

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Guilherme Nascimento Leite

**Implementação de algoritmo de processamento
digital de imagens para segmentação de ovos
de galinha**

Uberlândia, Brasil

2024

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Guilherme Nascimento Leite

**Implementação de algoritmo de processamento digital de
imagens para segmentação de ovos de galinha**

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, como parte dos requi-
sitos exigidos para a obtenção título de Ba-
charel em Ciência da Computação.

Orientador: Mauricio Cunha Escarpinati

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2024

Guilherme Nascimento Leite

Implementação de algoritmo de processamento digital de imagens para segmentação de ovos de galinha

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 Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 16 de maio de 2024:

Mauricio Cunha Escarpinati
Orientador

Professor

Professor

Uberlândia, Brasil
2024

Resumo

Este trabalho apresenta o desenvolvimento de um algoritmo para segmentação automática de ovos em imagens utilizando técnicas de processamento digital de imagens. O objetivo é desenvolver um método eficiente e preciso para identificar e separar os ovos em uma série de imagens que contêm ovos dispostos em linhas e colunas. O método desenvolvido passa por várias etapas, incluindo pré-processamento da imagem, detecção de bordas, identificação dos ovos, segmentação e análise dos resultados. Com o auxílio das bibliotecas OpenCV, NumPy e Matplotlib, foi possível implementar o algoritmo de forma eficaz. Os resultados obtidos ao final demonstraram uma acurácia de 100% na identificação e ordenação dos ovos para todos os casos teste. No entanto, é importante ressaltar que ainda há espaço para melhorias, como a integração de técnicas de aprendizado de máquina para aumentar a precisão da segmentação. O desenvolvimento deste trabalho contribui como um complemento para um projeto de pesquisa desenvolvido no Laboratório de Visão Computacional e Recuperação de Dados da UFU, de forma que seja viável automatizar a etapa de segmentação os ovos das imagens, que antes era realizada de forma semi-automática.

Palavras-chave: segmentação, processamento digital de imagens, ovos, pré-processamento, OpenCV.

Abstract

This work presents the development of an algorithm for automatic segmentation of eggs in images using digital image processing techniques. The goal is to develop an efficient and accurate method for identifying and separating eggs in a series of images that contain eggs arranged in rows and columns. The developed method goes through several steps, including image pre-processing, edge detection, egg identification, segmentation and analysis of the results. With the help of the OpenCV, NumPy and Matplotlib libraries, it was possible to implement the algorithm effectively. The results obtained at the end demonstrated 100 percent accuracy in identifying and sorting eggs for all test cases. However, it is important to highlight that there is still room for improvement, such as the integration of machine learning techniques to increase segmentation accuracy. The development of this work contributes as a complement to a research project developed at the Computer Vision and Data Recovery Laboratory at UFU, so that it is feasible to automate the stage of segmenting eggs from images, which was previously carried out semi-automatically.

Keywords: segmentation, digital image processing, eggs, preprocessing, OpenCV.

Lista de ilustrações

Figura 1 – Imagem que representa a equação para cálculo de máscara na filtragem Gaussiana	15
Figura 2 – Imagem representativa do processo de convolução de um filtro Gaussiano	16
Figura 3 – Imagem de imagem antes e depois de passar pela detecção de bordas do algoritmo de Canny	16
Figura 4 – Figura comparativa entre a imagem original e a imagem rotacionada .	22
Figura 5 – Imagem após passar ser convertida para escala de cinza	30
Figura 6 – Imagem após a aplicação do filtro Gaussiano	30
Figura 7 – Imagem resultante após passar pelo método de detecção de bordas Canny	31
Figura 8 – Imagem da linha nº 1 da grade de ovos referente à primeira abordagem	35
Figura 9 – Imagem do ovo nº 1 da primeira linha referente à primeira abordagem	36
Figura 10 – Imagem final de todos os componentes conectados referente à primeira abordagem	36
Figura 11 – Imagem do ovo nº 1 referente à segunda abordagem	41
Figura 12 – Imagem final de todos os componentes identificados referente à segunda abordagem	41

Sumário

1	INTRODUÇÃO	8
1.1	Objetivo	9
1.2	Justificativas	9
2	REVISÃO BIBLIOGRÁFICA	11
2.1	Pré-processamento	11
2.1.1	Redimensionamento da imagem	12
2.1.1.1	Redimensionamento proporcional	12
2.1.1.2	Redimensionamento livre	12
2.1.1.3	Redimensionamento com cortes	12
2.1.1.4	Redimensionamento com interpolação	12
2.1.1.5	Redimensionamento com rotação	13
2.2	Transformada de Hough	13
2.3	Conversão para escala de cinza	13
2.4	Filtro Gaussiano	14
2.5	Detecção de bordas	15
2.6	Segmentação	17
2.7	Modelos e algoritmos	18
3	TRABALHOS RELACIONADOS	19
4	MÉTODO	21
4.1	Aquisição dos Dados	21
4.2	Pré-processamento	21
4.2.1	Rotação da imagem	21
4.2.2	Filtro Gaussiano	22
4.2.3	Detecção de bordas	23
4.2.4	Extração dos contornos	23
4.3	Segmentação dos ovos de galinha	23
5	DESENVOLVIMENTO	25
5.1	Primeira abordagem	26
5.2	Segunda abordagem	34
6	CONCLUSÃO	42

REFERÊNCIAS 43

1 Introdução

O emprego de técnicas de processamento digital de imagens tem ganhado um importância significativa na realidade atual. Suas aplicações trouxeram significativos avanços para processos de seleção, qualidade e análises diversas tanto na indústria como em centro de pesquisas. Nesse contexto, uma importante etapa do processo é a segmentação e identificação desses ovos para futuras análises. Por isso, a análise precisa das dimensões dos ovos desempenha um papel de extrema importância. Essa medida tem a capacidade de prever a taxa de eclosão dos ovos, as características de qualidade da casca, estudos relacionados à dinâmica populacional e à morfologia ecológica, bem como a estimativa do peso dos pintinhos e a avaliação dos conteúdos internos dos ovos, conforme ressaltado por Narushin (2005) em sua pesquisa ([NARUSHIN, 2005](#)).

A fim de efetuar essa análise, é importante proceder à segmentação de um conjunto de imagens representativas de ovos de galinha. Em cada imagem, requer-se a capacidade de discernir, delimitar e segmentar individualmente cada ovo de galinha. Esse propósito é alcançado mediante a aplicação de algoritmos que incorporam técnicas de processamento de imagem, tornando ainda mais eficiente a análise dos ovos. Durante a detecção de ovos, existem quatro etapas principais: aquisição, pré-processamento, segmentação e identificação ou classificação. A segmentação da imagem do ovo de galinha é um processo que pode utilizar diversos métodos de detecção que são comumente usados, como: e thresholding, otsu, k-means, etc. como citado no artigo ([SAIFULLAH; MEHRIDDINOVICH; TOLENTINO, 2021](#)).

Os trabalhos ([BORGES, 2023](#)) e ([NARUSHIN, 2005](#)) abordam de maneira abrangente os objetivos relacionados à análise das dimensões dos ovos de galinha e o impacto dessa análise nas pesquisas científicas e na indústria avícola. No contexto da segmentação de imagens, a documentação da biblioteca OpenCV (Open Source Computer Vision) disponível em ([OPENCV.ORG](#),) apresenta detalhadamente diversos métodos para a segmentação de imagens, oferecendo também exemplos práticos de implementações. Complementando o tópico de processamento de imagens, o artigo ([ULASZEWSKI; JANOWSKI; JANOWSKI, 2021](#)) desenvolvido por Maciej Ulaszewski, Robert Janowski e Andrzej Janowski (2021) detalha minuciosamente os métodos utilizados para detecção de ovos de galinhas utilizando em partes a biblioteca OpenCV.

1.1 Objetivo

Este trabalho tem como objetivo fornecer uma contribuição substancial a um projeto de pesquisa desenvolvido no Laboratório de Visão Computacional e Recuperação de Dados da Universidade Federal de Uberlândia. O referido projeto se dedica à coleta de imagens de ovos de galinha e à realização de análises detalhadas para extrair informações cruciais sobre os ovos.. Atualmente, o método empregado no projeto envolve a segmentação semi-automática de cada imagem contendo ovos, que são dispostos inicialmente em linhas e colunas como uma grade. Esse processo é altamente exigente em termos de recursos humanos, pois é preciso considerar o trabalho humano que deve ser realizado em cada imagem e também, a grande quantidade de imagens que serão dispostas, além de existir a possibilidade da ocorrência de erros durante a segmentação semi-automática, demandando um esforço considerável para realizar a tarefa. Diante disso, este trabalho propõe o desenvolvimento do algoritmo de segmentação automática de ovos, com o objetivo de otimizar e automatizar o procedimento que até então era executado de maneira semi-automática. Essa automação possibilitará um aumento significativo na eficiência e na capacidade do projeto, permitindo processar uma quantidade maior de dados com precisão e eficácia.

1.2 Justificativas

A justificativa para o desenvolvimento do algoritmo de segmentação automática de ovos se baseia em diversas razões fundamentais. Primeiramente, o atual método de segmentação manual demanda uma quantidade significativa de recursos humanos e tempo, tornando o processo intensivo e suscetível a erros humanos. A automação dessa etapa por meio de um algoritmo reduzirá drasticamente a carga de trabalho manual, permitindo uma alocação mais eficiente dos recursos humanos envolvidos no projeto.

Outra justificativa importante é a capacidade do algoritmo em lidar com um volume maior de dados de maneira rápida e eficiente. Com a automação da segmentação, o projeto terá a capacidade de processar uma quantidade significativamente maior de imagens em um período de tempo menor, ampliando assim a capacidade de análise e a abrangência dos resultados obtidos. A adoção de tecnologias avançadas, como a visão computacional, não apenas agiliza os processos, mas também demonstra o compromisso do projeto em buscar soluções eficazes e atualizadas para os desafios enfrentados neste projeto.

Portanto, o desenvolvimento do algoritmo de segmentação automática de ovos é não apenas justificado, mas também essencial para o avanço e o sucesso do projeto de pós-graduação em andamento na Universidade Federal de Uberlândia. A automação dessa etapa não apenas aumentará a eficiência e a precisão da etapa de segmentação dos ovos,

mas também posicionará o projeto na vanguarda da pesquisa em tecnologias aplicadas nessa área.

2 Revisão Bibliográfica

2.1 Pré-processamento

O pré-processamento de imagens é uma etapa destinada a aprimorar a qualidade e a adequação das imagens para análise ou manipulação que serão realizadas nas próximas etapas do processo.

De acordo com o trabalho (GONZALEZ; WOODS, 2000) que aborda todas as etapas do processamento de imagens, a etapa de pré-processamento é essencial, pois é nela que são corrigidas todas as imprecisões, ruídos ou imperfeições que a imagem possa conter. Além disso, o pré-processamento também é responsável por realçar detalhes importantes nas imagens que serão processadas e analisadas posteriormente. De acordo com este trabalho, os defeitos decorrentes da aquisição da imagem são diversos e a correção deles envolvem métodos como: melhoras no brilho e no contraste, redução de ruídos, correção da iluminação irregular, realce de bordas e outros.

Levando em conta o contexto histórico, a necessidade de pré-processamento surgiu com o advento da fotografia digital e da digitalização de imagens analógicas, onde o ruído eletrônico era comum. Avanços recentes na área incluem o uso de técnicas de aprendizado profundo para realizar o pré-processamento de maneira mais automatizada e adaptativa, ajustando-se às características específicas das imagens. Além disso, o pré-processamento também evoluiu a ponto de utilizar redes neurais convolucionais para remover ruídos de imagens. Exemplos de pré-processamento de imagens incluem o reconhecimento de caracteres, remover ruídos eletrônicos em radiologias, remover manchas e imperfeições em imagens antigas, etc.

Neste projeto, o pré-processamento das imagens dos ovos desempenha um papel crucial no aprimoramento da qualidade e adequação das imagens para as análises subsequentes. Por isso, vale destacar a importância da análise específica das características das imagens de ovos de galinha para a escolha e aplicação das técnicas mais adequadas. Por exemplo, ao lidar com imagens de ovos, é comum encontrar desafios como variações na iluminação, manchas ou irregularidades na casca dos ovos, inclinações variadas nas imagens, etc. Todos esses são fatores que podem afetar a qualidade da imagem e a precisão da análise e devem ser considerados durante a etapa de pré-processamentos das imagens.

Nesse contexto, foram desenvolvidas e aplicadas uma variedade de técnicas, adaptadas especificamente para preparar as imagens de ovos de galinha, garantindo que estejam otimizadas para processamento eficiente e que a segmentação dos ovos possa ser realizada com sucesso nas etapas seguintes do projeto.

As técnicas de pré-processamento utilizadas neste projeto foram: redimensionamento da imagem, aplicação de filtro cinza, aplicação de filtro Gaussiano e detector de bordas Canny.

2.1.1 Redimensionamento da imagem

O redimensionamento de imagem é uma técnica bastante útil que tem a objetivo de adaptar a imagem de acordo com a finalidade que será abordada. O redimensionamento envolve várias ideias desde aumentar ou diminuir o tamanho da imagem até rotacionar ela. Existem vários motivos para aplicar redimensionamento de imagem à um trabalho, como adequar a imagem ao tamanho da tela, redução do tamanho do arquivo afim de economizar espaço de armazenamento, melhorar a qualidade visual aumentando a nitidez e detalhamento em uma imagem de baixa resolução, dentre outros. Para isso, existem alguns tipos de redimensionamento de imagens que deve sempre ser considerado ao desenvolver um projeto. Alguns dos tipos que são mais utilizados são: redimensionamento proporcional, o redimensionamento livre, redimensionamento com cortes e redimensionamento com interpolação.

2.1.1.1 Redimensionamento proporcional

Nesta abordagem as dimensões da imagem são alteradas mas de forma que mantém a proporção original da imagem. Este tipo de redimensionamento é útil para ser aplicado em imagens que não podem sofrer distorções de nenhum modo.

2.1.1.2 Redimensionamento livre

Este tipo de redimensionamento permite alterar a largura e a altura da imagem independentemente, podendo resultar em distorções. Geralmente é utilizada de forma mais subjetiva em algum projeto onde as proporções entre as dimensões da imagem não precisam necessariamente ser mantidas da forma como foram concebidas originalmente.

2.1.1.3 Redimensionamento com cortes

O intuito desse tipo de redimensionamento é basicamente diminuir o tamanho da imagem original cortando e descartando segmentos indesejados da imagem.

2.1.1.4 Redimensionamento com interpolação

Este tipo de abordagem altera a quantidade de pixels presente na imagem utilizando diferentes algoritmos para calcular os novos valores de cor. Os algoritmos de interpolação mais comuns são: Nearest Neighbor (simples e rápido, podendo resultar em imagens pixelizadas), Bilinear (um pouco mais suave que a Nearest Neighbor, porém pode

apresentar borrões) e Bicúbica, que oferece uma melhor qualidade dentre todos os outros métodos de interpolação citados, no entanto é o mais lento deles. Neste caso é importante balancear os fatores qualidade e desempenho ao utilizar este tipo de redimensionamento de imagem.

2.1.1.5 Redimensionamento com rotação

Neste tipo de abordagem, o mais importante é rotacionar a imagem, geralmente com o objetivo de alinhar tabelas, linhas, colunas ou grades que estão dispostas na imagem de forma desalinhada. Nesta abordagem pode haver o corte de alguns segmentos da imagem original dependendo do ângulo de rotação que será aplicado.

2.2 Transformada de Hough

A Transformada de Hough é uma ferramenta essencial de visão computacional usada na área de processamento de imagens para detecção de formas geométricas tais como: círculos, elipses, retas, etc.

O funcionamento da transformada de Hough para identificação de retas é destacado no trabalho (DORINI; ROCHA, 2011), onde é explicado de forma resumida que a técnica consiste em mapear cada pixel da imagem para o espaço de parâmetros, que é organizado em forma de um acumulador n-dimensional, onde n corresponde ao número de parâmetros. Para cada pixel contido na imagem, procura-se todas as linhas que passam por essa coordenada. À medida que o mapeamento é realizado para o espaço de parâmetro, o acumulador correspondente àquele mapeamento é incrementado. Por fim basta decidir um valor de threshold para o acumulador. Dessa forma é possível decidir se o valor acumulado consiste ou não em um objeto procurado, neste caso, uma reta.

O método mencionado é de grande utilidade neste trabalho, pois pode ser utilizado na etapa de redimensionamento da imagem, onde será utilizada a técnica de redimensionamento por rotação. Para isso, é realizado o cálculo do ângulo de rotação, baseando-se na mediana dos valores dos ângulos de cada reta, encontrados através da aplicação da Transformada de Hough. Dessa maneira, torna-se possível redimensionar a imagem utilizando a rotação.

2.3 Conversão para escala de cinza

O trabalho (NOGUEIRA, 2016) desenvolvido por Ricardo César de Almeida Nogueira destaca a importância da conversão para escala de cinza na área de processamento digital de imagens, dizendo que é um requisito para muitas aplicações, como em detecção de borda, impressão em preto e branco, e processamento de imagem em único canal.

A conversão para escala de cinza desempenha um papel fundamental no pré-processamento de imagens, simplificando sua manipulação e análise. Ao transformar uma imagem colorida em tons de cinza, essa técnica elimina a informação cromática, concentrando-se apenas na intensidade luminosa de cada pixel. Isso facilita a diferenciação de objetos com base em seus níveis de luminosidade, tornando mais fácil detectar bordas, texturas e outros detalhes importantes na imagem.

Uma das vantagens mais significativas da conversão para escala de cinza é a preservação dos detalhes e texturas da imagem original, enquanto elimina a complexidade das informações de cor. Essa preservação permite que os algoritmos de processamento de imagem realizem análises mais precisas e eficientes, sem comprometer a qualidade ou a clareza dos resultados.

Além disso, a conversão para escala de cinza simplifica significativamente o processamento da imagem, reduzindo tanto o tempo quanto a complexidade computacional envolvidos nas operações subsequentes. Isso é especialmente importante em projetos que lidam com grandes volumes de dados ou que requerem análises em tempo real, onde a eficiência do processamento é essencial.

É importante ressaltar que a conversão para escala de cinza tem uma ampla gama de aplicações em processamento de imagens, incluindo reconhecimento facial, análise de textura, detecção de bordas e compressão de imagens. Todas as vantagens mencionadas a respeito desta técnica de pré-processamento de imagens foram levadas em consideração ao adotá-la no desenvolvimento do projeto em questão.

2.4 Filtro Gaussiano

O filtro Gaussiano é utilizado para suavizar a imagem e reduzir o impacto de ruídos indesejados, que podem ser originados a partir imprecisões da câmera durante a captura das imagens ou por condições ambientais adversas como iluminação, reflexos, imprecisões humanas na captura da imagem, etc.

O trabalho ([MATOSAK; MEDEIROS; RODRIGUES](#),) apresenta a importância da utilização do filtro Gaussiano em processamento de imagens e destaca a vantagem do realce das imagens obtida através da aplicação do filtro para facilitar a extração de feições ou mesmo a correção de ruídos da imagem.

Uma das principais vantagens do filtro Gaussiano é sua capacidade de reduzir a sensibilidade a pequenas variações nos valores dos pixels, o que contribui para uma detecção de bordas mais precisa e estável. Ao suavizar a imagem, o filtro Gaussiano ajuda a destacar as características mais importantes da imagem, tornando a detecção de bordas mais confiável. Além disso, o filtro de Gauss mantém o mesmo número de canais da imagem

de entrada. Isso significa que a informação de cor e intensidade é sempre preservada.

Inicialmente, uma máscara composta por um número ímpar de linhas e colunas é definida. A filtragem ocorre por meio de um processo de convolução, que é realizado deslocando uma máscara (também chamada de filtro) sobre a imagem original, movendo-se um pixel por vez. Cada pixel resultante é obtido através da multiplicação dos valores dos pixels na vizinhança coberta pela máscara pelos valores correspondentes do filtro. Em seguida esses produtos são somados, obtendo o valor de pixel da nova imagem.

Este processo é repetido para todos os pixels da imagem, incluindo aqueles localizados nas bordas, para os quais é criada uma "borda falsa" na imagem, geralmente preenchida com valores nulos.

O trabalho (MATOSAK; MEDEIROS; RODRIGUES,) cita como é realizado o processo de convolução indicado na figura 2 e também como é feito o cálculo para a geração da máscara a partir da equação de distribuição normal bidimensional. Na figura 1, x e y correspondem à distância do ponto da máscara equivalente ao pixel filtrado até o termo a ser definido, medida em unidades de pixel. Já σ é o desvio padrão a ser utilizado na filtragem.

$$h(x, y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Figura 1 – Equação que implementa a máscara utilizada na filtragem Gaussiana.

2.5 Detecção de bordas

A detecção de bordas atua justamente na identificação de transições abruptas nos níveis de intensidade dos pixels, que são realçados após a aplicação de alguma técnica de filtragem, geralmente indicando a presença de uma borda ou contorno.

O trabalho (WANGENHEIM, 2007) desenvolvido por Daniela Marta Seara e Geovani Cássia da Silva Espezim Elizandro aborda o conceito de detecção de bordas e sua utilidade acerca da área de processamento de imagens. Com a detecção de bordas, é possível definir propriedades como áreas, perímetros e forma de determinados componentes na imagem. Além disso, são mencionados os conceitos por trás dos operadores de Roberts e de Sobel por exemplo, que foram largamente utilizados durante a história nas aplicações que envolviam a construção de máscaras para a detecção de bordas em imagens.

No texto (HEISE, 2020) escrito por Luísa Mendes Heise e Guilherme Salustiano são explicados diversos conceitos que permeiam técnicas de filtragem e detecção de bordas em imagens. De acordo com os autores, intuitivamente, pode-se dizer que uma borda é

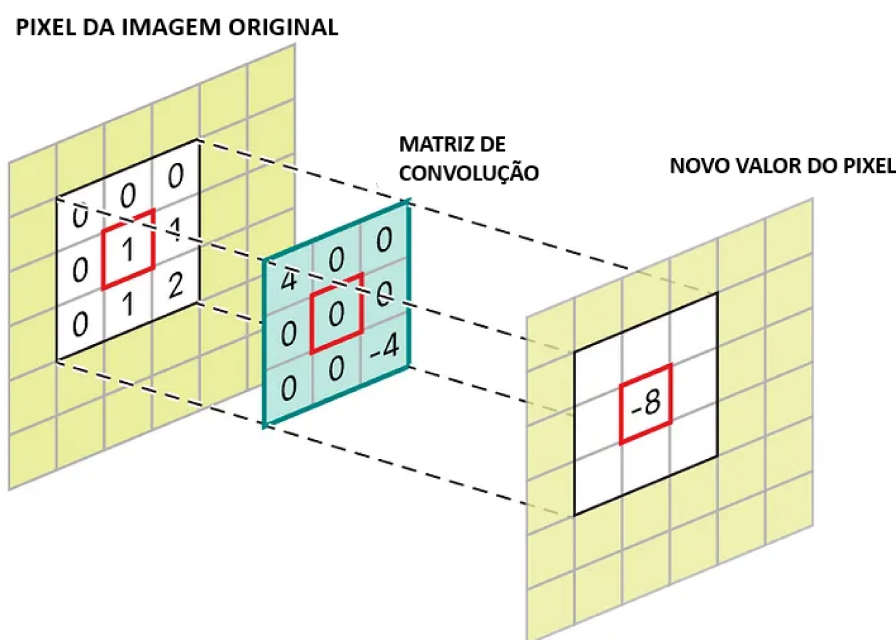


Figura 2 – Processo de convolução de um filtro Gaussiano. Imagem adaptada de (SHARDA, 2021)

um “lugar geométrico” que delimita um objeto qualquer, no entanto esta definição é imprecisa e precisa ser incrementada. Por isso, ainda neste artigo, os autores explicam que outra forma de se identificar bordas em uma imagem é procurar por pontos onde a intensidade luminosa muda abruptamente. No artigo são abordados diversos conceitos e técnicas importantes para a detecção de bordas em uma imagem. Dentre várias técnicas de detecção de bordas como o operador de Sobel, a mais relevante e aplicável a este projeto chama-se "Detector de borda de Canny". A figura 3 extraída do artigo (HEISE, 2020) apresenta duas imagens, a primeira à esquerda representa a imagem original. A segunda é a mesma imagem após passar pelo processo de detecção de bordas implementado no algoritmo de Canny.

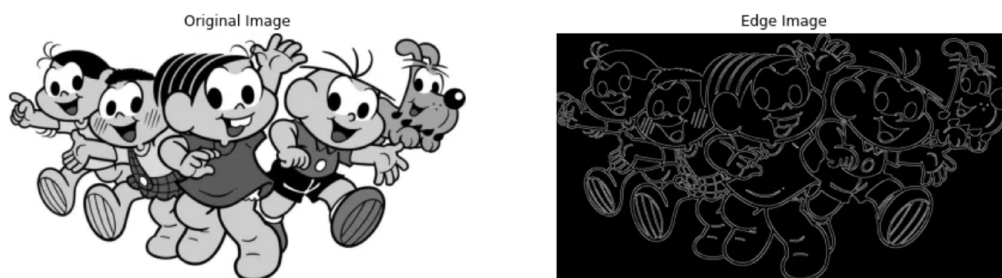


Figura 3 – Aplicação do algoritmo de Canny para detecção de bordas. Imagem extraída do texto (HEISE, 2020)

Todas as técnicas que serão mencionadas, podem ser adaptadas e aplicadas com

o auxílio da biblioteca de visão computacional OpenCV, utilizada para desenvolver primordialmente a maioria das etapas deste trabalho. Historicamente, a filtragem e detecção de bordas tiveram origem nas décadas de 1960 e 1970, com o desenvolvimento de operadores como o operador Sobel e o operador de Roberts. Avanços recentes na área também incluem o uso de técnicas de aprendizado profundo, como as redes neurais convolucionais, para realizar filtragem e detecção de bordas de maneira mais eficaz e adaptativa. Os exemplos mais comuns de uso destas técnicas são: detecção de contornos em imagens médicas como tomografia ou radiografia para diagnóstico e realce de características em imagens de satélite para análise geoespacial.

Neste projeto, o detector de bordas Canny foi utilizado para identificar os contornos dos ovos nas imagens digitais. O objetivo era segmentar automaticamente os ovos para posterior análise. O detector de bordas Canny funciona através da suavização da imagem com um filtro Gaussiano aplicado anteriormente na etapa de pré-processamento, cálculo do gradiente da imagem suavizada, redução da largura das bordas e aplicação de um algoritmo de histerese para eliminar bordas falsas (que não estiverem ligadas a bordas fortes).

2.6 Segmentação

A segmentação de imagens é o processo de dividir uma imagem em regiões ou objetos significativos com base em características específicas, como cores, texturas, bordas ou conteúdo semântico. Inicialmente, as primeiras técnicas de processamento de imagens digitais, incluindo a segmentação de imagens, não obtiveram grande atenção dos estudiosos. Contudo, a partir da década de 1960, essa área começou a ter o seu reconhecimento, isso porque imagens da Lua passaram a ser processadas com a intenção de remover as distorções (Gonzales e Woods 2001).

Desde então, essa área de pesquisa evoluiu de forma significativa, de acordo com o trabalho ([ALBANEZ MARCOS AURÉLIO BATISTA, 2016](#)) realizado por Daniela de Oliveira Albanez, Marcos Aurélio Batista e Sérgio Ferreira da Silva. Avanços subsequentes viram o desenvolvimento de abordagens mais sofisticadas, como a segmentação por regiões, que agrupa os pixels semelhantes formando regiões coerentes na imagem.

Além disso, o uso de redes neurais convolucionais também trouxe avanços significativos na segmentação das imagens, permitindo a identificação precisa de objetos complexos em imagens. Alguns exemplos do uso de segmentação de imagens são: a identificação de órgãos em imagens médicas, separação de objetos em fotografias para reconhecimento automático e a segmentação de estruturas em imagens de satélite para mapeamento urbano e ambiental.

Além dos métodos de segmentação mencionados anteriormente, outro método am-

plamente utilizado e que foi aplicado à este trabalho é a chamada segmentação por bordas. Nesta etapa, as bordas dos ovos na imagem são identificadas por meio do método de detecção de bordas Canny como transições abruptas de intensidade de pixel, que indicam mudanças significativas nas propriedades da imagem, como mudanças de cor, textura ou profundidade e foram utilizadas para realizar a etapa de segmentação. Desta maneira foi possível realizar a segmentação dos ovos na imagem após detectá-los como bordas. Assim, com as coordenadas de cada ovo e as bordas bem detalhadas, foi possível conceber a segmentação desejada.

2.7 Modelos e algoritmos

As redes neurais convolucionais (CNNs) representam uma classe de algoritmos de aprendizado de máquina que são projetadas para imitar o processamento visual humano e são especialmente eficazes na análise de dados de imagens. O artigo ([RAVINDRA, 2017](#)), mostra como as redes neurais convolucionais são utilizadas para reconhecer objetos em imagens.

De acordo com este artigo, o reconhecimento de imagens é um método de aprendizagem de máquina projetado para se assemelhar à maneira como o cérebro humano funciona. Com este método, os computadores ensinam a reconhecer os elementos visuais dentro de uma imagem. Ao confiar em bancos de dados grandes e observar padrões emergentes, os computadores podem analisar imagens e formular tags e categorias relevantes que fazem sentido. O objetivo é utilizar um banco de imagens semelhantes a fim do algoritmo ser capaz de reconhecer estes padrões e a partir disso conseguir identificar o objeto desejado em diversas imagens.

As CNNs tiveram origem nas décadas de 1980 e 1990, mas foi no início dos anos 2010 que elas ganharam destaque com o desenvolvimento de arquiteturas como a AlexNet e a VGGNet, que alcançaram resultados impressionantes em competições de reconhecimento de objetos. Exemplos notáveis do uso de CNNs incluem reconhecimento de objetos, classificação de imagens médicas, segmentação semântica de imagens urbanas e até mesmo a geração de imagens realistas por meio das chamadas Redes Generativas Adversariais (GANs).

Embora as redes neurais convolucionais (CNNs) sejam uma ferramenta poderosa e amplamente utilizada em uma variedade de aplicações de visão computacional, neste trabalho, foram utilizados outros algoritmos e métodos para realizar o processamento dos ovos de galinha nas imagens fornecidas.

3 Trabalhos Relacionados

O artigo ([HEISE, 2020](#)) aborda a Detecção de Bordas e Transformações Morfológicas em Imagens utilizando a biblioteca OpenCV. O objetivo central do artigo é aprimorar a capacidade de identificar contornos e características fundamentais em determinadas imagens, proporcionando uma base sólida para a segmentação de elementos específicos. O método empregado envolve a aplicação de algoritmos de detecção de bordas, como o Canny, e transformações morfológicas, tais como erosão e dilatação, para aprimorar a definição e o destaque das bordas dos objetos na imagem. Os principais resultados destacam a eficácia dessas técnicas no realce de características estruturais, proporcionando uma preparação adequada para etapas subsequentes de processamento. Este artigo aborda um tema importante para o desenvolvimento do meu projeto, que por sua vez fará o uso de algumas das ferramentas e métodos descritos no artigo, além de estabelecer uma base sólida na utilização da biblioteca OpenCV, que também será amplamente utilizada no trabalho para a segmentação de ovos de galinha. A compreensão aprofundada das técnicas de detecção de bordas e transformações morfológicas fornecidas por este artigo é essencial para a implementação bem-sucedida do algoritmo de Processamento Digital de Imagens no contexto específico da minha pesquisa.

O trabalho ([ULASZEWSKI; JANOWSKI; JANOWSKI, 2021](#)) conduzido pelos autores Maciej Ulaszewski e Robert Janowski and Andrzej Janowski aborda a aplicação de visão computacional para a detecção de ovos em uma linha de produção em tempo real. O principal objetivo deste estudo é explorar e implementar soluções eficazes para a detecção automatizada de ovos, utilizando técnicas avançadas de visão computacional que são fortes candidatos para o uso neste projeto. No artigo, os autores desenvolveram um software dedicado utilizando Python3 e bibliotecas especializadas, destacando a versatilidade e eficácia dessa linguagem de programação no contexto da visão computacional. Duas abordagens principais foram investigadas: o uso de redes neurais, incluindo os modelos SSD-Mobilenetv2, FR-CNN e YOLOv3, e abordagens de correspondência de modelos. A pesquisa tem o intuito de analisar e comparar a eficácia dessas abordagens na detecção precisa de ovos em tempo real em uma linha de produção. Este trabalho se relaciona diretamente com o estudo em desenvolvimento, uma vez que ambos estão centrados na detecção de ovos, embora com abordagens diferentes. Enquanto o meu trabalho se concentra em técnicas de processamento digital de imagens, este artigo explora a aplicação de redes neurais e correspondência de modelos, fornecendo insights que podem complementar a abordagem adotada na minha pesquisa.

A detecção de ovo de galinha com base no conceito de processamento de imagem é abordada no artigo ([SAIFULLAH; MEHRIDDINOVICH; TOLENTINO, 2021](#)). Nesse

artigo, a detecção de ovos de galinha é abordada enfatizando a relevância para a indústria avícola. O estudo evidencia a capacidade de prever peso, características físicas e até mesmo características embrionárias a partir da análise de imagens processadas dos ovos. Por conta disso, o objetivo principal deste artigo é discutir os conceitos fundamentais do processamento de imagens na detecção de ovos de galinha e sua aplicação técnica. Dentro desse contexto, são destacadas quatro partes essenciais do processo de processamento de imagens: aquisição de imagens, pré-processamento, segmentação e identificação ou classificação. Este estudo é relevante para o meu trabalho, pois reforça a compreensão detalhada das etapas mencionadas no artigo que servirão de base para a definição das etapas deste trabalho e posteriormente para o desenvolvimento do algoritmo relacionado a ele.

4 Método

A metodologia aplicada no trabalho segue uma linha estruturada e sequencial, abordando os seguintes conceitos:

1. Aquisição dos Dados
2. Pré-processamento
3. Escolha da Técnica de Segmentação
4. Desenvolvimento da solução
5. Análise e Interpretação dos Resultados
6. Elaboração da documentação

O trabalho conta com dois algoritmos finais que seguem abordagens distintas a partir de um determinado ponto. No entanto, ambos foram capazes de alcançar o mesmo resultado no desfecho do processo. A seguir, apresenta-se uma explicação detalhada e o objetivo de cada etapa da metodologia adotada:

4.1 Aquisição dos Dados

A aquisição dos dados é uma etapa fundamental em qualquer projeto de análise de imagens, pois a qualidade e a representatividade dos dados têm um impacto direto nos resultados obtidos. Neste projeto, foram utilizadas imagens produzidas no trabalho de (TOMÉ, 2004), nelas, a aquisição das imagens foram realizadas por um dispositivo construído para este devido fim. A quantidade de dados não necessariamente precisa ser grande, mas deve abranger todos diferentes casos. Dessa forma foi possível implementar um algoritmo que fosse capaz de segmentar os dados de forma minuciosa e eficiente.

4.2 Pré-processamento

4.2.1 Rotação da imagem

A primeira etapa do algoritmo se concentrou na correção da inclinação das imagens dos ovos. Isso é crucial para garantir que a grade de ovos esteja o mais alinhada possível, isto é, que as linhas de ovos estejam paralelas à 180° ou bem próximas disso, facilitando significativamente a associação dos ovos que fazem parte da mesma linha na grade. Esta

assosiação leva em consideração uma diferença máxima entre as coordenadas verticais de cada ovo. Por conta disso, a rotação da imagem com o objetivo de alinhar os ovos desempenha uma função crucial para que a assosiação de ovos por linha seja bem-sucedida, diminuindo ao mínimo a possibilidade de erros na etapa de agrupamento dos ovos por linha.

A figura 4 a seguir mostra a comparação entre duas imagens. A primeira é a imagem original e a segunda, a imagem rotacionada, com o alinhamento adequado.

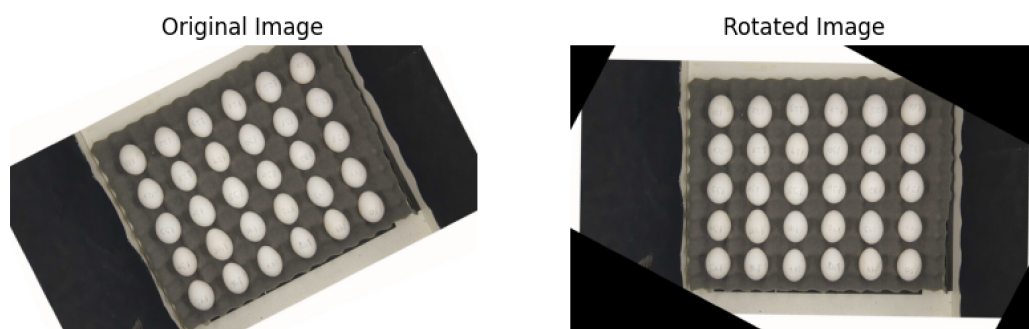


Figura 4 – Comparação entre a imagem original e a imagem após a etapa de rotação, com o alinhamento adequado.

Para realizar essa correção de inclinação, foi utilizado a Transformada de Hough, uma técnica poderosa em visão computacional para detecção de formas geométricas, como círculos, quadriláteros e linhas retas. Neste trabalho, a Transformada de Hough é responsável por identificar as linhas presentes na imagem que representam as bordas da grade de ovos e assim torna possível o cálculo do ângulo de rotação que será utilizado para alinhar a grade de ovos a 180° , paralela à borda da imagem. O valor do ângulo de inclinação é calculado a partir da mediana dos ângulos das retas encontradas. Dessa maneira a imagem é rotacionada de acordo com o valor do ângulo de inclinação obtido e posteriormente, é submetida à próxima etapa.

Antes da segmentação dos ovos, é necessário aplicar uma série de técnicas de tratamento de imagem para prepará-la para a segmentação. Primeiramente, a imagem é convertida para escala de cinza, o que simplifica o processamento, reduzindo-a a uma única intensidade luminosa por pixel.

4.2.2 Filtro Gaussiano

Em seguida, aplica-se um filtro Gaussiano para suavizar a imagem e reduzir o ruído, garantindo uma detecção mais precisa das bordas dos ovos. Este filtro atenua as transições abruptas de intensidade na imagem que podem ter sido provocados por imprecisões da camera ou condições ambientais, resultando em uma imagem mais suave e uniforme.

4.2.3 Detecção de bordas

Após a aplicação do filtro Gaussiano, é utilizado o método de detecção de bordas conhecido como algoritmo "canny". Este algoritmo identifica as bordas presentes na imagem através da análise do gradiente da imagem. Ou seja, este método é capaz de identificar bordas presentes na imagem baseando-se na mudança de intensidade da imagem, destacando as transições de intensidade que correspondem às bordas dos objetos presentes na cena. As bordas detectadas são representadas por pixels brancos em um fundo preto, facilitando sua identificação e análise.

4.2.4 Extração dos contornos

Por fim, serão extraídos os contornos das bordas detectadas, que representam as formas dos ovos dispostos na imagem e serão utilizados como base para a segmentação dos ovos durante a próxima etapa do algoritmo.

4.3 Segmentação dos ovos de galinha

A segmentação dos ovos é executada de modo a identificar cada ovo em sua respectiva posição na grade da imagem de maneira precisa, garantindo que a ordem dos ovos não seja alterada. Esta ordem é estabelecida da esquerda para a direita e de cima para baixo, ou seja, o primeiro ovo é o mais a direita acima, ao mesmo passo que o último ovo é aquele que se encontra mais a direita na região inferior. O processo de ordenação dos ovos é conduzido por meio da seguinte abordagem:

Inicialmente, os ovos são separados em linhas, ou seja, pertencerão ao mesmo grupo apenas ovos que estejam posicionados na mesma linha da grade. Para conseguir discernir se um ovo está na mesma linha que outro foi utilizado como critério de comparação a posição de cada ovo na imagem, levando em consideração sua coordenada no eixo horizontal. Ou seja, se a diferença entre valor das coordenadas horizontais de cada ovo estiver dentro de um limite pré-estabelecido, significa que estes ovos estão posicionados na mesma linha do grid, portanto são adicionados ao mesmo grupo.

O método de redimensionamento da imagem por rotação aplicado na etapa de pré-processamento, desempenha um papel fundamental nesse processo. Ele é responsável por manter os ovos alinhados o mais próximo possível da grade, o que viabiliza o estabelecimento dessa relação de linhas entre os ovos. Graças a esse método, a segmentação pode ser conduzida de forma mais precisa e eficiente, garantindo a identificação correta da posição de cada ovo na imagem da grade.

Após a definição dos grupos de ovos com base nas linhas da grade, o próximo passo é ordená-los de acordo com suas coordenadas verticais. Esse processo permite que os ovos

sejam organizados de maneira sequencial, seguindo a ordem especificada anteriormente. Ao final, é realizado um procedimento de separação dos ovos em imagens menores, ou seja, para cada ovo é gerada uma imagem e armazenada com um título que indica a sua posição original na grade. Por exemplo: o primeiro ovo da grade é segmentado ao final deste processo e recebe o nome do arquivo de "1.jpeg", para que seja possível identificar cada respectivo ovo apenas pelo seu título ao final da segmentação. Ao término do processo, tem-se em uma pasta contendo imagens dos ovos dispostos de maneira sequencial, conforme o critério de ordenação estabelecido.

5 Desenvolvimento

Durante todo o processo de implementação do algoritmo, a biblioteca OpenCV desempenhou um papel fundamental. O OpenCV é uma biblioteca de código aberto amplamente reconhecida e utilizada em projetos de visão computacional, oferecendo uma variedade de funcionalidades para manipulação e processamento de imagens. Sua documentação ([OPEN...](#),) foi amplamente utilizada para auxiliar em diversas partes do código e ao longo do desenvolvimento do algoritmo, esta biblioteca foi utilizada para realizar operações essenciais, como leitura de imagem, manipulação de cores, detecção de bordas, aplicação de filtros, etc. Sua ampla variedade de funções e algoritmos pré-implementados simplificou significativamente o desenvolvimento do código, permitindo realizar uma implementação mais eficiente para obter a solução.

Durante o desenvolvimento da solução, foram implementadas duas abordagens semelhantes, cada uma delas com suas próprias nuances. Essas abordagens foram concebidas com o objetivo comum de solucionar o problema proposto neste trabalho: a segmentação precisa dos ovos em imagens de grade de maneira ordenada. Ao final desse processo, foram obtidos dois algoritmos distintos, ambos capazes de atingir o mesmo objetivo final. Nesta seção, será detalhado como cada uma dessas abordagens foi implementada, desde o início até a obtenção dos algoritmos finais.

Além do OpenCV, foram utilizadas as bibliotecas NumPy e Matplotlib para complementar o desenvolvimento do algoritmo. A biblioteca NumPy é amplamente reconhecida por suas poderosas capacidades de manipulação de arrays e operações matemáticas, fornecendo uma base sólida para o processamento eficiente de dados numéricos. Neste projeto a biblioteca NumPy foi utilizada principalmente no cálculo de ângulo de inclinação da imagem, no cálculo da mediana, manipulação de arrays e na cópia de imagens.

A biblioteca Matplotlib foi empregada para visualização e análise dos resultados obtidos durante o desenvolvimento do algoritmo. Com o Matplotlib, não apenas foi possível exibir os resultados finais do algoritmo, mas também visualizar detalhes intermediários do processamento das imagens, como os resultados parciais de cada etapa do pré-processamento e segmentação. Essa capacidade de visualização detalhada permitiu uma análise minuciosa do desempenho do algoritmo em diferentes estágios, servindo como uma espécie de "debugger visual", permitindo identificar e corrigir rapidamente possíveis problemas ou falhas no algoritmo. Essa ferramenta de visualização tornou o processo de desenvolvimento mais eficiente, contribuindo significativamente para a qualidade do algoritmo final.

5.1 Primeira abordagem

Para iniciar a implementação do algoritmo, foi criada a função principal denominada **process_image**. Esta função desempenha um papel central, sendo responsável por conduzir a etapa inicial de leitura e pré-processamento da imagem de entrada. Além disso, essa função também engloba chamadas de outras funções auxiliares, as quais foram desenvolvidas com o propósito de modularizar e facilitar o entendimento do código, caso seja necessário realizar manutenções futuramente.

Ao chamar a função **process_image**, dentro da função a primeira coisa que realizada é a leitura da imagem de entrada com o auxílio da biblioteca openCV. A linha de código que implementa esta leitura é indicada pelo fragmento de código 1.

Algorithm 1 Leitura da imagem

```
original_image = cv2.imread(image_path)
```

Essa linha de código utiliza a função **cv2.imread** do OpenCV para carregar a imagem localizada no caminho especificado pela variável **image_path**. O resultado dessa operação é armazenado na variável **original_image**, que contém a imagem em sua forma original e não processada.

Após realizar a leitura da imagem, o próximo passo é modificar a imagem utilizando redimensionamento por rotação, para que os ovos contidos na grade da imagem estejam o mais alinhados possível. Para realizar a rotação ideal da forma mais precisa possível, é necessário calcular o ângulo de inclinação da imagem. Para isso, é chamada a função auxiliar **find_inclination_angle** que leva como parâmetro a imagem lida inicialmente. A chamada desta função está indicada na linha 2.

Algorithm 2 Encontra ângulo de inclinação da imagem

```
inclination_angle = find_inclination_angle(original_image)
```

A função auxiliar **find_inclination_angle** é responsável por encontrar o ângulo de inclinação contido na imagem a partir de um método chamado Transformada de Hough, utilizado neste projeto para identificar retas contidas na imagem. No entanto, dentro do escopo da função **find_inclination_angle**, antes de identificar as retas, é necessário realizar um pré-tratamento na imagem, convertendo ela para a escala de cinza e detectando contornos com o auxílio das respectivas funções do OpenCV: **cvtColor** e **Canny**. O trecho de código que realiza este tratamento na imagem está contido no fragmento 3.

A função **cvtColor** recebe como parâmetros a imagem que será convertida de BGR (escala padrão de cores do OpenCV) para escala de cinza, ou seja, cada pixel da imagem resultante terá um valor de intensidade luminosa, ignorado as cores.

Algorithm 3 Tratamento da imagem para identificação das linhas

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 50, 150, apertureSize=3)
```

A função **Canny** recebe quatro parâmetro. O primeiro é a imagem em escala de cinza. O segundo e o terceiro parâmetros representam o limite inferior e limite superior para detecção de bordas respectivamente. Os limites. Pixels com gradiente de intensidade dentro dessa faixa serão identificados como parte da borda. O último parâmetro representa as dimensões do operador de Sobel que será utilizado para encontrar gradientes de intensidade. O valor 3 indica que o operador de Sobel usará uma janela de 3x3 pixels para calcular os gradientes.

Após realizar este tratamento, as linhas presentes na imagem serão identificadas com o auxílio da função **HoughLines** também do OpenCV. Este fragmento de código está indicado no algoritmo 4

Algorithm 4 Identificação das linhas contidas na imagem

```
lines = cv2.HoughLines(edges, 1, np.pi/180, 200)
```

Nesta função o primeiro parâmetro é a imagem de bordas na qual serão detectadas as linhas. O segundo parâmetro representa a distância mínima em pixels entre duas linhas para serem consideradas distintas. O terceiro indica a precisão do ângulo das linhas detectadas. E por último, o quarto parâmetro representa o limite mínimo de votos ou número mínimo de interseções necessárias para detectar uma linha na transformada de Hough. Quanto maior for esse limite, mais restrita se torna a detecção.

Após detectar as linhas utilizando a transformada de Hough, o código a seguir extrai os ângulos das linhas detectadas e calcula o ângulo de inclinação do conjunto de todos os ângulos encontrados.

Primeiro, cria-se um array vazio para armazenar os valores dos ângulos que serão calculados. Em seguida, o loop é responsável por percorrer todas as linhas e obter o ângulo em radianos da linha em relação ao eixo x, representado pelo valor theta. Após encontrar um valor de theta, este é convertido para graus com a ajuda da biblioteca NumPy. Ao final do laço, o valor do ângulo é adicionado à lista de ângulos. Depois, calcula-se a mediana da lista e obtém-se o valor do ângulo que será retornado pela função. No fragmento de código 5 é apresentada a lógica que implementa esta etapa.

O próximo passo é calcular o ângulo de rotação que será aplicado à imagem. O critério para a rotação se baseia na seguinte condição: caso o ângulo de inclinação obtido seja maior que 90, o ângulo de rotação receberá o valor equivalente ao ângulo de inclinação - 180. Caso contrário, o ângulo de rotação receberá o próprio valor da inclinação. Essa

Algorithm 5 Obtenção do ângulo de inclinação da imagem

```

angles = []
if lines is not None:
    for line in lines:
        rho, theta = line[0]
        angle = np.degrees(theta)
        angles.append(angle)

if angles:
    inclination_angle = np.median(angles)
else:
    inclination_angle = 0.0

return inclination_angle

```

cláusula tem o objetivo de limitar o ângulo de rotação para casos em que a imagem original estivesse muito inclinada, e não acabar rotacionando a imagem ao contrário.

Ao obter o valor do ângulo de rotação da imagem, a função **process_image** chama a função auxiliar **rotate_image** para de fato realizar a rotação da imagem, passando como parâmetros a imagem original e o ângulo de rotação obtido anteriormente.

A função **rotate_image** calcula o ponto central da imagem usando as dimensões da imagem e o converte em uma tupla. Em seguida, é gerada uma matriz de rotação usando o ponto central e o ângulo fornecido, utilizando a função **getRotationMatrix2D**. Essa matriz de rotação é então aplicada à imagem original usando a função **warpAffine**, resultando na imagem rotacionada, que é retornada como resultado da função.

A função **getRotationMatrix2D** recebe três parâmetros: o ponto central de rotação da imagem, o ângulo de rotação e por último a escala aplicada a rotação.

A função **warpAffine** recebe quatro parâmetros: a imagem, a matriz obtida anteriormente, o tamanho da imagem de saída e a flag que indica o tipo de interpolação que será utilizado na rotação da imagem, neste caso, **INTER_LINEAR**, que indica uma interpolação linear suave. O código implementado dentro da função **rotate_image** pode ser visualizado no algoritmo 6.

Algorithm 6 Rotação da imagem de acordo com o ângulo de inclinação

```

center = tuple(np.array(image.shape[1::-1]) / 2)
rotation_matrix = cv2.getRotationMatrix2D(center, angle, 1.0)
rotated_image = cv2.warpAffine(image, rotation_matrix,
image.shape[1::-1], flags=cv2.INTER_LINEAR)
return rotated_image

```

Todas as funções auxiliares que foram chamadas dentro do contexto da função

rotate_image são pré-implementadas e fornecidas pela biblioteca OpenCV.

Após a rotação da imagem, procede-se com a limpeza dos diretórios de resultados, evitando qualquer mistura com os resultados de execuções anteriores. Em seguida, a imagem rotacionada é submetida a um processo de tratamento composto por duas etapas principais: conversão para escala de cinza e aplicação do filtro Gaussiano. O código relacionado à esta etapa está em algoritmo 7

Algorithm 7 Tratamento da imagem rotacionada

```
gray = cv2.cvtColor(rotated_image, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (9, 9), 0)
```

A função **cvtColor** é utilizada para converter a imagem para escala de cinza que será posteriormente utilizada para aplicação do filtro Gaussiano, que funciona da seguinte maneira: a função **GaussianBlur** recebe como parâmetros a imagem em escala de cinza, a matriz de 9x9 pixels que é aplicada para suavizar a imagem e o desvio padrão do filtro Gaussiano, que neste caso recebe o valor padrão 0, indicando um cálculo automático do desvio padrão com base no tamanho do filtro.

Em seguida, a imagem tratada é submetida ao processo de detecção de bordas com o detector Canny. Depois a identificação dos contornos é feita pela função **findContours** que recebe de entrada a lista de bordas detectadas e as flags **RETR_EXTERNAL** e **CHAIN_APPROX_SIMPLE** que representam respectivamente o modo de recuperação do contorno, indicando apenas contornos externos e o método de compreensão do contorno, utilizando apenas os pontos de extremidade. O código que implementa esta etapa está contido no algoritmo 9

Algorithm 8 Detecção de bordas e identificação dos contornos

```
edges = cv2.Canny(blur, canny_low, canny_high)
contours, __ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

As figuras 5 6 e 7 representam a imagem após passar pelas etapas de conversão para escala de cinza, aplicação do filtro Gaussiano e detecção de bordas com o método Canny, respectivamente.

Para garantir que apenas os contornos relevantes sejam considerados na etapa seguinte, é realizado um processo de filtragem dos contornos. Nesse processo, a lista de contornos é percorrida, e apenas aqueles contornos cuja área seja superior ao limite mínimo predefinido de 500 são adicionados a uma nova lista de contornos filtrados. Essa abordagem ajuda a eliminar contornos indesejados que poderiam ter sido identificados anteriormente.

Neste estágio do algoritmo, já foram identificados os contornos de todos os ovos presentes na imagem. Agora, é necessário delimitar cada contorno com um retângulo, o

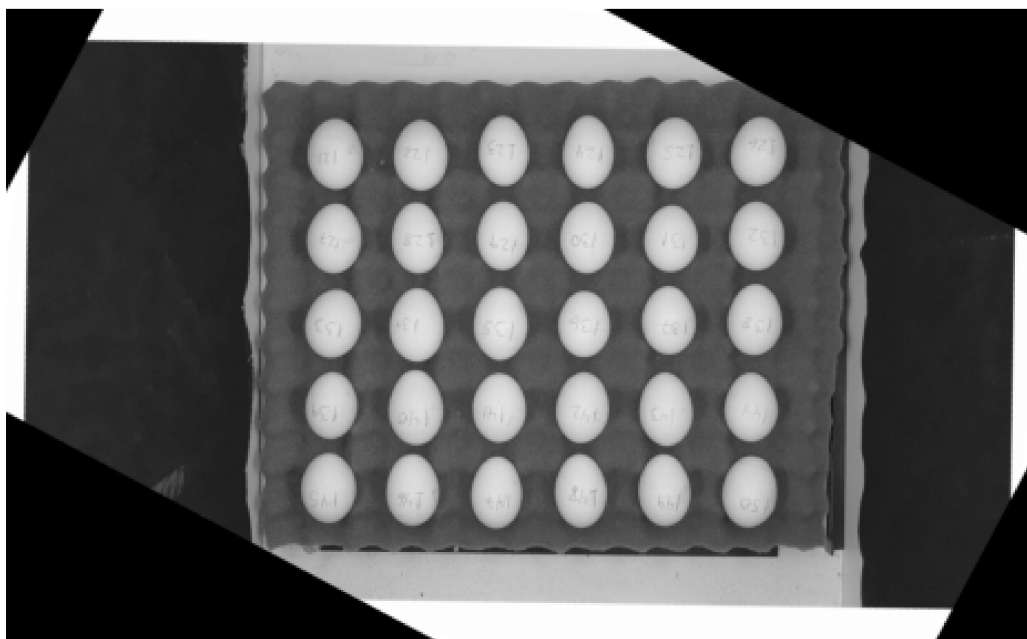


Figura 5 – Imagem resultante após passar pela conversão para escala de cinza.

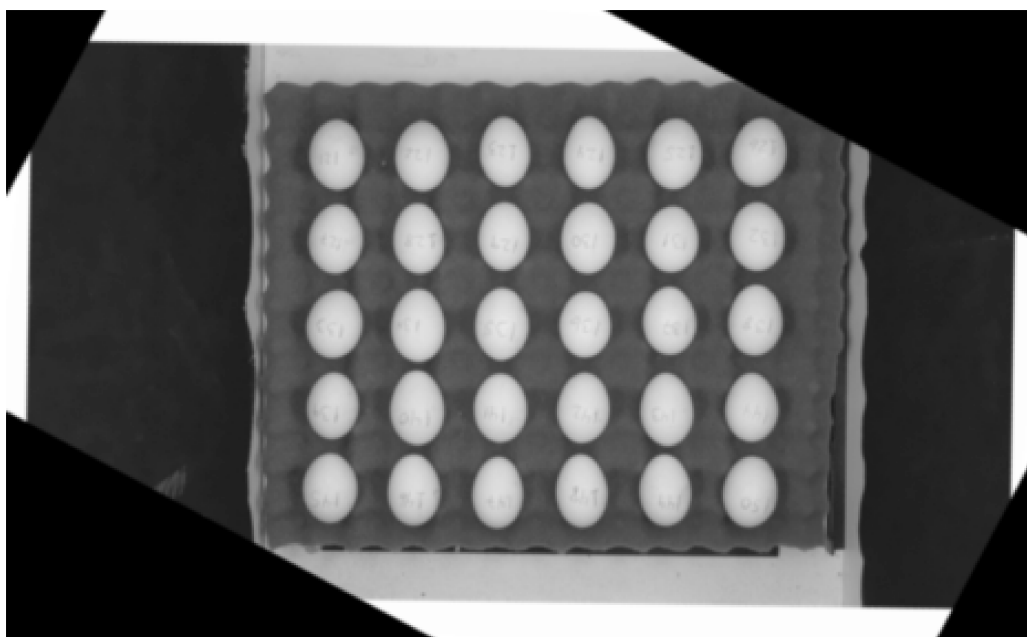


Figura 6 – Imagem resultante após aplicação do filtro Gaussiano.

que é feito percorrendo a lista de contornos filtrados e atribuindo quatro variáveis x , y , w , h a cada contorno, que representam respectivamente as coordenadas dos pontos inferior e superior da esquerda e o comprimento e altura do retângulo. Além disso, para garantir que o retângulo delimitador não se sobreponha às bordas do ovo, uma folga específica foi adicionada, garantindo que nenhuma parte do contorno seja perdida. O trecho de código responsável por essa tarefa é o algoritmo 9:

Ao final do laço, obtém-se uma lista de retângulos delimitadores, que contém cada

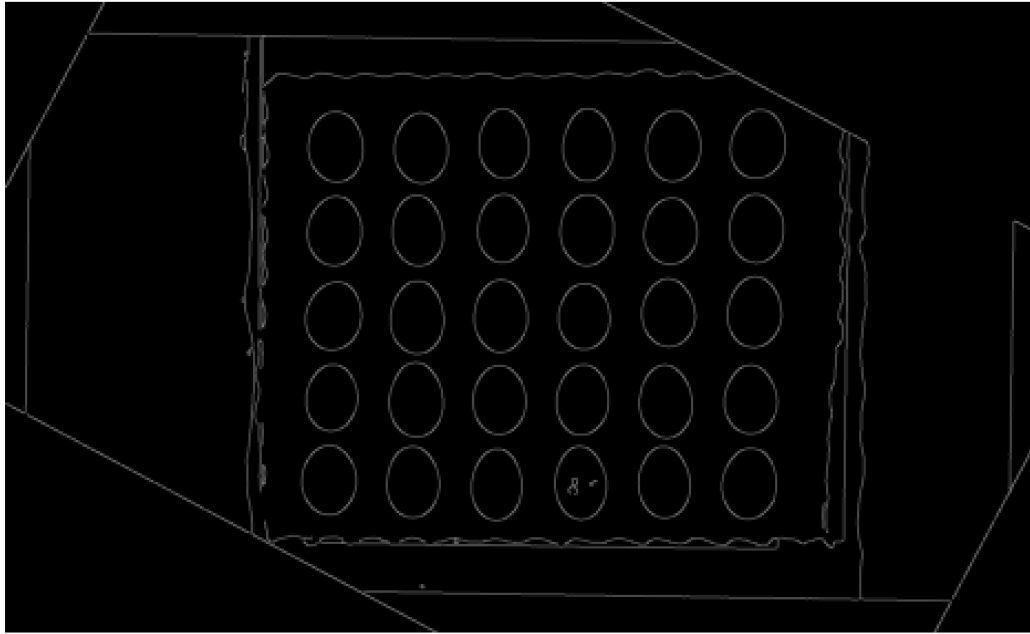


Figura 7 – Imagem resultante após a utilização do algoritmo Canny para detecção de bordas

Algorithm 9 Detecção de bordas e identificação dos contornos

```

bounding_rectangles = []
for contour in filtered_contours:
    x, y, w, h = cv2.boundingRect(contour)
    looseness_x = int(w * looseness_factor)
    looseness_y = int(h * looseness_factor)
    x = max(0, x - looseness_x)
    y = max(0, y - looseness_y)
    w += 2 * looseness_x
    h += 2 * looseness_y
    bounding_rectangles.append((x, y, w, h))

```

ovo da imagem. Agora, é necessário associar componentes que estão conectados, que neste caso, são aqueles que se encontram na mesma linha da grade de ovos. Para isso, é criado um laço de repetição que percorre pela lista de retângulos delimitadores e verifica se o retângulo em questão já foi visitado. Este código é implementado no algoritmo 10.

Caso ainda não estiver sido visitado, a função auxiliar **search_components** é chamada. Esta função implementa uma busca para encontrar os componentes conectados a um retângulo delimitador. O critério utilizado para que um retângulo pertença à mesma linha que outro é que a diferença entre a coordenada y de seus centroides seja menor que a distância máxima pré-determinada como 30. Os centroides são calculados através de outra função auxiliar chamada de **calculate_centroid**. Caso satisfaça esta condição, significa que eles pertencem a mesma linha. O processo é realizado para todos os retângulos, e a função retorna um componente conectado, que é uma lista de retângulos delimita-

Algorithm 10 Verificação se componente já foi visitado

```

connected_components = []
visited = set()
for rect in bounding_rectangles:
    if rect not in visited:
        connected_component = search_components(rect,
        bounding_rectangles, visited)
        connected_components.append(connected_component)

```

dores pertencentes à mesma linha. A função **search_components** está implementada no algoritmo 11

Algorithm 11 Associação entre componentes conectados

```

connected_component = []
stack = [start_rect]
while stack:
    rect = stack.pop()
    if rect not in visited:
        connected_component.append(rect)
        visited.add(rect)
        for neighbor in rectangles:
            if neighbor not in visited:
                centroid1 = calculate_centroid(rect)
                centroid2 = calculate_centroid(neighbor)
                if abs(centroid1[1] - centroid2[1]) < max_distance:
                    stack.append(neighbor)
return connected_component

```

Ao retornar o componente conectado para a função principal **process_image**, este componente é adicionado à uma lista de componentes conectados, que basicamente representa uma lista que contém todas as linhas de ovos, onde cada linha é representada por uma lista de retângulos delimitadores.

Agora que todos os ovos foram identificados e separados por linhas, resta apenas ordená-los pelo eixo vertical. Dessa maneira é possível segmentar os ovos sem perder a ordem deles na imagem original. A ordenação é realizada no fragmento de código 12:

Algorithm 12 Ordenação dos componentes pelo eixo vertical

```

connected_components.sort(key=lambda component:
calculate_centroid(component[0])[1])

```

Depois que todas as linhas de ovos na imagem estão devidamente ordenadas, o código chama a função **save_row_images**, passando como parâmetro a imagem rotacionada

no início e os componentes conectados já ordenados. Esta função é responsável por criar uma representação visual das linhas de ovos na imagem, desenhando retângulos verdes ao redor dos ovos em cada linha e salvando essas imagens em arquivos separados e ordenados no diretório **eggs_rows**.

Ao entrar no escopo da função, inicialmente é criado o diretório **eggs_rows** que é onde serão salvas todas as imagens representativas de cada linha de ovos. Depois, um laço itera sobre a lista de componentes conectados. Para cada componente, uma cópia da imagem rotacionada **rotated_image** é feita usando a função **copy** da biblioteca NumPy, garantindo que a imagem original não seja modificada. Esta cópia é armazenada na variável **row_image**. Em seguida, outro loop itera deste vez sobre cada retângulo do componente atual. Para cada retângulo, são extraídas as suas coordenadas x, y, w e h para que posteriormente cada componente da linha seja destacado na imagem. O código da função **save_row_images** está indicado no algoritmo 13.

Algorithm 13 Salvamento das imagens que representam as linhas de componentes

```
rows_dir = "eggs_rows"
if not os.path.exists(rows_dir):
    os.makedirs(rows_dir)
for i, component in enumerate(connected_components):
    row_image = np.copy(rotated_image)
    for rect in component:
        x, y, w, h = rect
        cv2.rectangle(row_image, (x, y), (x+w, y+h), (0, 255, 0), 2)
    cv2.imwrite(os.path.join(rows_dir, f"row_{i+1}.png"), row_image)
```

Voltando à execução da função **process_image**, o próximo passo é chamar a função auxiliar **save_component_images**. O propósito dela é semelhante ao da função anterior, porém, desta vez, cada componente de cada componente conectado será segmentado individualmente.

Inicialmente, é criado o diretório **eggs_results**, onde serão armazenados de forma ordenada todos os arquivos referentes a cada ovo segmentado. Em seguida, os componentes conectados são ordenados com base na coordenada X dos centróides dos retângulos. Após a ordenação, a função itera sobre cada componente, e para cada retângulo dentro do componente, a região correspondente ao ovo é recortada da imagem rotacionada. Esses ovos recortados são então salvos como arquivos no diretório **eggs_results**, com nomes que indicam sua posição na linha e na grade de ovos, permitindo uma referência fácil e organizada para cada ovo segmentado. O Código referente à esta função está no algoritmo 14

A este ponto do algoritmo, a solução já está completa e todos os ovos foram segmentados da imagem original e armazenados em ordem no diretório especificado ante-

Algorithm 14 Salvamento das imagens dos componentes segmentados

```

results_dir = "eggs_results"
if not os.path.exists(results_dir):
    os.makedirs(results_dir)

connected_components = [sorted(component, key=lambda rect:
calculate_centroid(rect)[0]) for component in connected_components]

for i, component in enumerate(connected_components):
    for j, rect in enumerate(component):
        x, y, w, h = rect
        egg_image = rotated_image[y:y+h, x:x+w]
        cv2.imwrite(os.path.join(results_dir,
f"row_{i+1}_egg_{j+1}.png"), egg_image)

```

riormente.

Após a execução desta função, o fluxo de execução do algoritmo volta a função **process_image** e executa uma última função chamada **plot_result**, que basicamente é responsável por imprimir uma versão da imagem original rotacionada com todos os componentes (ovos) indicados por um retângulo vermelho. Além disso, a imagem conta com arestas traçadas entre cada componente conectado, ou seja, todos os componentes que pertencem à mesma linha da grade, compartilham uma aresta.

Esta etapa do algoritmo foi essencial e necessária, agindo como um "debugger visual" que auxilia na identificação e compreensão de eventuais erros que aconteceram ao decorrer de todo o desenvolvimento do código. A exibição das imagens destacadas proporciona uma visão clara e detalhada do processo de segmentação, permitindo uma análise visual das regiões identificadas e facilitando a interpretação de possíveis problemas ou discrepâncias no resultado final. O código em algoritmo 15 é referente à função **plot_result**.

As imagens indicadas por figura 8, figura 9 e imagem 10 representam respectivamente: a identificação da primeira linha ou primeiro conjunto de componentes conectados, um componente segmentado e imagem final que indica a identificação de cada ovo e o agrupamento dos componentes por linhas.

5.2 Segunda abordagem

A segunda abordagem implementada neste projeto é bastante semelhante à primeira e difere apenas em alguns aspectos específicos. Assim como na primeira abordagem, o programa começa executando a função **process_image** com a imagem de entrada. No início do código da função, são chamadas as mesmas funções auxiliares

Algorithm 15 Exibição da imagem resultante das identificação dos ovos e das linhas

```

fig , ax = plt.subplots()
ax.imshow(cv2.cvtColor(rotated_image , cv2.COLOR_BGR2RGB))
for component in connected_components:
    for rect in component:
        x, y, w, h = rect
        rect_patch = Rectangle((x, y), w, h, fill=None,
                                edgecolor='red', linewidth=2)
        ax.add_patch(rect_patch)
    for rect1, rect2 in combinations(component, 2):
        centroid1 = calculate_centroid(rect1)
        centroid2 = calculate_centroid(rect2)
        ax.plot([centroid1[0], centroid2[0]], [centroid1[1],
                                                centroid2[1]], 'r-')
ax.set_title("Rotated Image with Connected Components and Edges")
ax.axis('off')
plt.show()

```

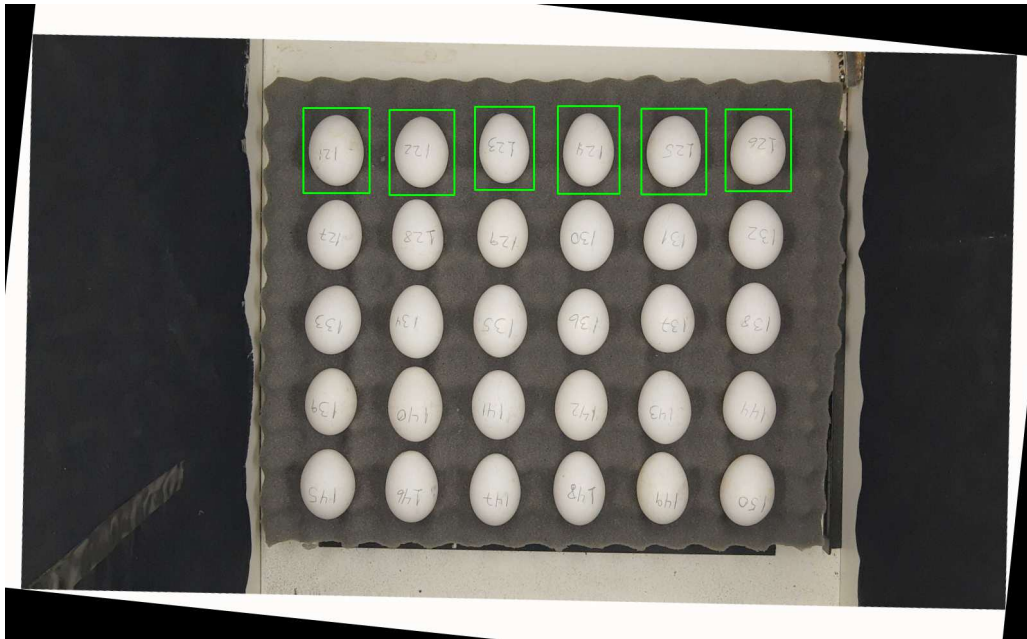


Figura 8 – Linha nº 1 da grade de ovos

find_inclination_angle e **rotate_image** para identificar o ângulo de rotação e realizar o redimensionamento da imagem por rotação. Esta etapa de redimensionamento segue exatamente o mesmo procedimento da primeira abordagem.

Após obter a imagem devidamente rotacionada, acontece a chamada da função **finddeggs** que recebe de entrada a imagem rotacionada. Inicialmente, são extraídas as dimensões de altura e largura da imagem, para que seja possível calcular a área total da imagem. Depois disso, o código segue para a etapa de pré-processamento da imagem, onde



Figura 9 – Ovo nº 1 da primeira linha de ovos

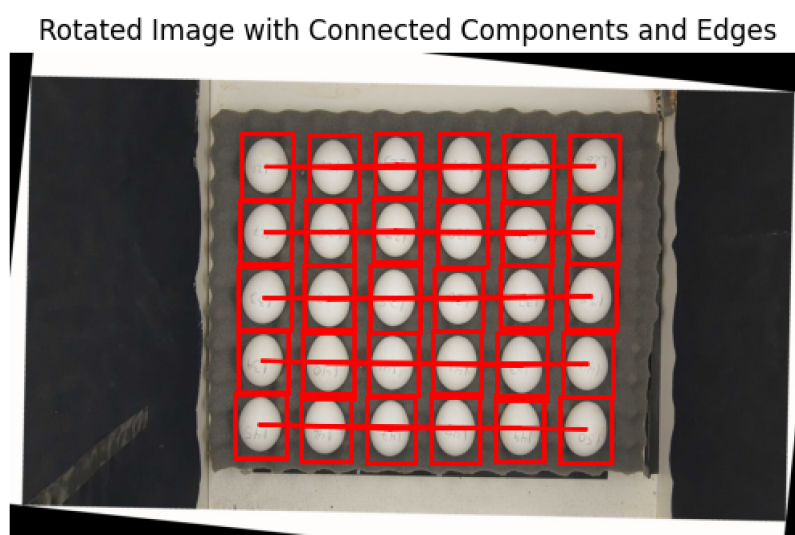


Figura 10 – Ovos identificados e conectados por arestas

ela é convertida para escala de cinza e também é realizada a aplicação do filtro Gaussiano, exatamente da mesma forma como é feito na primeira abordagem.

Ainda no pré-processamento, a imagem suavizada passa por uma técnica chamada de limiarização de Otsu, que determina automaticamente o valor do limiar com base na distribuição de intensidades dos pixels na imagem, tornando-a ideal para imagens com diferentes condições de iluminação. A técnica citada é aplicada a partir do fragmento 16:

Posteriormente, é realizada uma análise de componentes conectados na imagem binarizada através da função `connectedComponentsWithStats` para identificar regiões conectadas de pixels com valores iguais (ou maiores que zero). Essa função retorna várias

Algorithm 16 Limiarização

```
threshold = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY |
cv2.THRESH_OTSU)[1]
```

informações sobre os componentes conectados, como número de componentes, rótulos de cada pixel, estatísticas de cada componente e centróides de cada componente. A função recebe por padrão como parâmetros a imagem binarizada, o número de vizinhos considerados para determinar a conectividade entre os pixels, o tipo de rótulo de saída e por último o tipo de algoritmo utilizado para realizar a análise, que nesse caso não foi passado nenhum, deixando o padrão. Esta função é desempenhada pela linha de código 17:

Algorithm 17 Análise dos componentes conectados

```
analysis = cv2.connectedComponentsWithStats(threshold, 4, cv2.CV_32S)
```

A partir do resultado dessa função, são obtidos alguns valores que serão utilizados depois, como: `totalLabels` (número total de componentes encontrados na imagem), `values` (contém várias estatísticas extraídas da imagem) e os centroides de cada componente.

Após passar pela etapa de pré-processamento e extrair dados valiosos de cada componente e da imagem, é necessário adicioná-los à lista de ovos que foi criada no início da função. No entanto, é preciso filtrar todos os componentes identificados para que nenhum ruído ou componente indesejado seja adicionado à lista. Para fazer isso, foi utilizado um laço de repetição que percorre todos os componentes encontrados na imagem. O primeiro passo dentro do loop é calcular a área do componente, utilizando a variável **values** e a flag **CC_STAT_AREA** que representa a área do componente conectado ao rótulo *i*.

Agora que já temos a área do componente, o próximo passo é calcular dois valores importantes para realizar a filtragem dos componentes. O primeiro valor a ser calculado é chamado de **percArea**, que calcula a porcentagem da área do componente em relação à área total da imagem que foi obtida no início da função. Isso facilita a filtrar e diferenciar componentes pequenos ou muito grandes.

Posteriormente, é necessário calcular a razão de aspecto do componente, dividindo sua altura pela largura. Esse procedimento facilita a filtrar componentes que não tem uma forma semelhante à do ovo. Para calcular a razão de aspecto do componente, são utilizados: variável **values** e as flags **CC_STAT_HEIGHT** e **CC_STAT_WIDTH** que representam respectivamente altura e largura do componente que está sendo iterado.

Agora que obtivemos os dados essenciais para a análise do componente, é aplicado um critério para decidir se ele será adicionado à lista de ovos. Este critério é definido pela seguinte cláusula: se **percArea** estiver entre 0.3 e 1.0, e a razão de aspecto deste

componente estiver entre 0.4 e 1.6, significa que o componente atende aos requisitos para ser identificado como um ovo. Portanto, são extraídas suas coordenadas e dimensões, e, ao final, o componente é adicionado à lista de ovos. Ao término do loop, a lista de ovos estará completa, contendo todos os ovos da imagem que atenderam aos critérios estabelecidos. O código referente a este laço de repetição está no algoritmo 18:

Algorithm 18 Critério para decidir se o componente vai para a lista de ovos

```

for i in range(1, totalLabels):
    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)

```

Antes de partir para a etapa de ordenação, é necessário separar os ovos em grupos, permitindo agrupar ovos que estão na mesma linha.

Primeiro, define-se as variáveis **posVet** como 0 e **grupo** como 1 para iniciar o agrupamento. Em seguida, itera-se sobre os ovos. Se o ovo ainda não foi categorizado ainda, ele é marcado com o número de grupo atual e é verificado se outros ovos próximos também devem pertencer a esse mesmo grupo. Para isso, um loop interno percorre os ovos subsequentes e verifica se a diferença percentual entre suas posições na linha e a posição do ovo atual é inferior a 20%. Se sim, esses ovos também são marcados com o mesmo número de grupo.

Ao final dessa operação, o número de grupo é incrementado para a próxima categoria. Essa categorização é útil para agrupar componentes que estão próximos na linha, facilitando a próxima etapa do algoritmo. O fragmento de código 19 é responsável pelo agrupamento dos ovos de acordo com a abordagem explicada anteriormente:

A este ponto do algoritmo, todos os componentes já passaram pela filtragem, foram adicionados à lista de ovos e agrupados. O próximo passo é ordená-los seguindo o mesmo critério da primeira abordagem (começando do topo e esquerda e terminando abaixo na direita).

A ordenação é realizada em duas etapas, primeiro são ordenados pela linha e

Algorithm 19 Agrupamento dos ovos por linhas

```

posVet=0
grupo = 1
for i in range(posVet , 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]) < 20:
                ovos[k][0]=grupo
        grupo=grupo+1

```

em seguida pela coluna dentro de cada grupo. O primeiro bloco de código implementa a ordenação pela linha, onde dois loops for aninhados são usados para comparar cada ovo com todos os outros e trocar suas posições se necessário para garantir que os ovos estejam ordenados de cima para baixo na imagem. Em seguida, o segundo bloco de código implementa a ordenação pela coluna, onde novamente são usados dois loops for aninhados para comparar cada ovo com todos os outros no mesmo grupo e trocar suas posições se necessário para garantir que os ovos estejam ordenados da esquerda para a direita na imagem. Finalmente, a lista ordenada de ovos é retornada para o próximo estágio do processamento de imagem. O algoritmo 20 realiza o procedimento de ordenação:

Algorithm 20 Ordenação dos ovos pela linha e pela coluna

```

#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

```

Após a função **findeggs** retornar a lista ordenada com todos os ovos identificados, o fluxo de execução do algoritmo retorna à função **process_image**, que realiza o procedimento final. Neste passo, é criado o diretório **final_eggs**, onde todos os ovos são

armazenados obedecendo à ordem estabelecida. Cada ovo é nomeado de acordo com sua posição na grade, recebendo o nome **egg_n.jpg**, onde "n" representa sua posição. O código referente à esta etapa está no fragmento de código 21

Algorithm 21 Criação do diretório com os ovos armazenados em ordem

```
if not os.path.exists('final_eggs'):
    os.makedirs('final_eggs')

for i, ovo in enumerate(ovos):
    x, y, w, h = ovo[2], ovo[3], ovo[4], ovo[5]
    egg_img = rotated_image[y:y+h, x:x+w]
    cv2.imwrite(f'final_eggs/egg_{i+1}.jpg', egg_img)
```

Ao final da função **process_image**, foi incluído um trecho de código que plota a imagem rotacionada, destacando todos os ovos identificados com retângulos vermelhos. Essa visualização, assim como na primeira abordagem, desempenhou um papel importante como um "debugger visual", auxiliando na detecção de possíveis problemas no código. O código 22 faz referência à esta etapa.

Algorithm 22 Exibição da imagem final com a identificação dos ovos

```
fig, ax = plt.subplots(1)
ax.imshow(cv2.cvtColor(rotated_image, cv2.COLOR_BGR2RGB))

for ovo in ovos:
    rect = Rectangle((ovo[2], ovo[3]), ovo[4], ovo[5],
                     linewidth=1, edgecolor='r', facecolor='none')
    ax.add_patch(rect)

plt.show()
```

Finalmente, o algoritmo é encerrado com todos os ovos armazenados de forma separada e ordenada no diretório **final_eggs** e a plotagem final da imagem original rotacionada com a identificação de cada ovo.

Os resultados alcançados em ambas as abordagens foram notáveis, com uma taxa de acerto de 100% na identificação e ordenação dos ovos para os casos teste. Além disso, é importante ressaltar que ambos os algoritmos foram extensivamente testados e validados utilizando uma diversidade de casos de teste fornecidos pelo professor Maurício Cunha Escarpinati. Esses casos abrangem uma gama de cenários e situações que permitem uma avaliação abrangente e robusta do desempenho dos algoritmos em condições diversas.

As imagens indicadas por figura 11 e figura 12 representam respectivamente: a imagem de um ovo segmentado e a imagem que exibida ao final do algoritmo, contendo todos os ovos que foram identificados no processo.



Figura 11 – Ovo nº 1 segmentado da imagem

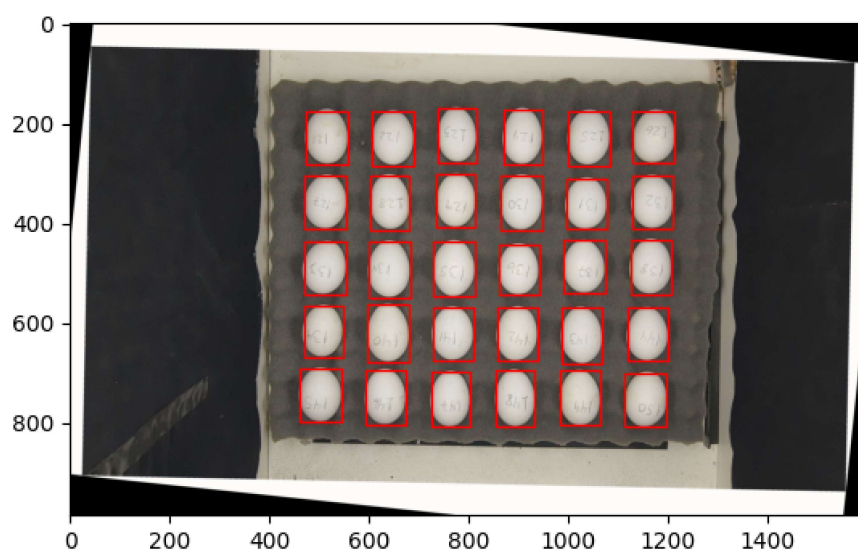


Figura 12 – Ovos identificados

6 Conclusão

Uma das principais contribuições deste projeto foi a criação de um algoritmo capaz de segmentar os ovos de forma precisa e automática, fornecendo uma solução viável para o desafio proposto. A conclusão deste projeto serviu de complemento para o projeto relacionado da pós-graduação. Através do desenvolvimento geral do trabalho, foi possível perceber a importância de cada etapa do projeto em si. A implementação bem-sucedida de etapas como o pré-processamento, detecção de bordas, identificação de ovos e segmentação bem como o desenvolvimento de uma documentação detalhada do código, destacou a importância de uma abordagem sistemática e abrangente para resolver problemas complexos.

O desenvolvimento deste projeto proporcionou uma compreensão aprofundada dos desafios envolvidos em todas as etapas do processamento digital de imagens. Ao longo do processo, foram exploradas diferentes abordagens e técnicas de processamento de imagens, desde o pré-processamento das imagens até a segmentação final dos ovos. A utilização de bibliotecas como OpenCV, NumPy e Matplotlib foi fundamental para a implementação eficiente do algoritmo, facilitando a manipulação dos dados de forma eficiente.

Foi possível adquirir conhecimentos não apenas através da implementação do algoritmo, mas também por meio da exploração e leitura das documentações das funções e bibliotecas utilizadas. Essa imersão na documentação foi fundamental para compreender os princípios e técnicas subjacentes e as melhores práticas de programação, ampliando significativamente minha compreensão do processo de processamento de imagens. Este conteúdo aplicado juntamente com os desafios encontrados ao decorrer do trabalho, foram uma valiosa fonte de aprendizado que proporcionou insights e habilidades que certamente serão aplicáveis em futuros projetos e no avanço da minha carreira profissional.

Embora os resultados alcançados pelo algoritmo desenvolvido serem satisfatórios, ainda há margem para melhorias e otimizações futuras. Uma possível abordagem para aprimorar o desempenho do algoritmo seria a integração de técnicas de aprendizado de máquina. Ao incorporar modelos de machine learning, é possível aperfeiçoar a precisão da segmentação e tornar o algoritmo mais adaptável a diversas condições e variações nas imagens dos ovos, contribuindo para sua robustez e eficácia em diferentes contextos.

Referências

- ALBANEZ MARCOS AURÉLIO BATISTA, S. F. d. S. Daniela de O. O que é segmentação de imagens? Universidade Federal de Goiás, Campus de Catalão, 2016. Disponível em: <https://files.cercomp.ufg.br/weby/up/615/o/O_que_e_segmentacao_de_imagens.pdf?1481557289#:~:text=O%20procedimento%20de%20segmenta%C3%A7%C3%A3o%20de,%2C%20distinguindo%20das%20das%20outras.> Citado na página 17.
- BORGES, C. H. de P. **Ajuste polinomial para refinamento do resultado de segmentação de imagem para cálculo de volume de ovos.** 2023. <<https://repositorio.ufu.br/handle/123456789/38633>>. Citado na página 8.
- DORINI, L. E. B.; ROCHA, A. de R. Detecção de círculos em imagens através da transformada de hough. 2011. Citado na página 13.
- GONZALEZ, R. C.; WOODS, R. E. **Processamento de imagens digitais.** [S.l.]: Editora Blucher, 2000. Citado na página 11.
- HEISE, G. S. L. M. Detecção de Bordas e Transformações Morfológicas em Imagens com OpenCV. 2020. Disponível em: <<https://medium.com/turing-talks/detec%C3%A7%C3%A3o-de-bordas-e-transforma%C3%A7%C3%B5es-morfol%C3%B3gicas-em-imagens-com-opencv-8aecf8c8ba2f>>. Citado 3 vezes nas páginas 15, 16 e 19.
- MATOSAK, B.; MEDEIROS, N.; RODRIGUES, F. Filtro gaussiano adaptativo implementado em ambiente julia. Disponível em: <https://www.cartografia.org.br/cbc/trabalhos/4/595/CT04-119_1506775015.pdf>. Citado 2 vezes nas páginas 14 e 15.
- NARUSHIN. Egg geometry calculation using the measurements of length and breadth. 2005. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0032579119445537>>. Citado na página 8.
- NOGUEIRA, R. C. de A. **Análise de Conversão de Imagem Colorida para Tons de Cinza Via Contraste Percebido.** 2016. Disponível em: <<https://www.cin.ufpe.br/~tg/2016-1/rcan.pdf>>. Citado na página 13.
- OPEN Source Computer Vision. Disponível em: <<https://docs.opencv.org/4.x/index.html>>. Citado na página 25.
- OPENCV.ORG. **Image Processing.** [S.l.]. Accessed: 2024-05-13. Disponível em: <https://docs.opencv.org/4.x/d7/dbd/group__imgproc.html>. Citado na página 8.
- RAVINDRA, S. How Convolutional Neural Networks Accomplish Image Recognition? 2017. Disponível em: <<https://www.kdnuggets.com/2017/08/convolutional-neural-networks-image-recognition.html>>. Citado na página 18.
- SAIFULLAH, S.; MEHRIDDINOVICH, R. I.; TOLENTINO, L. K. Chicken egg detection based-on image processing concept: A review. 2021. Disponível em: <<http://jurnal.upnyk.ac.id/index.php/cip/article/view/6129/3957>>. Citado 2 vezes nas páginas 8 e 19.

SHARDA, A. **Image Filters: Gaussian Blur**. 2021. Disponível em: <<https://aryamansharda.medium.com/image-filters-gaussian-blur-eb36db6781b1>>. Citado na página 16.

TOMÉ, M. P. S. **Um novo método para auferição de características de cascas de ovos utilizando processamento de imagens**. 82 f. Dissertação (Mestrado em Ciência da Computação) — Faculdade de Computação da Universidade Federal de Uberlândia, Uberlândia-MG, 2004. Citado na página 21.

ULASZEWSKI, M.; JANOWSKI, R.; JANOWSKI, A. **Application of computer vision to egg detection on a production line in real time**. 2021. <https://ddd.uab.cat/pub/elcvia/elcvia_a2021v20n2/elcvia_a2021v20n2p113.pdf>. Citado 2 vezes nas páginas 8 e 19.

WANGENHEIM, A. V. Introdução a visão computacional: Técnicas de segmentação simples de imagens. **Departamento de Informática e Estatística-INE/CTC/UFSC. The Ciclops Project, 2007b**. Disponível em:< <http://www.inf.ufsc.br/~visao/threshold.html>>. Data de acesso, 2007. Citado na página 15.