



UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

PROBLEMA DE DETECÇÃO DE FUROS EM OBJETOS

PROCESSAMENTO DE IMAGENS - T01 - 2022.2

Bruno Henrique de Jesus Silva
Gustavo de Farias Souza Almeida
João Pedro da Silva Santos
Matheus Lima Pinheiro



Departamento de Computação/UFS

São Cristóvão – Sergipe 2023

1 Introdução

O projeto foi baseado em uma simulação feita pela empresa *GenericStuff* para trabalhar em um sistema de inspeção de objetos. Este sistema tem como objetivo, a partir de uma imagem, identificar quantas objetos estão dispostos e quais deles possuem furos em seu interior. Com base nisso, deve-se implementar um algoritmo que informe, para cada imagem capturada pelo sistema, o total de objetos na imagem, além da quantidade de objetos com furos e sem furos que estão contidos na imagem de entrada. Para contextualizar, foi considerado que os objetos passam sobre uma esteira rolante e uma câmera captura as imagens binárias (preto e/ou branco). Além disso, os objetos não tocam entre si, não estão sobrepostos e podem ter qualquer forma ou tamanho.

Com isso, a saída do programa deverá ser o retorno do total de objetos na imagem, além da quantidade de objetos com furos e sem furos a imagem possui, de modo que o funcionamento seja dado pelo console.

2 Implementação

Para a resolução do problema, utilizamos o algoritmo *flood-fill*. Em suma, uma imagem de duas dimensões, a partir da localização de um pixel na tela e uma cor, ele substitui a cor do pixel fornecido e todos os pixels adjacentes/vizinhos (que possuem a mesma cor) por uma outra cor fornecida. Assim, ele foi utilizado para efetuar a contagem dos objetos (com e sem furos), além de ser utilizado para contagem dos buracos existentes na imagem.

2.1 Conceitos utilizados

Antes da implementação propriamente dita, se faz necessária a explicação de alguns conceitos que tornaram possível o entendimento e a implementação do algoritmo *flood-fill*. A maior parte destes conceitos foi discutida ao longo da disciplina de “Processamento de Imagens” ou de outras disciplinas, como “Grafos”. O livro “*Machine Vision*”, de Jain et al. (1995), serviu de auxílio para a parte teórica.

Os conceitos são:

- **Imagem binária:** uma das formas mais antigas e utilizadas formas de se representar digitalmente uma imagem. Diferente do tipo mais comum, no qual existem 256 níveis de cinza, imagens binárias utilizam apenas o valor 1, para objetos, e 0, para o fundo. O algoritmo desenvolvido utiliza o formato PBM ASCII

(PGM do tipo 1) para este tipo de imagem. É possível adaptar o algoritmo para aceitar imagens não binárias, com um processo de *thresholding* (limiarização).

- **Vizinhança:** por definição, dois pixel são “vizinhos” caso eles possuam pelo menos uma fronteira (isto é, um lado) em comum, no caso da vizinhança 4, ou pelo menos um canto, no caso da vizinhança 8. No algoritmo, foi definida vizinhança 8, para os objetos, e 4, para o fundo.

- **Caminho:** um caminho é uma sequência de pixels, na qual cada pixel deve ser vizinho tanto do seu sucessor quanto do seu antecessor, que conectam dois pixels de uma imagem.

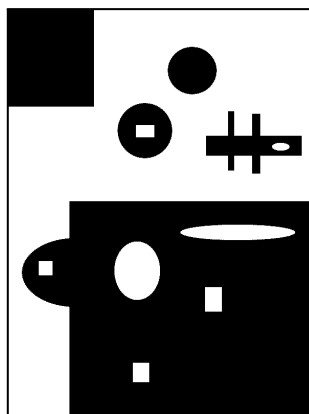
- **Conexão:** dois pixels são ditos conectados quando existe um caminho composto de apenas pixels pertencentes ao objeto entre eles.

- **Componentes conexos:** conjunto de pixels no qual todos os seus componentes são conectados entre si. A base do algoritmo *flood-fill*, utilizado no projeto, é basicamente a substituição da cor de todos os pixels em um mesmo componente conexo.

- **Segmentação de imagens:** basicamente, é a divisão de uma imagem em partes ou regiões menores para facilitar o tratamento e a utilização dessas imagens. O *thresholding* é um dos processos mais simples de segmentação, mas existem outros mais complexos. Neste projeto, a parte de segmentação está relacionada à detecção dos objetos como entidades separadas entre si.

2.2 Leitura da Imagem

Foi criada uma classe chamada *Imagem_PBM*, a qual armazena um arquivo de imagem do tipo .pbm, como a mostrada a seguir, de modo que os pixels sejam salvos em uma matriz de inteiros (dimensão: altura x largura).



Exemplo de imagem alvo

A imagem, para ser lida, precisa estar na pasta *Testes* do projeto, seu nome precisa ser passado via console, onde será carregada dentro do método *carregaimagem()* que fica no arquivo *main.py*. Este método faz a leitura das linhas iniciais e armazena informações como cabeçalho, formato, largura e altura, além de efetuar duas estruturas de repetição para ler cada valor do arquivo .pbm e armazenar na matriz.

2.3 Contagem dos objetos

Foi utilizado o algoritmo flood-fill para “rotular” os objetos nas imagens binárias fornecidas. A ideia principal do algoritmo é atuar de forma recursiva, de modo que primeiro é encontrado um pixel que faz parte de um objeto (valor = 1) e substituímos sua cor atual, para então analisar os 8 pontos ao seu redor (considerando também a diagonal vizinha), já que a vizinha dos objetos é 8, até que todos aqueles referentes ao mesmo objeto sejam também preenchidos com o mesmo rótulo, que são definidos por números a partir de “2”.

Nesta parte, foi utilizada uma estrutura de pilha para manter um registro de quais pontos ainda precisam ser analisados. Ao final desta etapa, cada objeto apresenta seus pixels rotulados com um valor específico daquela figura, ou seja, diferente para cada objeto. O resultado pode ser exemplificado da seguinte forma:

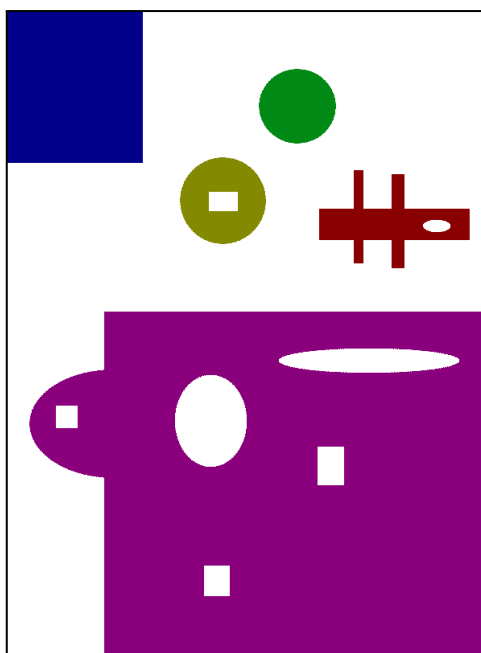


Imagem ilustrativa do funcionamento do *Flood-fill* para a contagem de objetos

2.4 Contagem de buracos

A seguir, foi utilizado um método para adição de *padding* na imagem original para que pudéssemos separar o fundo dos buracos durante o processo e, assim, evitar que os buracos fossem preenchidos durante a execução do *flood-fill*.

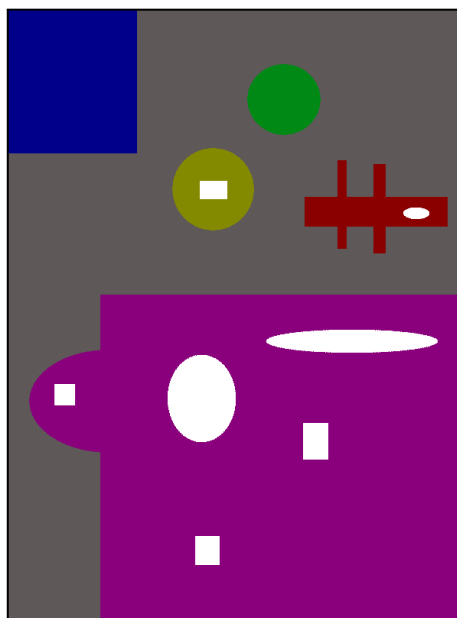


Imagem ilustrativa do funcionamento do *Flood-fill* para o preenchimento do fundo

Após o reconhecimento e marcação do fundo, foi feita a contagem dos buracos através de outra aplicação do *flood-fill* marcando os pixels alvo brancos (representados pelo 0) com a *label* “-2”, considerando uma vizinhança 4.

Durante essa aplicação será identificado a qual objeto aquele buraco pertence, isto é, é verificado qual o primeiro pixel não branco da vizinhança do buraco, e retornado em um *array* o resultado, de modo que seja possível observar quantos buracos existem em cada figura.

2.5 Concessões

Felizmente, não foi necessária a realização de concessões em relação ao escopo inicial do projeto, isto é, o algoritmo desenvolvido é capaz de reconhecer objetos e buracos de quaisquer tamanhos e formatos. Isso ocorre devido a natureza do *flood-fill*, que funciona com base na conexão entre os pixels. Desta forma, a aplicação não é limitada apenas a figuras compostas por figuras geométricas simples, como quadrados e retângulos, o que poderia ocorrer com a utilização de métodos que envolvessem máscaras.

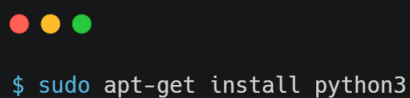
3 Instruções de execução em SO baseado em Linux

Para executar o código do nosso projeto em um sistema baseado em Linux primeiro é necessário garantir que o *python* esteja instalado, podemos fazer isso da seguinte forma:



```
$ python --version  
  
// ou  
  
$ python3 --version
```

Caso exista uma versão do *python* instalada irá exibi-la, caso contrário será retornada uma mensagem de erro significando que será necessário fazer a instalação do *python* na máquina. A instalação pode executando o seguinte comando no terminal:



```
$ sudo apt-get install python3
```

Após a instalação a máquina está pronta para executar o código do projeto, bastando entrar no diretório em que ele está e digitar o comando a seguir para executá-lo:



```
$ python main.py
```

Com o programa em execução será solicitado para o usuário informar o nome de um arquivo de imagem para ser processado pelo nosso algoritmo, vale lembrar que

essa imagem deve estar dentro do diretório *ProjetoPDI/Testes* para que possa ser utilizada como parâmetro para o programa. Segue um exemplo de execução com a imagem 04.pbm:

```
Informe o nome do arquivo (sem a extensão): 04
Imagem Testes/04.pbm carregada com sucesso!

Esperando resultados...

-----
=> Total: 5 objeto(s)
-----
> 0 objeto '1' contém 0 buraco(s)
> 0 objeto '2' contém 0 buraco(s)
> 0 objeto '3' contém 1 buraco(s)
> 0 objeto '4' contém 1 buraco(s)
> 0 objeto '5' contém 5 buraco(s)
-----
=> Objetos SEM buraco: 2
=> Objetos COM buraco: 3
-----

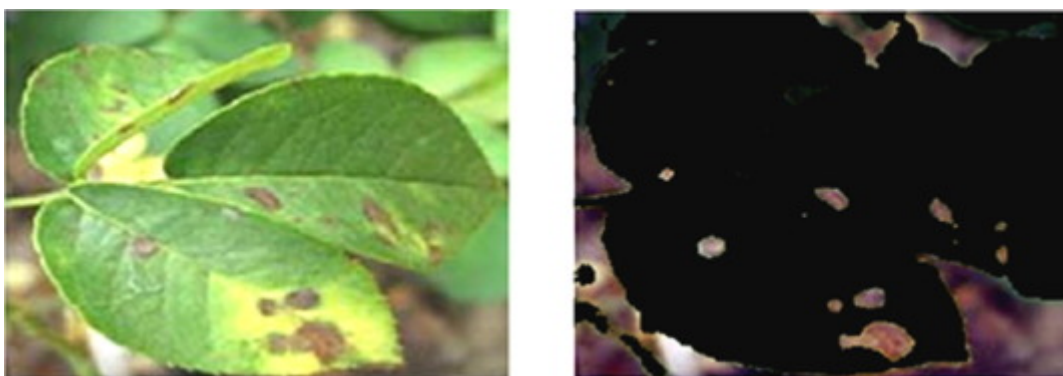
Execução finalizada!
```

4 Potencial de uso do algoritmo desenvolvido

A detecção de furos é necessária na indústria alimentícia, pois afeta a segurança do consumidor. Se uma lata ou embalagem houver algum furo, pode haver contaminação química ou microbiana no produto, além de causar vazamentos na hora de embalar, uma contaminação no alimento pode resultar em doenças e problemas de saúde no consumidor. Além disso, mesmo que não ocorra contaminação, esses furos podem causar perda de sabor, textura e nutrientes do alimento, gerando uma insatisfação do cliente com aquele produto. ([HELLMEISTER, 2001](#))

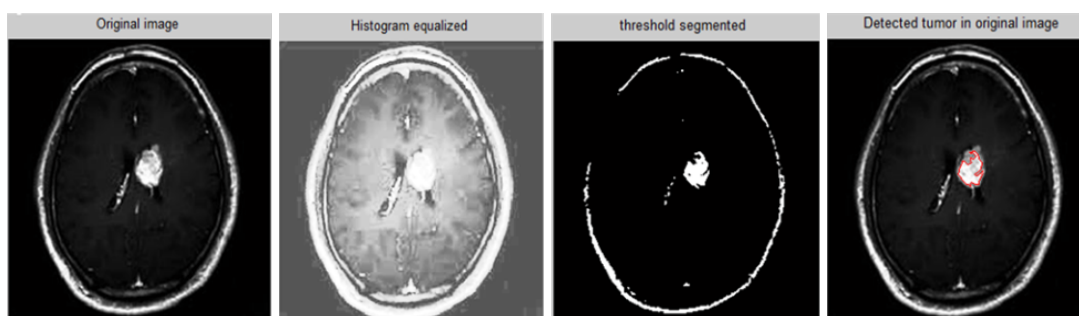
Nesse contexto, o algoritmo pode ser uma ótima alternativa para detectar furos e vazamentos de embalagens de alimentos. Após passar as embalagens por uma esteira e câmera de captura de imagens binárias, o algoritmo pode ser aplicado nessas imagens detectando furos nos objetos, a utilização do algoritmo proporciona um alto grau de precisão e eficiência no processo de inspeção, uma vez que elimina a necessidade de inspeção manual, que pode ser cansativa e propensa a erros, garantindo a segurança e qualidade de embalagens e produtos.

Outro uso possível para um algoritmo de detecção de buracos em objetos é o diagnóstico de doenças em plantas. Segundo Singh e Misra (2017), a presença de doenças em uma planta, como a mostrada na imagem a seguir, podem ser detectadas com a verificação da presença de manchas ou buracos nas folhas da mesma. Desta forma, o algoritmo poderia ser utilizado em imagens já tratadas, após a aplicação de *thresholding*, para auxiliar este tipo de processo.



Fonte: Singh e Misra (2017)

Além disso, também foi pesquisada a possibilidade da utilização de técnicas de processamento de imagens para o diagnóstico de doenças em seres humanos. Um dos estudos encontrados foi o proposto por Isselmou et al. (2016), que utiliza algoritmos de *thresholding* e operações morfológicas para a detecção, extração e estudo de tumores cerebrais em imagens de ressonância magnética. A partir da leitura do artigo, foi verificado que o *flood-fill* implementado pelo grupo pode ser considerado uma ferramenta mais simples para propósitos similares. As imagens a seguir mostram o processo de detecção de tumores já mencionado:



Fonte: Isselmou et al. (2016)

5 Adaptações necessárias para funcionar

Para uma implementação próxima ao potencial de uso do nosso algoritmo, como descrito no tópico anterior, será necessário realizar algumas adaptações. Entre as adaptações necessárias estão: reconhecer imagens RGB, garantir que todas as imagens terão o mesmo padrão de qualidade (ambiente luminoso, posicionamento do objeto de interesse, etc) e eliminar possíveis ruídos na imagem. Além disso, para a implantação na indústria alimentícia, seria necessário fazer parte do processo de fabricação acoplado a esteiras para que faça o reconhecimento um a um dos objetos de interesse.

6 Relação com o conceito de ética

A principal discussão ética observada pelo grupo, dadas as possibilidades de uso encontradas, foi a questão da obtenção de diagnósticos incorretos de doenças, que é algo já ocorre com softwares mais complexos, aqueles que utilizam aprendizagem de máquina e redes neurais, e fica ainda pior em relação a um algoritmo tão simples quanto o apresentado no presente trabalho, além de pôr em risco a vida de pacientes. Desta forma, mesmo com a utilização de sistemas de diagnóstico, a presença humana em tais processos ainda se mostra essencial.

Referências

HELLMEISTER, Fiorella. Detecção de vazamento em embalagens .**CETEA Informativo**, Campinas, v. 13, n. 3, p. 7-9, jul./ago./set. 2001. Disponível em: https://ital.agricultura.sp.gov.br/arquivos/cetea/informativo/v13n3/v13n3_artigo3.pdf. Acesso em: 02 Maio 2023.

ISSELMOU, Abd El Kader; ZHANG, Shuai; XU, Guizhi. **A Novel Approach for Brain Tumor Detection Using MRI Images**. Journal of Biomedical Science and Engineering, [s. l.], v. 9, ed. 10B, p. 44-52, 2016.

JAIN, Ramesh C.; KASTURI, Rangacher; SCHUNCK, Brian G. **Machine Vision**. 1. ed. [S. l.]: McGraw-Hill Science/Engineering/Math, 1995. 549 p.

SINGH, Vijai; MISRA, A.K. **Detection of plant leaf diseases using image segmentation and soft computing techniques**. Information Processing in Agriculture, [s. l.], v. 4, ed. 1, p. 41-49, 2017.