), utilizada para a atualização e exibição da imagem base.

Essa estrutura favoreceu não apenas a clareza do código, como também a aplicação eficiente de técnicas de paralelismo, viabilizadas pela separação entre tarefas CPU-bound e I/O-bound. Além disso, tornou possível a reutilização de componentes em diferentes partes do projeto e a aplicação de testes e melhorias incrementais, com impacto direto na performance do sistema.

5.8 Análise Comparativa de Desempenho

Com o objetivo de avaliar o impacto do paralelismo no desempenho do processamento de imagens, foram conduzidos testes práticos em dois notebooks com arquiteturas de processadores distintas. A análise baseia-se na comparação entre os tempos de execução obtidos nos modos com e sem paralelismo, utilizando um conjunto de dez testes para cada processador.

Os processadores avaliados foram:

- **Intel(R) Core(TM) i5-6200U**: 2 núcleos físicos, 4 threads, frequência de 2,40 GHz.
- **Intel(R) Core(TM) i3-N305**: 8 núcleos físicos, 8 threads, frequência de 1,80 GHz.

Os resultados estão organizados nas Tabelas 1 e 2 que reúnem os tempos de execução obtidos nos dois processadores avaliados, sobre uma imagem distinta, contendo 30 ovos dispostos em bandejas. As imagens foram numeradas sequencialmente para fins de identificação, sendo utilizadas entradas distintas em todos os testes, o que garante diversidade de conteúdo e volume de dados processados.

5.8.1 Fórmulas utilizadas (redução e média)

A redução percentual do tempo foi calculada utilizando a fórmula abaixo, onde T_{sem} representa o tempo de execução sem paralelismo e T_{com} representa o tempo com paralelismo:

$$Redução(\%) = \left(\frac{T_{sem} - T_{com}}{T_{sem}}\right) \times 100 \tag{5.1}$$

Para consolidar os dados obtidos, foi realizada a média aritmética dos tempos de execução com e sem paralelismo para cada processador. O cálculo da média foi feito com base na seguinte fórmula:

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_n}{n} \tag{5.2}$$

Onde $\bar{x}$ representa a média dos tempos, x_i representa o tempo de execução de cada teste, e n é o número total de testes (n = 10).

5.8.2 Resultados experimentais por processador

Na Tabela 1 são apresentados os resultados de desempenho no processador Intel Core i5-6200U, enquanto a Tabela 2 reúne os resultados no Intel Core i3-N305.

Tabela 1 – Desempenho no Intel Core i5-6200U

Imagem	**Tempo sem paralelismo**	**Tempo com paralelismo**	**Redução (%)**
1	72.96 s	41.14 s	43.64%
2	118.22 s	75.15 s	36.42%
3	209.43 s	90.86 s	56.60%
4	135.67 s	75.45 s	44.37%
5	157.89 s	89.27 s	43.47%
6	166.41 s	78.32 s	52.93%
7	140.32 s	80.18 s	42.86%
8	194.75 s	94.88 s	51.28%
9	173.66 s	97.71 s	43.75%
10	187.92 s	84.97 s	54.78%

Fonte: o autor (2025)

Tabela 2 – Desempenho no Intel Core i3-N305

Imagem	**Tempo sem paralelismo**	**Tempo com paralelismo**	**Redução (%)**
1	74.74 s	52.66 s	29.55%
2	71.83 s	48.77 s	32.10%
3	77.54 s	52.02 s	32.93%
4	85.33 s	58.56 s	31.37%
5	89.72 s	63.05 s	29.71%
6	83.48 s	58.17 s	30.30%
7	91.62 s	63.17 s	31.03%
8	87.23 s	59.74 s	31.55%
9	79.91 s	56.71 s	29.06%
10	86.78 s	59.88 s	30.98%

Fonte: o autor (2025)

5.8.3 Cálculo das Médias

A Tabela 3 apresenta os resultados médios para o processador Intel Core i5-6200U, enquanto a Tabela 4 apresenta os dados correspondentes ao Intel Core i3-N305. A redução

percentual do tempo foi calculada a partir das médias, permitindo uma comparação direta do impacto do paralelismo entre os dois processadores.

Tabela 3 – Cálculo das médias de tempo de execução no processador Intel Core i5-6200U

Modo de Execução	**Média (s)**
Sem paralelismo	155.63
Com paralelismo	80.39
Redução percentual	**52.06%**

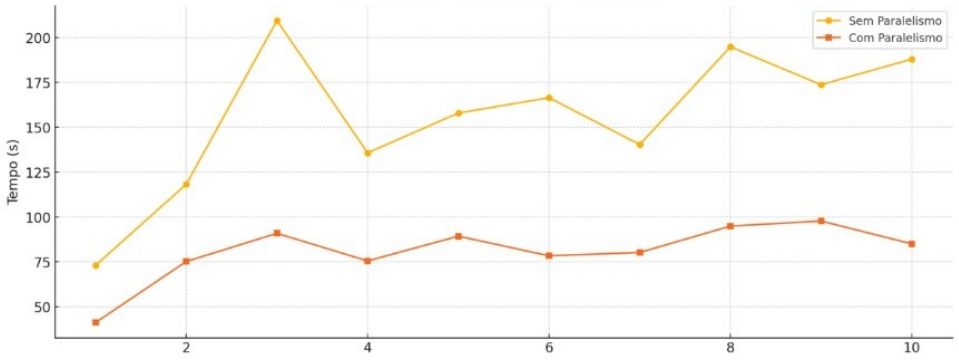
Tabela 4 – Cálculo das médias de tempo de execução no processador Intel Core i3-N305

Modo de Execução	**Média (s)**
Sem paralelismo	82.75
Com paralelismo	57.08
Redução percentual	**31.46%**

5.8.4 Gráficos Comparativos

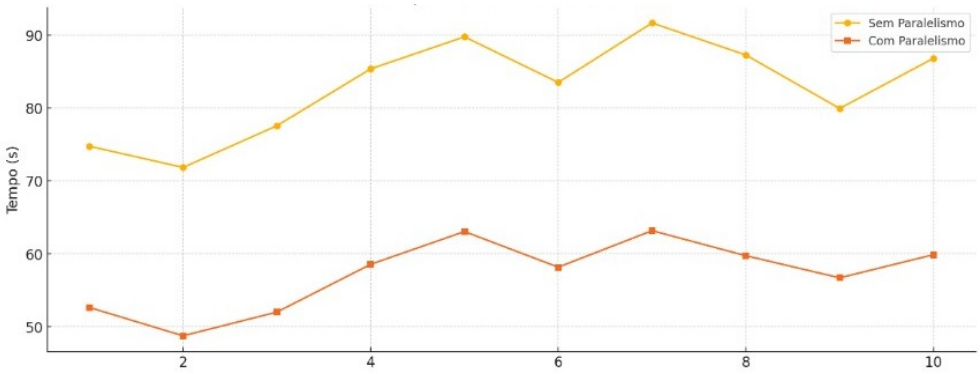
As Figuras 13, 14 e 15 apresentam a comparação de desempenho entre as execuções com e sem paralelismo nos dois processadores avaliados. Esses gráficos evidenciam os tempos de execução individuais e a redução percentual obtida, oferecendo uma visão objetiva dos efeitos da paralelização.

Figura 13 – Desempenho com e sem paralelismo – Intel Core i5-6200U



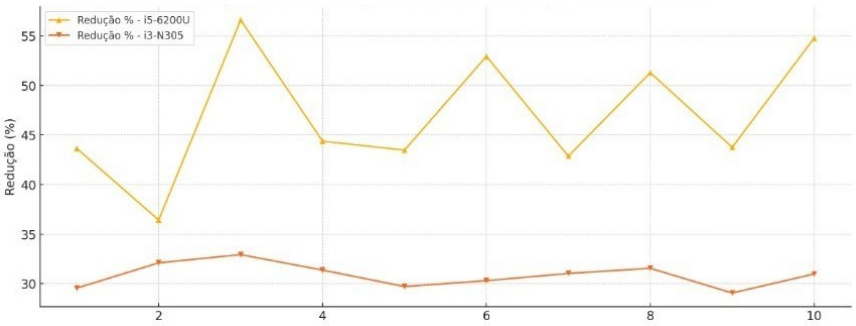
Fonte: o autor (2025)

Figura 14 – Desempenho com e sem paralelismo – Intel Core i3-N305



Fonte: o autor (2025)

Figura 15 – Comparação da redução percentual entre os dois processadores



Fonte: o autor (2025)

5.8.5 Discussão

Os experimentos evidenciaram ganhos consistentes com a paralelização nos dois processadores avaliados. O Intel Core i3-N305, com oito núcleos e maior paralelismo intrínseco, apresentou menores tempos absolutos de execução. Já o Intel Core i5-6200U, com dois núcleos físicos e quatro threads, exibiu redução percentual mais expressiva após a aplicação do paralelismo. Em plataformas com menos núcleos, a fração paralelizável do algoritmo representa um ganho proporcionalmente maior sobre o tempo sequencial, enquanto em plataformas mais paralelas o benefício absoluto é maior, porém a taxa percentual tende a ser menor devido ao impacto relativo de overheads (criação/coordenação de processos, movimentação de dados) e de limitações de memória/banda.

Esses resultados indicam que o número absoluto de núcleos físicos não é o único determinante da eficiência do paralelismo. Outros fatores, como a frequência de operação, a arquitetura interna do processador, a carga do sistema e o equilíbrio entre tarefas CPU-bound e I/O-bound, influenciam diretamente o desempenho final.

Dessa forma, confirma-se que a abordagem de paralelização adotada se mostrou eficaz e adaptável em diferentes contextos computacionais, trazendo ganhos de desempenho

relevantes mesmo em arquiteturas mais modestas.

6 Conclusão

Este projeto teve como objetivo aprimorar o desempenho de um sistema de processamento digital de imagens voltado à análise morfológica de ovos de galinha, por meio da reestruturação do código-fonte e da aplicação de técnicas de paralelismo. A motivação partiu da constatação de que a versão original apresentava elevado tempo de execução, comprometendo seu desempenho.

A solução foi estruturada com foco na modularidade do código e na adoção de paralelismo via a classe *ProcessPoolExecutor*, além da reformulação da interface gráfica com ênfase em usabilidade.

Os testes demonstraram reduções expressivas no tempo de execução em ambos os processadores, confirmando a eficácia da paralelização com o uso da classe *ProcessPoolExecutor*. Observou-se que o Intel Core i3-N305, por possuir mais núcleos físicos, apresentou tempos absolutos menores de execução, enquanto o Intel Core i5-6200U obteve reduções percentuais mais elevadas. Isso confirma que a aplicação do paralelismo trouxe melhorias relevantes, mesmo em computadores com menor capacidade, mostrando que a solução proposta é flexível e eficaz em diferentes contextos.

Como perspectiva para trabalhos futuros, recomenda-se a realização de testes em equipamentos com maior capacidade computacional, a fim de avaliar o comportamento do sistema em ambientes com múltiplos núcleos de alto desempenho. Além disso, seria pertinente ampliar o volume de dados processados, utilizando imagens com mais de 30 ovos, para validar a escalabilidade da solução e investigar possíveis gargalos em cenários de carga intensiva.

Referências

AMOS, D. Python gui programming with tkinter. **Real Python**, 2020. Citado na página 21.

BENSAKHRIA, A. **Accelerating Higgs Boson Signal Classification Using Advanced Computing Platforms**. 2023. Citado 2 vezes nas páginas 5 e 14.

BEVILACQUA, A.; Loli Piccolomini, E. Parallel image restoration on parallel and distributed computers. **Parallel Computing**, v. 26, n. 4, p. 495–506, 2000. ISSN 0167-8191. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167819199001155>. Citado na página 24.

CEZAR, M. d. L. A. **Desenvolvimento de um Sistema de Gerenciamento de Tarefas com Ênfase na Experiência do Usuário**. 2023. Trabalho de Conclusão de Curso – Universidade Estadual da Paraíba. Acesso em: 17 maio 2025. Disponível em: <https://dspace.bc.uepb.edu.br/jspui/bitstream/123456789/31021/2/TCC%20-%20Matheus%20de%20Lima%20Alves%20Cezar>. Citado na página 25.

Codecademy Team. **Understanding the Global Interpreter Lock (GIL) in Python**. 2025. <https://www.codecademy.com/article/understanding-the-global-interpreter-lock-gil-in-python>. Acesso em: 8 jul. 2025. Citado 2 vezes nas páginas 5 e 16.

FOUNDATION, P. S. **Python 3 documentation: concurrent.futures**. 2024. Acesso em: 06 maio 2025. Disponível em: <https://docs.python.org/3/library/concurrent.futures.html>. Citado na página 22.

GONNORD, L.; HENRIO, L.; MOREL, L.; RADANNE, G. A survey on parallelism and determinism. **arXiv**, 10 2022. Survey sobre modelos de programação paralela e determinismo. Disponível em: <https://arxiv.org/abs/2210.15202>. Citado na página 10.

HEROUX, M. A.; BERNHOLDT, D. E.; MCINNES, L. C.; RAYBOURN, E. M. **Best Practices for HPC Software Developers (HPC-BP) Webinar Series**. 2022. <https://ideas-productivity.org/resources/series/hpc-best-practices-webinars/>. IDEAS Productivity / Exascale Computing Project (ECP). Recurso institucional sobre boas práticas de desenvolvimento de software HPC. Citado na página 13.

MAURER, P. G. G.; ZAMBERLAN, A. de O.; COMPUTAÇÃO, C. d. C. da. Protótipo de interação humano-computador para processamento da língua natural em llms. Citado na página 24.

MORDVINTSEV, A.; ABID, K. Opencv-python tutorials documentation. **Obtenido de https://media. readthedocs. org/pdf/opencv-python-tutroals/latest/opencv-python-tutroals. pdf**, 2014. Citado na página 21.

NGUYEN, Q. **Mastering Concurrency in Python: Create faster programs using concurrency, asynchronous, multithreading, and parallel programming**. [S.l.]: Packt Publishing Ltd, 2018. Citado 3 vezes nas páginas 13, 14 e 15.

NICOLESCU, C.; JONKER, P. A data and task parallel image processing environment. **Parallel Computing**, v. 28, n. 7, p. 945–965, 2002. ISSN 0167-8191. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167819102001059>. Citado na página 24.

NIELSEN, J. Heuristics for user interface design. **Retrieved Feb**, v. 16, p. 2011, 2010. Citado 2 vezes nas páginas 19 e 20.

Nouer Uz Zaman. **Difference between Multiprocessing and Multithreading**. 2023. Https://medium.com/@noueruzzaman/tug-of-war-multiprocessing-vs-multithreading-55341c1f2103. Acesso em: 8 jul. 2025. Citado 2 vezes nas páginas 5 e 15.

Python Software Foundation. **Python 3 documentation – Global Interpreter Lock (GIL)**. 2024. <https://docs.python.org/3/glossary.html#term-global-interpreter-lock>. Acesso em: 10 ago. 2024. Citado 2 vezes nas páginas 16 e 17.

_____. **Python FAQ — Why doesn't CPython run multiple threads at once?** 2024. Documentação oficial do Python 3. Disponível em: <https://docs.python.org/3/faq/general.html#why-doesn-t-cpython-run-multiple-threads-at-once>. Citado na página 15.

RHO, T.-G.; CHO, B.-K. Non-destructive evaluation of physicochemical properties for egg freshness: A review. **Agriculture**, v. 14, n. 11, p. 2049, 2024. Disponível em: <https://www.mdpi.com/2077-0472/14/11/2049>. Citado na página 11.

SAITO, P. T. M. Otimização do processamento de imagens médicas utilizando a computação paralela. 2007. Citado na página 23.

SHNEIDERMAN, B. **Designing the User Interface: Strategies for Effective Human-Computer Interaction**. 3rd. ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1997. ISBN 0201694972. Citado 2 vezes nas páginas 5 e 20.

SHNEIDERMAN, B.; PLAISANT, C.; COHEN, M.; JACOBS, S.; ELMQVIST, N. **Designing the User Interface: Strategies for Effective Human-computer Interaction**. Pearson, 2017. (Global edition). ISBN 9781292153919. Disponível em: <https://books.google.com.br/books?id=nhDYtQEACAAJ>. Citado na página 19.

SRINATH, K. Python–the fastest growing programming language. **International Research Journal of Engineering and Technology**, v. 4, n. 12, p. 354–357, 2017. Citado na página 21.

STRECK, L. F. da Silva Silva e Andrea S. Charão Charão e Romulo P. Benedetti Benedetti e N. A. S. Explorando paralelismo em python para múltiplas execuções de modelos de culturas agrícolas. **Teste2**, 2018. Disponível em: <http://143.54.25.88/index.php/teste2/article/view/1597>. Citado 3 vezes nas páginas 5, 17 e 18.

SZELISKI, R. **Computer Vision: Algorithms and Applications**. 2. ed. Cham: Springer, 2022. ISBN 978-3-030-34371-2. Disponível em: <https://link.springer.com/book/10.1007/978-3-030-34372-9>. Citado na página 10.

TANENBAUM, A. S.; BOS, H. **Modern Operating Systems**. 5. ed. Boston: Pearson, 2022. ISBN 978-0137618873. Disponível em: <https://www.pearson.com/en-us/pearsonplus/p/9780137618880>. Citado 2 vezes nas páginas 13 e 16.

TULCHAK, L.; RCHUK, History of python. 2016. Citado na página 21.

WALT, S. Van der; SCHÖNBERGER, J. L.; NUNEZ-IGLESIAS, J.; BOULOGNE, F.; WARNER, J. D.; YAGER, N.; GOUILLART, E.; YU, T. scikit-image: image processing in python. **PeerJ**, PeerJ Inc., v. 2, p. e453, 2014. Citado na página 22.

Apêndices

APÊNDICE A – CÓDIGOS FONTE

As seções a seguir reúnem os códigos-fonte essenciais das implementações desenvolvidas neste trabalho (processamento de imagens e interface gráfica), organizados por módulo.

A.1 main.py (interface gráfica)

```
import os
from tkinter import Tk, Text, mainloop, Button, Menu, filedialog
   as dlg, messagebox, Frame, Label, StringVar, Toplevel
import cv2
import numpy as np
import sys
import urlopen
import Processamento
import threading
import time

# Lists to store the Lines coordinators
elementLines = []
calibrationLine = []
tempLines = []
clicked = 0
totalLines = 0
totalColumns = 0
pixFactor = 0.247525
dFactor = 1
auxFactor = 1
videoMode = -1
fullImage = None
baseImage = None
originalImage = None
temp_image = None
nomeArquivo = "" # Vari vel para armazenar o nome do arquivo
opencv_running = False #Vari vel para controlar o loop do OpenCV
menu_active = False #Vari vel para controlar a exibi  o do
   menu interativo
```

```
interactive_menu = None  # Vari vel global para armazenar o menu
    interativo

def captureImage(source):
    """Captura uma imagem de um arquivo local ou de uma URL e
       abre o menu interativo."""
    global fullImage, baseImage, originalImage, totalLines,
       totalColumns, rFactor, nomeArquivo
    global opencv_running, elementLines, tempLines,
       calibrationLine, menu_active

    elementLines.clear()
    tempLines.clear()
    calibrationLine.clear()
    menu_active = True  # Ativa o menu interativo somente ap s
       abrir uma imagem

    if source == 1:
        nomeArquivo = dlg.askopenfilename()
        if nomeArquivo:
            fullImage = cv2.imread(nomeArquivo)
            status_message.set("Imagem␣carregada␣de␣arquivo")
    else:
        status_message.set("Fonte␣n o␣reconhecida")
        return

    if fullImage is not None:
        totalLines, totalColumns, rFactor = Processamento.
           adjustImageDimension(fullImage)
        down_points = (totalColumns, totalLines)
        # Cria a imagem base limpa (nunca modificada)
        baseImage = cv2.resize(fullImage, down_points,
           interpolation=cv2.INTER_LINEAR)
        # originalImage n o ser  alterada; ela pode ser usada
           apenas para refer ncia
        originalImage = baseImage.copy()
        status_message.set("Imagem␣carregada␣com␣sucesso")
        threading.Thread(target=run_opencv_loop, daemon=True).
           start()
        show_interactive_menu()  # Exibe o menu interativo junto
           com a imagem
    else:
```

```
        status_message.set("Erro ao carregar a imagem")

def show_interactive_menu(): #Fun   o para exibir o menu
    interativo
     global interactive_menu
     interactive_menu = Toplevel()
     interactive_menu.title("Op   es de Imagem")
     interactive_menu.geometry("250x150")
     interactive_menu.resizable(False, False)

     def delete_last_line(): #Fun   o para deletar a ultima linha
         global elementLines, tempLines
         if elementLines:
             elementLines.pop()
         # Limpa tamb m a lista de linhas tempor rias
         tempLines.clear()
         update_image_cache()

     def save_image(): #Fun   o para salvar a imagem
         if fullImage is not None:
             fileNameSave = dlg.asksaveasfilename(confirmoverwrite
                =False)
             if fileNameSave:
                 cv2.imwrite(fileNameSave, fullImage)
         interactive_menu.destroy()

     def close_everything():
         global opencv_running, interactive_menu

         print("    Encerrando o programa...")

         # 1. Parar o loop do OpenCV imediatamente
         opencv_running = False

         # 2. Fechar todas as janelas OpenCV corretamente
         if cv2.getWindowProperty("Window", cv2.WND_PROP_VISIBLE)
            >= 0:
             print("    Fechando OpenCV...")
             cv2.destroyAllWindows()
             cv2.waitKey(1)  # Pequeno delay para garantir
                fechamento
```

```
    # 3. Fechar o menu interativo, se ainda estiver aberto
    if interactive_menu is not None:
        print("    Fechando menu interativo...")
        try:
            interactive_menu.destroy()
        except Exception as e:
            print(f"      Erro ao fechar menu interativo: {e
                }")

        interactive_menu = None

def processamento_action():  # Fun   o para processamento de
    imagem
    global pixFactor, nomeArquivo, fullImage, originalImage,
        totalLines, totalColumns, rFactor, dFactor,
        elementLines
    pixFactor = 0.250417362270450751252086811352 25
    #teste nome arquivo
    t0 = time.perf_counter()
    print("Nome Arquivo antes da chamada de processamento ==>
        ", nomeArquivo)
    Processamento.subImagens(fullImage, totalColumns,
        totalLines, rFactor, pixFactor, dFactor, nomeArquivo)
    print(f"Tempo total de subImagens: {time.perf_counter() -
        t0:.2f}s")
    update_image_cache()
    cv2.namedWindow("Window")  # Cria a janela
    cv2.setMouseCallback("Window", mouseActions)


interactive_menu.protocol("WM_DELETE_WINDOW", lambda:
    threading.Thread(target=close_everything, daemon=True).
    start())

Button(interactive_menu, text="      Processamento de Imagem
    ", command=processamento_action).pack(pady=5, fill='x')
Button(interactive_menu, text="       Deletar  ltima  Linha"
    , command=delete_last_line).pack(pady=5, fill='x')
Button(interactive_menu, text="    Salvar Imagem", command=
    save_image).pack(pady=5, fill='x')
Button(interactive_menu, text="   Fechar", command=
    close_everything).pack(pady=5, fill='x')
```

```
def close_interactive_menu():
    global interactive_menu, status_message

    if interactive_menu is not None:
        status_message.set("Fechando␣menu␣interativo")

        try:
            # Garantir que a janela do menu seja fechada antes de
                definir como None
            interactive_menu.destroy()
        except Exception as e:
            print(f"      ␣Erro␣ao␣fechar␣menu␣interativo:␣{e}")

        interactive_menu = None  # Remover refer ncia para
           evitar novos acessos

def update_image_cache():
    global baseImage, totalColumns, totalLines
    if baseImage is not None:
        temp_image = baseImage.copy()
        Processamento.drawLines(temp_image, calibrationLine,
           tempLines, elementLines, totalColumns, totalLines)
        cv2.imshow("Window", temp_image)
        cv2.waitKey(1)

def retrieve_input(textBox): # Esta fun   o obt m o valor de
   entrada de um Text widget do tkinter
    global pixFactor
    inputValue = textBox.get("1.0", "end-1c")
    pixFactor = int(inputValue)
    status_message.set(f"Fator␣de␣pixel␣atualizado␣para␣{
       pixFactor}")
    textBox.quit()

def capture_distance(): # Fun   o para capturar a dist ncia
   entre dois pontos
    root = Tk()
    root.title('Informe␣refer ncia␣em␣cm')
    root.geometry("300x80")
    textBox = Text(root, height=2, width=10)
```

```
    textBox.pack()
    buttonCommit = Button(root, height=1, width=10, text="
       Confirma", command=lambda: retrieve_input(textBox))
    buttonCommit.pack()
    mainloop()
    root.destroy()

def mouseActions(action, x, y, flags, *userdata): # Fun   o para
    a   es do mouse
    # Referencing global variables
    global elementLines, tempLines, originalImage, clicked,
       totalLines, totalColumns, calibrationLine, pixFactor,
       dFactor, auxFactor
    # Mark the top left corner when left mouse button is pressed

    if action == cv2.EVENT_LBUTTONDOWN:
        if clicked == 3:
            calibrationLine.append((x, y, clicked))
            calibrationLine.append((x, y, clicked))
        else:
            clicked = 1
            tempLines = [(y, clicked)]
            # When left mouse button is released, mark bottom
               right corner
    elif action == cv2.EVENT_LBUTTONUP:
        if clicked == 3:
            calibrationLine[1] = (x, y, clicked)
            xi, yi, c = calibrationLine[0]
            xf, yf, c = calibrationLine[1]
            calibrationLine = []
            clicked = 0
            try:
                dX = xf * (1 / auxFactor) - xi * (1 / auxFactor)
                dY = yf * (1 / auxFactor) - yi * (1 / auxFactor)
                dFactor = (dX ** 2 + dY ** 2) ** 0.5
                print(f'Medida␣em␣Pixel␣=>␣{dFactor}')
                capture_distance()
                pixFactor = pixFactor / dFactor
            except ValueError:
                pixFactor = 1
        else:
            elementLines.append((y, clicked))
```

```
            clicked = 0
            # Draw the rectangle
        elif action == cv2.EVENT_RBUTTONDOWN:
            clicked = 2
            tempLines = [(x, clicked)]
        elif action == cv2.EVENT_RBUTTONUP:
            elementLines.append((x, clicked))
            clicked = 0
        elif action == cv2.EVENT_MOUSEMOVE:
            if clicked != 0:
                tempLines = []
                if clicked == 3 and len(calibrationLine) > 1:
                    calibrationLine[1] = (x, y, clicked)
                if clicked == 1:
                    tempLines = [(y, clicked)]
                if clicked == 2:
                    tempLines = [(x, clicked)]

def run_opencv_loop(): #Fun  o para rodar o loop do OpenCV
    global originalImage, opencv_running
    opencv_running = True
    cv2.namedWindow("Window", cv2.WINDOW_NORMAL)
    cv2.setMouseCallback("Window", mouseActions)

    while opencv_running:
        if originalImage is not None:
            temp_image = originalImage.copy()
            Processamento.drawLines(temp_image, calibrationLine,
               tempLines, elementLines, totalColumns, totalLines)
            cv2.imshow("Window", temp_image)

        key = cv2.waitKey(1)
        if key == 113 or not opencv_running:  # Sai se pressionar
            "q" ou o programa for encerrado
            break

        #      Nova condi  o para encerrar o OpenCV
           corretamente:
        if cv2.getWindowProperty("Window", cv2.WND_PROP_VISIBLE)
           < 1:
            opencv_running = False  # Para o loop
            close_interactive_menu()
```

```
            break

    cv2.destroyAllWindows()

def toggle_fullscreen(root): # Fun  o para alternar entre tela
   cheia e janela
    is_fullscreen = root.attributes("-fullscreen")
    root.attributes("-fullscreen", not is_fullscreen)
    status_message.set("Tela␣cheia␣ativada" if not is_fullscreen
       else "Tela␣cheia␣desativada")

def toggle_video_mode(): # Fun  o para alternar entre modo de
   v deo e imagem
    global videoMode
    videoMode *= -1
    status_message.set("Modo␣de␣v deo␣alternado")

def show_help():
    help_text = (
        "Comandos␣Dispon veis:\n\n"
        "Deletar␣Linha:␣Remove␣a␣ ltima ␣linha␣desenhada\n"
        "Salvar␣Imagem:␣Salva␣a␣imagem␣atual␣no␣sistema\n"
        "Alternar␣Modo␣de␣V deo:␣Ativa/Desativa␣o␣modo␣de␣v deo
           \n"
        "Carregar␣Imagem:␣Seleciona␣uma␣imagem␣do␣arquivo\n"
        "Capturar␣Imagem␣URL:␣Captura␣imagem␣de␣uma␣URL␣ao␣vivo\n
           "
    )
    messagebox.showinfo("Ajuda␣-␣Comandos", help_text)

def exit_program(root):
    global opencv_running, interactive_menu

    print("    ␣Encerrando␣o␣programa...")

    # Parar o loop do OpenCV
    opencv_running = False

    # Fechar todas as janelas do OpenCV corretamente
    if cv2.getWindowProperty("Window", cv2.WND_PROP_VISIBLE) >=
       0:
        print("    ␣Fechando␣OpenCV...")
```

```
            cv2.destroyAllWindows()
            cv2.waitKey(1)  # Pequeno delay para garantir que fechou

        # Fechar o menu interativo, se ainda estiver aberto
        if interactive_menu is not None and isinstance(
           interactive_menu, Toplevel):
            print("    Fechando menu interativo...")
            try:
                interactive_menu.destroy()
            except Exception as e:
                print(f"      Erro ao fechar menu interativo: {e}")
            interactive_menu = None

        # For ar o Tkinter a sair imediatamente
        print("    For ando atualiza  o do Tkinter...")
        try:
            root.quit()  # Sai do mainloop() imediatamente
            root.update_idletasks()
            root.update()
        except Exception as e:
            print(f"      Erro ao atualizar root: {e}")

        # Garantir que o `mainloop()` foi realmente encerrado
        print("    Finalizando Tkinter...")
        try:
            root.destroy()
        except Exception as e:
            print(f"      Erro ao destruir root: {e}")

        # FOR AR encerramento se ainda estiver travado
        print("    For ando encerramento total...")
        time.sleep(0.2)  # Pequeno delay final para garantir
           fechamento
        os._exit(0)



def main_menu():
    global status_message
    root = Tk()
    root.title("Menu Interativo - Processamento de Imagem")
```

```
root.geometry("375x500")  # Define um tamanho fixo adequado
    para monitores modernos

status_message = StringVar()
status_message.set("Pronto")

menu_bar = Menu(root)
help_menu = Menu(menu_bar, tearoff=0)
help_menu.add_command(label="Comandos␣Dispon veis", command=
    show_help)
help_menu.add_command(label="Tela␣Cheia", command=lambda:
    toggle_fullscreen(root))
menu_bar.add_cascade(label="Ajuda", menu=help_menu)
root.config(menu=menu_bar)

frame = Frame(root, bg="#e3f2fd")
frame.pack(fill="both", expand=True, padx=20, pady=20)

Button(frame, text="    ␣Carregar␣Imagem␣de␣Arquivo", command
    =lambda: captureImage(1), height=2, width=40, bg="#bbdefb",
     fg="#0d47a1", activebackground="#90caf9", activeforeground
    ="#0d47a1", font=("Arial", 12, "bold")).pack(pady=10)
Button(frame, text="    ␣Capturar␣Imagem␣via␣URL", command=
    lambda: captureImage(0), height=2, width=40, bg="#c8e6c9",
    fg="#1b5e20", activebackground="#a5d6a7", activeforeground=
    "#1b5e20", font=("Arial", 12, "bold")).pack(pady=10)
Button(frame, text="    ␣Alternar␣Modo␣de␣V deo", command=
    toggle_video_mode, height=2, width=40, bg="#ffe0b2", fg="#
    e65100", activebackground="#ffcc80", activeforeground="#
    e65100", font=("Arial", 12, "bold")).pack(pady=10)
Button(frame, text="  ␣Sair", command=lambda: threading.
    Thread(target=exit_program, args=(root,), daemon= True).
    start(), height=2, width=40, bg="#ffccbc", fg="#b71c1c",
    activebackground="#ffab91", activeforeground="#b71c1c",
    font=("Arial", 12, "bold")).pack(pady=10)

status_label = Label(root, textvariable=status_message, bg="#
    e3f2fd", fg="#0d47a1", font=("Arial", 10))
status_label.pack(side="bottom", fill="x", pady=10)

root.mainloop()
```

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


'''
def subImagens(imagem):
    imagens = {
        1: [355, 65, 555, 220],
        2: [365, 220, 555, 380],
        3: [365, 380, 555, 545],
        4: [365, 545, 555, 715],
        5: [365, 715, 555, 895],
        6: [365, 875, 555, 1055],
        7: [555, 220, 770, 380],
        8: [555, 220, 770, 380],
        9: [555, 380, 770, 545],
        10: [555, 545, 770, 715],
        11: [555, 715, 770, 895],
        12: [555, 875, 770, 1055],
        13: [770, 380, 965, 545],
        14: [770, 220, 965, 380],
        15: [770, 380, 965, 545],
        16: [770, 545, 965, 715],
        17: [770, 715, 965, 895],
        18: [770, 875, 965, 1055],
        19: [965, 545, 1150, 715],
        20: [965, 220, 1150, 380],
        21: [965, 380, 1150, 545],
        22: [965, 545, 1150, 715],
        23: [965, 715, 1150, 895],
        24: [965, 875, 1150, 1055]};
    i=1;
    while True:
        recorte = imagens[i];
        (xi, yi, xf, yf) = recorte;
        print(f'{xi}, {xf}, {yi}, {yf}')
        subImagem = imagem[xi:xf, yi:yf]
        resultado = Processamento.process(subImagem, 4)
        print(resultado)
        mensagem = f'Recorte {i}'
        cv2.imshow(mensagem, subImagem)
        k = cv2.waitKey(0)
```

```
        if k % 256 == 27:
            # ESC pressed
            print("Escape hit, closing...")
            break
        elif k % 256 == 32:
            i+=1;

def click_event(event, x, y, flags, param):
    global img
    if event == cv2.EVENT_LBUTTONDOWN:
        print(x,y)
        cv2.circle(img, (x, y), 10, (0, 0, 255), -1)
        cv2.imshow('image', img)
    if event == cv2.EVENT_RBUTTONDBLCLK:
        img = cv2.imread(img_path)
        print("cleaned")
        cv2.imshow('image', img)

def drawImage():
    while True:
        cv2.setMouseCallback('Image', click_event)
        cv2.imshow('IPWebcam', img)
        k=cv2.waitKey(1)
        if k%256 == 27:
            # ESC pressed
            print("Escape hit, closing...")
            return k



if sys.version_info[0] == 3:
    from urllib.request import urlopen
else:
    from urllib.request import urlopen
img_path = 'ImagemTeste.jpg'
global img
img = cv2.imread(img_path)
img = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)
cv2.namedWindow("image")
cv2.setMouseCallback('image', click_event)
```

```
while True:
    ##with urlopen('http://10.14.30.138:8080/shot.jpg') as url:
    ##    imgResp = url.read()

        # Numpy to convert into a array
    ##imgNp = np.array(bytearray(imgResp), dtype=np.uint8)

    # Finally decode the array to OpenCV usable format ;)
    ##img = cv2.imdecode(imgNp, -1)

    # put the image on screen

    cv2.imshow('image', img)
    k = cv2.waitKey(1)

    if k%256 == 27:
        # ESC pressed
        print("Escape hit, closing...")
        break
    elif k%256 == 32:
        cv2.imwrite("ImagemTeste.jpg", img)
        subImagens(img)

    # To give the processor some less stress
    # time.sleep(0.1)
cv2.destroyAllWindows()


cam = cv2.VideoCapture('http://10.14.30.138:4747/mjpegfeed')

cam.set(cv2.CAP_PROP_FRAME_WIDTH, 800)
cam.set(cv2.CAP_PROP_FRAME_HEIGHT, 600)

cv2.namedWindow("test")

img_counter = 0

while True:
    ret, frame = cam.read()
    if not ret:
```

```
        print("failed to grab frame")
        break
    frame = cv2.rotate(frame, cv2.ROTATE_90_CLOCKWISE)
    cv2.imshow("test", frame)

    k = cv2.waitKey(1)
    if k%256 == 27:
        # ESC pressed
        print("Escape hit, closing...")
        break
    elif k%256 == 32:
        # SPACE pressed
        img = frame
        #img = cv2.rotate(img, cv2.ROTATE_90_)
        height, width = img.shape[:2]
        colors = [(255, 0, 0)]

        cv2.imwrite("ImagemTeste.jpg", img)


        cv2.imshow("Quadro", img)
        cv2.waitKey(1)


cam.release()

cv2.destroyAllWindows()
'''
'''
Trecho de c digo de tratamento da rede yolo
        Class_names = []
        with open("coco.names", "r") as f:
            Class_names = [cname.strip() for cname in f.readlines
               ()]

        net1 = cv2.dnn.readNet("yolov4-tiny.weights", "yolov4-
           tiny.cfg")
        net2 = cv2.dnn.readNet("yolov4.weights", "yolov4.cfg")

        model1 = cv2.dnn_DetectionModel(net1)
        model1.setInputParams(size=(416, 416), scale=1 / 255)
```

```
model2 = cv2.dnn_DetectionModel(net2)
model2.setInputParams(size=(416, 416), scale=1 / 255)

print(height, " - ", width);
linhas = [0, 270, 565, height]
colunas = [0, 300, 544, width]

# for i in range(3):
#     for j in range(3):
#         newImg = img[linhas[i]:linhas[i+1],colunas[j]:
   colunas[j+1]]
newImg = img
classes, scores, boxes = model1.detect(newImg, 0.000001,
   0.000002)

box = zip(boxes);
print("Caixas ==> ", boxes)

for (classid, score, box) in zip(classes, scores, boxes):
    # box[0]+=colunas[j]
    # box[1]+=linhas[i]
    (largura, altura) = box[2:4]
    # if (altura < 600)  and (classid[0]==32):
    print("Altura x Largura  = (%d x %d) Razao de Aspecto
        = %.4f" % (altura, largura, altura / largura));
    color = colors[int(classid) % len(colors)]
    # label = f"{Class_names[classid[0]]} : {score}"
    label = f"{score}"
    cv2.rectangle(img, box, color, 2)
    cv2.putText(img, label, (box[0], box[1] - 10), cv2.
       FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

# for i in range(3):
#     for j in range(3):
#         newImg = img[linhas[i]:linhas[i+1],colunas[j]:
   colunas[j+1]]

newImg = img

classes, scores, boxes = model2.detect(newImg, 0.000001,
   0.000002)
```

```
        for (classid, score, box) in zip(classes, scores, boxes):
            # box[0]+=colunas[j]
            # box[1]+=linhas[i]
            (largura, altura) = box[2:4]
            # if (altura < 600)  and (classid[0]==32):
            print("Altura x Largura  = (%d x %d) Razao de Aspecto
                = %.4f" % (altura, largura, altura / largura));
            color = colors[int(classid) % len(colors)]
            # label = f"{Class_names[classid[0]]} : {score}"
            label = f"{score}"
            cv2.rectangle(img, box, color, 2)
            cv2.putText(img, label, (box[0], box[1] - 10), cv2.
               FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

'''

# #path = dlg.askopenfilename(title = "Selecione o ovo a ser
   avaliado",
# #                           filetypes=(('jpg files', '*.jpg'),
   ('jpeg files', '*.jpeg')))
#
# #img=cv2.imread(path)
# #colors = [(0,255,255),(255,255,0),(0,255,0),(255,0,0)]
# colors = [(255,0,0)]
#
# folder_selected = dlg.askdirectory() + "/";
# files = os.scandir(folder_selected)
#
#
# Class_names = []
# with open("coco.names", "r") as f:
#     Class_names = [cname.strip() for cname in f.readlines()]
#
# #cv2.imshow("Original", imagem)
#
# #cap=cv2.VideoCapture(0, cv2.CAP_DSHOW)
# #cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
# #cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
#
# #net = cv2.dnn.readNet("yolov4-tiny.weights", "yolov4-tiny.cfg
   ")
# net = cv2.dnn.readNet("yolov4.weights", "yolov4.cfg")
```

```
#
# model = cv2.dnn_DetectionModel(net)
# model.setInputParams(size=(416,416), scale = 1/255)
#
# for T in files:
#     if T.name.endswith(('.jpg','.jpeg')):
#         imgFile = folder_selected + T.name
#         img = cv2.imread(imgFile)
#         #contador = 1;
#
#         classes, scores, boxes = model.detect(img,
   0.000001,0.000002)
#
#         for (classid, score, box) in zip(classes,scores,boxes):
#             (largura,altura) = box[2:4]
#             #if (altura < 600)  and (classid[0]==32):
#             print("Altura x Largura  = (%d x %d) Razao de
   Aspecto = %.4f" % (altura, largura, altura / largura));
#             color = colors[int(classid)%len(colors)]
#             label = f"{Class_names[classid[0]]} : {score}"
#             cv2.rectangle(img,box,color,2)
#             cv2.putText(img,label,(box[0], box[1] - 10), cv2.
   FONT_HERSHEY_SIMPLEX,0.5, color, 2)
#
#         cv2.imshow("Quadro", img)
#         cv2.waitKey(0)
#
#                 #imgFile = folder_selected + 'Processed_' + f.
   name
#
#                 #cv2.imwrite(imgFile, img)
#
#
#
# #
# cv2.destroyAllWindows()
```

A.2 processamento.py (processamento com paralelismo)

```
from math import atan, degrees
from tkinter import filedialog as dlg
import cv2
```

```
import numpy as np
import pandas as pd
import os
import math
import multiprocessing
from openpyxl import Workbook  # pip install openpyxl
from openpyxl import load_workbook
from skimage.measure import label, regionprops_table
from skimage.transform import resize
from skimage.measure import find_contours
from skimage.color import rgb2hsv
from skimage.morphology import disk
from skimage.filters import median
from scipy.optimize import curve_fit  # Lib p/ fazer ajuste
   polinomial
import matplotlib.pyplot as plt
from pathlib import Path
from concurrent.futures import ProcessPoolExecutor
import time
from functools import wraps

def timing(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        start = time.perf_counter()
        result = func(*args, **kwargs)
        end = time.perf_counter()
        print(f"[TIMER] {func.__name__} levou {end-start:.3f}s")
        return result
    return wrapper

#
   ----------------------------------------------------------------------

# 1)  C P U only : roda em subprocessos, sem NENHUM I/O
#
   ----------------------------------------------------------------------

def cpu_process_egg(job):
    """
    Recebe um job com (egg_num, rec_bytes, shape, l0,l1,c0,c1,
       pixFactor),
```

```
    executa SOMENTE o processamento num rico e devolve tudo em
       mem ria.
    """
    egg_num, rec_bytes, shape, l0, l1, c0, c1, pixFactor = job
    rec = np.frombuffer(rec_bytes, dtype=np.uint8).reshape(shape)
    out = process(rec, 1, pixFactor, None, None, None)
    # agora out inclui tamb m pAi, pAf, pBi, pBf
    original, A, B, C, D, chosen_vol, chosen_area, x_data, y_data
       , pAi, pAf, pBi, pBf = out
    return egg_num, l0, l1, c0, c1, (original, A, B, C, D,
       chosen_vol, chosen_area, x_data, y_data, pAi, pAf, pBi, pBf
       )

def check_and_create_directory_if_not_exist(path_directory):
    if not os.path.exists(path_directory):
        os.makedirs(path_directory)
        print(f"The␣path␣{path_directory}␣is␣created!")

# Esta fun  o ajusta as dimens es de uma imagem para garantir
   que ela
# n o exceda 1800 pixels em largura ou altura, mantendo a
   propor  o.
# im > imagem de entrada
def adjustImageDimension(im):
    tLines, tColumns, c = im.shape
    if tLines > tColumns:
        rFactor = 1800 / tLines
        tLines = 1800
        tColumns = int(tColumns * rFactor)
    else:
        rFactor = 1800 / tColumns
        tColumns = 1800
        tLines = int(tLines * rFactor)
    return (tLines, tColumns, rFactor)

# Esta fun  o desenha linhas na imagem com base em coordenadas
   fornecidas para linhas
# de calibra  o, tempor rias e de elementos.
# variaveis: image -> imagem na qual as linhas ser o desenhadas
# calibrationLine -> coordenadas da linha de calibra  o
# temLines -> lista de coordenadas para linhas temp
# elementLines -> lista de coordenadas para linhas de elementos
```

```
# totalColumns -> numero total de colunas a srem desenhadas
# totalLines -> numero total de linhas a serem desenhadas
def drawLines(image, calibrationLine, tempLines, elementLines,
   totalColumns, totalLines):
    if len(calibrationLine) > 0:
        xi, yi, c = calibrationLine[0]
        xf, yf, c = calibrationLine[1]
        cv2.line(image, (xi, yi), (xf, yf), (0, 0, 255), 1)
    else:
        if len(tempLines) > 0:
            (posTemp, status) = tempLines[0]
            if status == 1:  # indica uma coluna
                cv2.line(image, (0, posTemp), (totalColumns - 1,
                   posTemp), (0, 255, 0), 1)
            elif status == 2:
                cv2.line(image, (posTemp, 0), (posTemp,
                   totalLines - 1), (255, 0, 0), 1)

        for i in elementLines:
            (posTemp, status) = i
            if status == 1:  # indica uma coluna
                cv2.line(image, (0, posTemp), (totalColumns - 1,
                   posTemp), (0, 255, 0), 1)
            elif status == 2:
                cv2.line(image, (posTemp, 0), (posTemp,
                   totalLines - 1), (255, 0, 0), 1)

    return (image)

# Essa fun  o extrai sub-imagens de uma imagem maior com base
# em linhas tempor rias e de elementos.
# variaveis: image -> imagem da qual as subimagens ser o
   extraidas
# temLines -> lista de coordenadas para linhas temp
# elementLines -> lista de coordenadas para linhas de elementos
@timing
def subImagens(img, tColumns, tLines, factor, pixFactor, dFactor,
    nomeArquivo):
    t0 = time.perf_counter()
    """
    1) monta a lista de jobs (imagem, coordenadas, pixFactor)
    2) paraleliza S  cpu_process_egg (sem I/O interno)
```

```
    3) fora do pool, faz TODO o I/O:
       - grava PNGs
       - escreve Relatorio.dad
       - gera gr ficos (disc_slices_curve_fit)
       - atualiza Excel (create_sheet)
       - comp e imagem final
    """

    imgProcess = img.copy()
    posEggs = findeggs(img)

    # inicializa limites para recorte final
    lMin, cMin = img.shape[0], img.shape[1]
    lMax, cMax = 0, 0

    # 2.1) montar jobs
    jobs = []
    for n, egg in enumerate(posEggs, 1):
        _, _, col, lin, larg, alt = egg
        l0 = int(lin - round(alt * 0.075))
        l1 = int(lin + round(alt * 1.075))
        c0 = int(col - round(larg * 0.075))
        c1 = int(col + round(larg * 1.075))
        rec = img[l0:l1, c0:c1].copy()
        jobs.append((n, rec.tobytes(), rec.shape, l0, l1, c0, c1,
            pixFactor))

    # 2.2) paralelizar S  processamento CPU
    maxw = max(1, multiprocessing.cpu_count() - 1)
    with ProcessPoolExecutor(max_workers=maxw) as exe:
        results = list(exe.map(cpu_process_egg, jobs))

    # 2.3) preparar diret rios de sa da
    root      = Path.cwd()/'results'/Path(nomeArquivo).stem
    processed = root/'processed'
    plots     = root/'fit_plots_results'
    for d in (root, processed, plots):
        check_and_create_directory_if_not_exist(d)
    rel = open(root/'Relatorio.dad', 'a')

    # 2.4) TODO o I/O ocorre AQUI, sequencialmente
    for egg_num, l0, l1, c0, c1, out in results:
```

```
img_proc, A, B, C, D, vol, area, x_data, y_data, pAi, pAf
    , pBi, pBf = out

# --- 2.4.1) PNG e .dad -----------------------
fn_png = processed / f'{egg_num}.png'
cv2.imwrite(str(fn_png), img_proc)

# vamos calcular vFormulaArea e vFormulaVolume:
try:
    termo1 = math.acos((B/2)/D) * (D**2/math.sqrt(D**2-(B
        /2)**2))
    termo2 = math.acos((B/2)/C) * (C**2/math.sqrt(C**2-(B
        /2)**2))
    vFormulaArea = 2*math.pi*(B/2)**2 + math.pi*(B/2)*(
        termo1+termo2)
except ValueError:
    vFormulaArea = 1
try:
    vFormulaVolume = (B/2)**2 * ((2*math.pi)/3) * (D + C)
except ValueError:
    vFormulaVolume = 1

rel.write(f"{fn_png},{A},{B},{C},{D},{vol},{area},{
    vFormulaArea},{vFormulaVolume}\n")

# --- 2.4.2) CRIA  S U B PASTA  por ovo ------------
egg_plots = plots / str(egg_num)
check_and_create_directory_if_not_exist(egg_plots)

# --- 2.4.3) ajuste polinomial e gr ficos -----
x_arr = np.array(x_data)
y_arr = np.array(y_data)
fits, errs = disc_slices_curve_fit(
    x_arr, y_arr,
    [polynomial_curv_3, polynomial_curv_5,
     polynomial_curv_7, polynomial_curv_9,
        polynomial_curv_11],
    nomeArquivo, egg_num, egg_plots
)

# 4.4) grava imagem rotacionada, usando os pontos
    p A i pBf
```

```
        rec = img[l0:l1, c0:c1]
        cSEx = int(min(pAi[0], pAf[0], pBi[0], pBf[0]))
        cSEy = int(min(pAi[1], pAf[1], pBi[1], pBf[1]))
        cIDx = int(max(pAi[0], pAf[0], pBi[0], pBf[0]))
        cIDy = int(max(pAi[1], pAf[1], pBi[1], pBf[1]))
        fn_rot = processed / f'rotate_{egg_num}.png'
        cv2.imwrite(str(fn_rot), rec[cSEx:cIDx, cSEy:cIDy])

        # 4.5) atualiza Excel
        # monta volume_results: 'Antigo' + cada ajuste
        old_slices = dict(zip(x_data, y_data))
        oldV, oldA = calcVolume(old_slices, pixFactor)
        volume_results = {'Antigo':(oldV, oldA)}
        for func, slice_d in fits.items():
            vv, aa = calcVolume(slice_d, pixFactor)
            volume_results[func] = (vv, aa)

        create_sheet(egg_plots, str(egg_num), volume_results,
           errs)

        # --- 2.4.6) cola no mosaico final ------------
        imgProcess[l0:l1, c0:c1] = img_proc
        lMin = min(lMin, l0)
        cMin = min(cMin, c0)
        lMax = max(lMax, l1)
        cMax = max(cMax, c1)

    rel.close()

    # recorta apenas a  rea  com ovos
    if lMax > lMin and cMax > cMin:
        rec = imgProcess[lMin:lMax, cMin:cMax]
    else:
        rec = imgProcess  # fallback, tudo
    tL, tC, _ = adjustImageDimension(rec)
    if tL and tC:
        disp = cv2.resize(rec, (tC, tL), interpolation=cv2.
           INTER_LINEAR)
    else:
        disp = rec

    # 2.7) **salva o mosaico redimensionado** (e  n  o  o
```

```
        imgProcess gigante)
    final_fn = root / 'imagem_processada_final.png'
    cv2.imwrite(str(final_fn), disp)
    print("Processamento conclu do em:", root)
    print(f"subImagens demorou: {time.perf_counter() - t0:.2f}s")

def process(frame, factor, pixFactor, egg_num, path_file_name,
    egg_folder_fit_plot_path):

    # Measures that will be returned
    A = B = C = D = 0

    # Uma c pia da imagem original    feita para preservar a
        imagem na resolu  o original.
    original = frame.copy()

    # Reduz a imagem para acelerar
    lin, col, ch = original.shape
    frame = resize(frame, [int(lin / factor), int(col / factor)])
    frame = np.uint8(frame * 255)

    # Segmenta  o HSV
    data = rgb2hsv(frame)
    channel1Min, channel1Max = 0.0, 1.0
    channel2Min, channel2Max = 0.0, 1.0
    channel3Min, channel3Max = 0.578, 1.0

    print('Creating the mask for segmentation')
    data = np.bitwise_and(
        np.bitwise_and(
            np.bitwise_and(data[:,:,0] >= channel1Min, data
                [:,:,0] <= channel1Max),
            np.bitwise_and(data[:,:,1] >= channel2Min, data
                [:,:,1] <= channel2Max)
        ),
        np.bitwise_and(data[:,:,2] >= channel3Min, data[:,:,2] <=
            channel3Max)
    )
    data[data > 0] = 1
    data = median(data, disk(3))
    data[data > 0] = 1
```

```
# Encontrar boundingbox do ovo
print('Process to identify the bounding box that contains the
   egg and subsequently improve segmentation')
linoriginal, coloriginal = data.shape
labels = label(data)
props = regionprops_table(labels, properties=('bbox','
   major_axis_length','minor_axis_length'))
df = pd.DataFrame(props)

fl_find_bb = False
for _, row in df.iterrows():
    if 0.30*linoriginal <= row['major_axis_length'] <=
       linoriginal and \
       0.15*coloriginal <= row['minor_axis_length'] <=
          coloriginal:
        data = data.astype('uint8')
        data[int(row['bbox-0']):int(row['bbox-2']),
             int(row['bbox-1']):int(row['bbox-3'])] += 1
        data[data != 2] = 0
        data[data == 2] = 255
        fl_find_bb = True
        break

if not fl_find_bb:
    return [original, -1, -1, -1, -1]

# Encontrar maior eixo (A) e eixo perpendicular (B)
print('Identifying the two points that form the longest
   straight line')
border_points = np.vstack(find_contours(data, 0.1))
max_distance = 0
pt1 = pt2 = []
for i in range(1, len(border_points)):
    for j in range(i+1, len(border_points)):
        d = np.linalg.norm(border_points[i]*factor -
           border_points[j]*factor)
        if d > max_distance:
            max_distance = d
            pt1, pt2 = border_points[i]*factor, border_points
               [j]*factor
if len(pt1) and len(pt2):
    original = cv2.line(original,
```

```
                        (int(pt1[1]),int(pt1[0])),
                        (int(pt2[1]),int(pt2[0])),
                        (47,141,255),2)
    A = max_distance

# Cálculo do segundo eixo (B) e coleta de pontos para ajuste
    de curva
print('Finds the angle of the line formed by the previous
   points and, later, finds the longest straight line')
msRmaior = (pt1[1]*factor - pt2[1]*factor) / (pt1[0]*factor -
    pt2[0]*factor)
ms = -1/msRmaior
ms = degrees(atan(ms))
max_distance = 0
pt3 = pt4 = []
dicSlicesNoFit = {}
xdata = []
ydata = []

for i in range(1, len(border_points)):
    for j in range(1, len(border_points)):
        if border_points[j][0] != border_points[i][0]:
            mdRmenor = ((border_points[i][1]*factor -
               border_points[j][1]*factor) /
                        (border_points[i][0]*factor -
                           border_points[j][0]*factor))
            bRmenor = border_points[i][1] - mdRmenor*
               border_points[i][0]
            md = degrees(atan(mdRmenor))
            if 0 <= abs(md - ms) <= 0.3:
                d = np.linalg.norm(border_points[i]*factor -
                   border_points[j]*factor)
                distP1 = abs(-mdRmenor*pt1[0] + pt1[1] -
                   bRmenor) / ((mdRmenor**2+1)**0.5)
                original = cv2.line(original,
                                    (int(border_points[i][1])
                                       ,int(border_points[i
                                       ][0])),
                                    (int(border_points[j][1])
                                       ,int(border_points[j
                                       ][0])),
                                    (30,105,210),1)
```

```
                dicSlicesNoFit[distP1] = d
                xdata.append(distP1)
                ydata.append(d)
                if d > max_distance:
                    max_distance = d
                    pt3, pt4 = border_points[i]*factor,
                       border_points[j]*factor

if len(pt3) and len(pt4):
    original = cv2.line(original,
                        (int(pt3[1]),int(pt3[0])),
                        (int(pt4[1]),int(pt4[0])),
                        (0,102,0),2)
    B = max_distance

# Linha de interse  o para C e  D
inter = lineLineIntersection(pt1,pt2,pt3,pt4)
if inter:
    d1 = np.linalg.norm(pt1 - inter)
    d2 = np.linalg.norm(pt2 - inter)
    if d1 > d2:
        C, D = d1, d2
        original = cv2.line(original,(int(pt1[1]),int(pt1[0])
           ),(int(inter[1]),int(inter[0])),(0,0,255),2)
        original = cv2.line(original,(int(pt2[1]),int(pt2[0])
           ),(int(inter[1]),int(inter[0])),(255,255,255),2)
    else:
        C, D = d2, d1
        original = cv2.line(original,(int(pt2[1]),int(pt2[0])
           ),(int(inter[1]),int(inter[0])),(0,0,255),2)
        original = cv2.line(original,(int(pt1[1]),int(pt1[0])
           ),(int(inter[1]),int(inter[0])),(255,255,255),2)

# ========================
# REMOVIDO TODO ESTE BLOCO DE C LCULO DE CURVA E I/O:
#
# dic_slices_fit, curve_fit_errors = \
#     disc_slices_curve_fit(xdata, ydata,
   poly_degree_functions,
#                           path_file_name, egg_num,
   egg_folder_fit_plot_path)
# volume_results = {}
```

```
    # (oldVolume, oldEggArea) = calcVolume(dicSlicesNoFit,
       pixFactor)
    # volume_results["Antigo"] = (oldVolume, oldEggArea)
    # for poly_function in dic_slices_fit:
    #     (poly_volume, poly_area) = calcVolume(dic_slices_fit[
       poly_function], pixFactor)
    #     volume_results[poly_function] = (poly_volume, poly_area
       )
    #     if poly_function == polynomial_curv_11:
    #         chosen_volume = poly_volume
    #         chosen_area = poly_area
    # create_sheet(egg_folder_fit_plot_path, egg_num,
       volume_results, curve_fit_errors)
    # ======================

    chosen_volume = 0.0
    chosen_area   = 0.0
    pAi, pAf, pBi, pBf = pt1, pt2, pt3, pt4
    # Desenha valores A, B, C, D e V
    A *= pixFactor; B *= pixFactor; C *= pixFactor; D *=
       pixFactor
    cv2.putText(original, f'A: {A:.3f}', (original.shape[1]-150,
        30), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (47,141,255), 1)
    cv2.putText(original, f'B: {B:.3f}', (original.shape[1]-150,
        50), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0,102,  0), 1)
    cv2.putText(original, f'C: {C:.3f}', (original.shape[1]-150,
        70), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0,  0,255), 1)
    cv2.putText(original, f'D: {D:.3f}', (original.shape[1]-150,
        90), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (255,255,255), 1)
    cv2.putText(original, f'V: {chosen_volume:.3f}', (original.
       shape[1]-150,110), cv2.FONT_HERSHEY_SIMPLEX, 0.75,
       (30,105,210),1)

    # retorna a tupla completa de dados e os vetores xdata, ydata
        para o ajuste ficar fora daqui
    return original, A, B, C, D, chosen_volume, chosen_area,
       xdata, ydata, pAi, pAf, pBi, pBf

# Function to return the intersection between two lines
def lineLineIntersection(A, B, C, D):
    # Line 01
    a1 = B[1] - A[1]
```

```
    b1 = A[0] - B[0]
    c1 = a1 * (A[0]) + b1 * (A[1])

    # Line 02
    a2 = D[1] - C[1]
    b2 = C[0] - D[0]
    c2 = a2 * (C[0]) + b2 * (C[1])

    determinant = a1 * b2 - a2 * b1

    if determinant == 0:
        return False
    else:
        x = (b2 * c1 - b1 * c2) / determinant
        y = (a1 * c2 - a2 * c1) / determinant
        return [int(x), int(y)]


def calcVolume(slices, pixFactor):
    dicSlices = sorted(slices.items())
    volume = 0
    areaLateral = 0
    # print('Inicio Ovo')
    for t in range(1, len(dicSlices)):
        elem1 = dicSlices[t]
        elem0 = dicSlices[t - 1]
        h2, rMaior = elem1
        h1, rMenor = elem0

        # print ("(",h2,",",rMaior,")")

        h = abs(h2 - h1) * pixFactor
        rMaior = (rMaior / 2) * pixFactor
        rMenor = (rMenor / 2) * pixFactor
        g = math.sqrt(h ** 2 + (rMaior - rMenor) ** 2)
        volume += ((math.pi * h) / 3) * (rMaior ** 2 + rMenor **
           2 + rMaior * rMenor)
        areaLateral += math.pi * g * (rMaior + rMenor)
    volume = volume / 100
    return [volume, areaLateral]
```

```
def disc_slices_curve_fit(x_data, y_data,
   polynomial_fit_degree_functions, path_file_name, egg_num,
   egg_folder_fit_plot_path):
    # last_function = polynomial_fit_degree_functions[-1]
    resultDiscSlices = {}
    # resultCurveError = []
    resultCurveError = {}
    for poly_fit_function in polynomial_fit_degree_functions:
        # Criar a curva de ajuste polinomial do grau escolhido na
            lista
        popt, pcov = curve_fit(poly_fit_function, x_data, y_data)

        # Using 'lm' method for Levenberg-Marquardt algorithm or
           'trf' method for Trust Region Reflective algorithm.
        # p0 = [10, 0.1, 1, 10, 0.1, 1, 1]
        # popt, pcov = curve_fit(poly_fit_function, x_data,
           y_data, p0=p0, method='lm')

        y_data_fit = poly_fit_function(x_data, *popt)

        # plotando o gr fico do ajuste com os os valores de
           coeficiente otimizados
        plt.cla()
        plt.plot(x_data, y_data, 'b-', label='Sem␣ajuste␣(antigo)
           ')
        plt.plot(x_data, y_data_fit, 'r-', label=f'Com␣ajuste:␣{
           poly_fit_function.__name__}')
        plt.suptitle(f'{Path(path_file_name).name}␣-␣Ovo␣{egg_num
           }', fontweight='bold')
        plt.title(f'Compara  o:␣Sem␣ajuste␣X␣C/␣ajuste␣({
           poly_fit_function.__name__})')
        plt.xlabel("Dist ncia␣X")
        plt.ylabel("Pontos␣de␣dist ncia␣Y")
        plt.legend()
        # plt.show()

        # plt.savefig(rf'{__plot_results_path}/{__root_file_name}
           _{poly_fit_function.__name__}')
        plt.savefig(Path(egg_folder_fit_plot_path, f'plot_{
           poly_fit_function.__name__}'))

        # erro do ajuste polinomial
```

```
            perr = np.sqrt(np.diag(pcov))
            print(f'Error␣of␣the␣curve␣fit␣-␣{poly_fit_function.
                __name__}:␣{perr}\n')

            # Convert lists (x_data, y_data_fit) to dictionary and
                append it to the result list as tuple
            resultDiscSlices[poly_fit_function] = dict(zip(x_data,
                y_data_fit))
            # resultCurveError.append((poly_fit_function, perr))
            resultCurveError[poly_fit_function] = perr

            # if(poly_fit_function == last_function):
            # using dict() and zip() to convert lists to dictionary
            # return dict(zip(x_data, y_data_fit))
        # print(f'FINAL RESULT : {resultDiscSlices}')
        # return [np.array(resultDiscSlices), resultCurveError]
        return [resultDiscSlices, resultCurveError]

# fun  o detecta ovos em uma imagem com base em certos
    crit rios de  rea  e propor  o.
# Ela retorna uma lista de ovos encontrados, onde cada ovo
    representado por uma lista [grupo, linha, x, y, largura, altura
    ]
def findeggs(originalImg):
    ovos = []
    (alt, larg, ch) = originalImg.shape
    AreaTotal = alt * larg
    # preprocess the image
    gray_img = cv2.cvtColor(originalImg, cv2.COLOR_BGR2GRAY)
    # Applying 7x7 Gaussian Blur
    blurred = cv2.GaussianBlur(gray_img, (7, 7), 0)
    # Applying threshold
    threshold = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY
        | cv2.THRESH_OTSU)[1]
    # Apply the Component analysis function
    analysis = cv2.connectedComponentsWithStats(threshold, 4, cv2
        .CV_32S)
    (totalLabels, label_ids, values, centroid) = analysis
    # Loop through each component
    for i in range(1, totalLabels):
        # Area of the component
        area = values[i, cv2.CC_STAT_AREA]
```

```
        percArea = (area * 100) / AreaTotal
        aspectRatio = float(int(values[i, cv2.CC_STAT_HEIGHT]) /
           int(values[i, cv2.CC_STAT_WIDTH]))

        if (percArea > 0.4) and (percArea < 1) and (aspectRatio >
            1.1) and (aspectRatio < 1.6):
            (col, lin) = centroid[i]
            x = values[i, cv2.CC_STAT_LEFT]
            y = values[i, cv2.CC_STAT_TOP]
            w = values[i, cv2.CC_STAT_WIDTH]
            h = values[i, cv2.CC_STAT_HEIGHT]
            elem = [0, lin, x, y, w, h]
            ovos.append(elem)
    posVet = 0
    grupo = 1
    # categoriza pela posi  o na linha, s o classificados
       aquelas regi es cuja posi  o n o linha variam abaixo de
        10%
    for i in range(len(ovos)):
        if ovos[i][0] == 0:
            ovos[i][0] = grupo
            for k in range(i + 1, len(ovos)):
                if (abs(ovos[k][1] - ovos[i][1]) * 100) / ovos[i
                   ][1] < 10:
                    ovos[k][0] = grupo
            grupo += 1

    # ordena pela linha
    for i in range(0, len(ovos) - 1):
        for j in range(i + 1, len(ovos)):
            if ovos[j][0] < ovos[i][0]:
                troca = ovos[j]
                ovos[j] = ovos[i]
                ovos[i] = troca

    # ordena pela coluna
    # controle = 1
    for i in range(0, len(ovos) - 1):
        for j in range(i+1, len(ovos)):
            if ((ovos[j][0] == ovos[i][0]) and (ovos[j][2] < ovos
               [i][2])):
                troca = ovos[j]
```

```
                ovos[j] = ovos[i]
                ovos[i] = troca
    return ovos

"""
Representa  o da curva - Curva polinomial de grau 3
A fun  o deve receber as coordenadas X e Y de seus pontos de
   dados como entradas e retornar os valores Y previstos para cada
    valor X.
"""


def polynomial_curv_3(x, a, b, c, d):
    return a * x ** 3 + b * x ** 2 + c * x + d


def polynomial_curv_5(x, a, b, c, d, e, f):
    return a * x ** 5 + b * x ** 4 + c * x ** 3 + d * x ** 2 + e
       * x + f


def polynomial_curv_7(x, a, b, c, d, e, f, g, h):
    return a * x ** 7 + b * x ** 6 + c * x ** 5 + d * x ** 4 + e
       * x ** 3 + f * x ** 2 + g * x + h


def polynomial_curv_9(x, a, b, c, d, e, f, g, h, i, j):
    return a * x ** 9 + b * x ** 8 + c * x ** 7 + d * x ** 6 + e
       * x ** 5 + f * x ** 4 + g * x ** 3 + h * x ** 2 + i * x + j


def polynomial_curv_11(x, a, b, c, d, e, f, g, h, i, j, k, m):
    return a * x ** 11 + b * x ** 10 + c * x ** 9 + d * x ** 8 +
       e * x ** 7 + f * x ** 6 + g * x ** 5 + h * x ** 4 + i * x
       ** 3 + j * x ** 2 + k * x + m


def piecewise_linear(x, x0, y0, k1, k2):
    return np.piecewise(x, [x <= x0], [lambda x: k1 * x + y0 - k1
        * x0, lambda x: k2 * x + y0 - k2 * x0])
```

```
# def sigmoid(x, a, b, c, d):
# y = a / (1 + np.exp(-b*(x-c))) + d
# return y

# N o funcionou : Linha indo para a direita, sem seguir os
   pontos
def sigmoid(x, a, b, c, d):
    z = b * (x - c)
    z = np.clip(z, -500, 500)  # limit the values of the
       exponential term
    return a / (1 + np.exp(-z)) + d


# Resultado: Semelhante a regress o polinomial de grau 3
def gaussian(x, a, b, c, d):
    return a * np.exp(-(x - b) ** 2 / (2 * c ** 2)) + d


def get_polynomial_curv_portuguese_name(polynomial_func):
    if polynomial_func == polynomial_curv_3:
        return "Ajuste␣Polinomial␣Grau␣3"
    elif polynomial_func == polynomial_curv_5:
        return "Ajuste␣Polinomial␣Grau␣5"
    elif polynomial_func == polynomial_curv_7:
        return "Ajuste␣Polinomial␣Grau␣7"
    elif polynomial_func == polynomial_curv_9:
        return "Ajuste␣Polinomial␣Grau␣9"
    elif polynomial_func == polynomial_curv_11:
        return "Ajuste␣Polinomial␣Grau␣11"
    else:
        return "Ajuste␣desconhecido"


# This function consists of the sum of three exponential
   functions with different amplitudes,
# centers, and widths, plus a constant d. You can adjust the
   initial parameter values to get a better fit.
# def exponential_combo(x, a1, b1, c1, a2, b2, c2, a3, b3, c3, d)
   :
# return a1 * np.exp(-((x - b1)/c1)**2) + a2 * np.exp(-((x - b2)/
   c2)**2) + a3 * np.exp(-((x - b3)/c3)**2) + d
# Esta fun  o cria ou atualiza uma planilha Excel com os
```

```
    resultados do volume e os erros
# de ajuste para diferentes tipos de ajuste polinomial
def create_sheet(path_file_name, egg_name, volume_results,
    curve_fit_errors):
     ordem_valores = ["Antigo", polynomial_curv_3,
        polynomial_curv_5, polynomial_curv_7, polynomial_curv_9,
                     polynomial_curv_11]

     if isinstance(path_file_name, Path):
         sheet_name_directory = Path(path_file_name.parents[2], '
            Comparacao_Volume_Area.xlsx')
         file_name = path_file_name.parents[1].stem

         if os.path.exists(sheet_name_directory):
             try:
                 workbook = load_workbook(sheet_name_directory)
             except Exception as e:
                 print(f"Arquivo ’{sheet_name_directory}’ 
                    corrompido ou inv lido. Criando novo workbook.
                    Erro: {e}")
                 workbook = Workbook()
         else:
             workbook = Workbook()

         sheetnames = workbook.sheetnames
         if file_name not in sheetnames:
             workbook.create_sheet(file_name)
             file_worksheet = workbook[file_name]
             file_worksheet["B1"] = "Volume - ANTIGO"
             file_worksheet["C1"] = "Area - ANTIGO"
             file_worksheet["D1"] = "Volume - POLINOM. GRAU 3"
             file_worksheet["E1"] = "Area - POLINOM. GRAU 3"
             file_worksheet["F1"] = "Erros - POLINOM. GRAU 3"
             file_worksheet["G1"] = "Volume - POLINOM. GRAU 5"
             file_worksheet["H1"] = "Area - POLINOM. GRAU 5"
             file_worksheet["I1"] = "Erros - POLINOM. GRAU 5"
             file_worksheet["J1"] = "Volume - POLINOM. GRAU 7"
             file_worksheet["K1"] = "Area - POLINOM. GRAU 7"
             file_worksheet["L1"] = "Erros - POLINOM. GRAU 7"
             file_worksheet["M1"] = "Volume - POLINOM. GRAU 9"
             file_worksheet["N1"] = "Area - POLINOM. GRAU 9"
             file_worksheet["O1"] = "Erros - POLINOM. GRAU 9"
```

```
            file_worksheet["P1"] = "Volume - POLINOM. GRAU 11"
            file_worksheet["Q1"] = "Area - POLINOM. GRAU 11"
            file_worksheet["R1"] = "Erros - POLINOM. GRAU 11"
        else:
            file_worksheet = workbook[file_name]

        # Find / Create egg row
        # is_new_egg = False
        egg_row = 0
        for row in file_worksheet.iter_rows(min_col=1, max_col=1)
           :
            for cell in row:
                if cell.value == egg_name:
                    egg_row = cell.row
        if egg_row == 0:
            max_col_row = len([cell for cell in file_worksheet["A
               "] if cell.value])
            egg_row = max_col_row + 2
            file_worksheet[f"A{egg_row}"] = egg_name
            # is_new_egg = True

        current_min_col = 2
        for value_type in ordem_valores:
            (volume, area) = volume_results[value_type]
            file_worksheet.cell(row=egg_row, column=
               current_min_col).value = volume
            file_worksheet.cell(row=egg_row, column=
               current_min_col + 1).value = area
            if value_type == "Antigo":
                current_min_col = current_min_col + 2
            else:
                file_worksheet.cell(row=egg_row, column=
                   current_min_col + 2).value = ', '.join(
                    str(error) for error in curve_fit_errors[
                       value_type])
                current_min_col = current_min_col + 3

        workbook.save(sheet_name_directory)
    else:
        print(f"input should be of type Path (pathlib) : {
           path_file_name}")
```

```
def exponential_combo(x, a1, b1, c1, a2, b2, c2, d):
    y1 = a1 * np.exp(-b1 * x) + c1
    y2 = a2 * np.exp(-b2 * (x - c2) ** 2) + d
    return y1 + y2
```

A.3 mouseController.py

```
import cv2


# function which will be called on mouse input
def drawLines(action, x, y, flags, *userdata):
    # Referencing global variables
    global elementLines, tempLines, originalImage, clicked,
       totalLines, totalColumns
    # Mark the top left corner when left mouse button is pressed

    if action == cv2.EVENT_LBUTTONDOWN:
        clicked = 1
        tempLines = [(y, clicked)]
        # When left mouse button is released, mark bottom right
           corner
    elif action == cv2.EVENT_LBUTTONUP:
        elementLines.append((y, clicked))
        clicked = 0
        # Draw the rectangle
    elif action == cv2.EVENT_RBUTTONDOWN:
        clicked = 2
        tempLines = [(x, clicked)]
    elif action == cv2.EVENT_RBUTTONUP:
        elementLines.append((x, clicked))
        clicked = 0
    elif action == cv2.EVENT_MOUSEMOVE:
        if clicked != 0:
            tempLines = []
            if clicked == 1:
                tempLines = [(y, clicked)]
            if clicked == 2:
                tempLines = [(x, clicked)]

    image = originalImage.copy()
```

```
if clicked != 0:
    (posTemp, status) = tempLines[0]
    if status == 1:  # indica uma coluna
        cv2.rectangle(image, (0, posTemp), (totalColumns-1,
           posTemp), (0, 255, 0), 2)
    elif status == 2:
        cv2.rectangle(image, (posTemp, 0), (posTemp,
           totalLines-1), (255, 0, 0), 2)

for i in elementLines:
    (posTemp, status) = i
    if status == 1:  # indica uma coluna
        cv2.rectangle(image, (0, posTemp), (totalColumns - 1,
            posTemp), (0, 255, 0), 2)
    elif status == 2:
        cv2.rectangle(image, (posTemp, 0), (posTemp,
           totalLines - 1), (255, 0, 0), 2)

cv2.imshow("Window", image)
```