
Universidade Federal de Sergipe
Departamento de Computação

# Relatório do Projeto de Processamento de Imagens

**Turma:** T02 2021.2
**Alunos:** Allan Juan, Átila Sousa e Jeferson da Silva
**Data:** 27/05/2022

# 1. Introdução

Antes de mais nada, vamos recapitular a especificação do problema a ser resolvido. Fomos incumbidos com a tarefa de desenvolver um programa atendendo a especificação a seguir:

*Dada uma imagem binária contendo objetos que podem ou não ter furos, deve-se informar o total de objetos na imagem, bem como quantos deles têm furos e quantos não têm.*

Entende-se por “objeto sem furos” uma região contínua de pixels pretos na imagem, enquanto “objetos com furos” são regiões contínuas de pixels pretos contendo um ou mais pixels brancos em seu interior.

A entrada será dada no formato *PBM ASCII*, uma imagem binária composta por **0**’s e **1**’s, onde o **0** representa a cor branca e o **1** a cor preta.

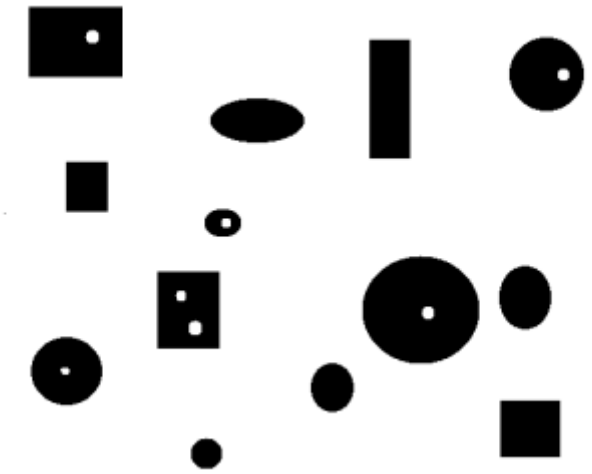

Figura 1 – Exemplo de Imagem de Entrada

# 2. Metodologia

## 2.1 Linguagem de programação escolhida

Implementamos a solução na linguagem de programação Python – a escolha se deve à proficiência dos discentes com essa tecnologia, e pela ergonomia que sua interface oferece para a resolução desse problema – em Python, é possível manipular matrizes com relativa facilidade, já que ela dispõe de recursos como listas dinâmicas e compreensão de listas. O único ônus está na sua limitação no tamanho de pilha, que atrapalha na implementação de funções recursivas quando a entrada é muito extensa.

## 2.2 Algoritmo

A implementação consiste em uma combinação de *Flood Fill* e pré-processamento morfológico através do algoritmo de preenchimento de regiões *Region Fill*. Ou seja, podemos dividir o algoritmo em duas partes ou sub-algoritmos:

1. ***Region Fill:*** preenche todos os buracos nos objetos, de modo que não reste nenhum objeto com furos.
2. ***Flood Fill:*** conta o número de objetos na imagem, identifica suas coordenadas e distingue os que têm furos dos que não têm.

### 2.2.1 Region Fill

Primeiro, é feita uma varredura na matriz, da esquerda para a direita e de cima para baixo, em busca do primeiro pixel preto (valor 1). Esse pixel é chamado de "semente".

Em seguida, é criada uma matriz nula (apenas zeros) com as mesmas dimensões da imagem. O pixel com as coordenadas do pixel semente tem o seu valor alterado para 1 na matriz recém-criada.

Então, essa matriz é dilatada usando um elemento estruturante em formato de cruz, e em seguida é aplicado o operador `AND` com o complemento da imagem original (inversa da imagem). Esse processo se repete em loop até que ele deixe de surtir efeito sobre a matriz, isto é, até que seu resultado seja igual ao resultado anterior. O procedimento completo é descrito por:

$$X_k = (X_{k-1} \oplus B) \cap A^c \qquad k = 1, 2, ...$$

### 2.2.2 Flood Fill

Depois da finalização do algoritmo de preenchimento de regiões, todos os buracos nos objetos da imagem estarão preenchidos. Então é aplicado o algoritmo *Flood Fill* na imagem sem buracos.

Para o *Flood Fill*, a ideia implementada consiste em, na varredura de pixels, toda vez que é encontrado um pixel de valor **1**, ocorre a detecção de um novo objeto. Então é determinada toda a área conectada a esse pixel encontrado, ou seja, todos os pixels de valor **1** conectados à ele.

À medida que o algoritmo detecta a área conectada, esses pixels também são marcados em uma tabela de forma que a varredura não volte a passar por eles, pois nesse caso seria detectado erroneamente um novo objeto.

Enquanto o *Flood Fill* detecta os objetos, foi implementada uma variação para a detecção de furos nos objetos utilizando a seguinte lógica: se algum pixel do objeto que está sendo preenchido pelo *Flood Fill* tiver valor diferente da imagem original (com buracos), então esse pixel era um buraco que foi preenchido pelo *Region Fill*, logo esse objeto possui furos.

# 3. Instruções de compilação e execução

A compilação e execução do código é bastante simples e feita em somente 1 linha. Basta executar no terminal o comando *python*, seguido do caminho do arquivo contendo o código principal e do caminho da imagem a ser processada. Os resultados serão exibidos no próprio terminal. É necessário ter o *Python 3+* instalado. Exemplo de execução:

*python detect-objects.py images/objects_1.pbm*

# 4. Resultados

Os resultados obtidos foram satisfatórios e robustos, visto que foram testadas várias imagens com diversos formatos e quantidades de objetos e furos. O tempo de execução também foi aceitável, já que o algoritmo de preenchimento de regiões é relativamente lento por aplicar repetidamente dilatações, operações lógicas e comparações. A seguir serão apresentadas as imagens testadas e a tabela dos seus respectivos resultados.

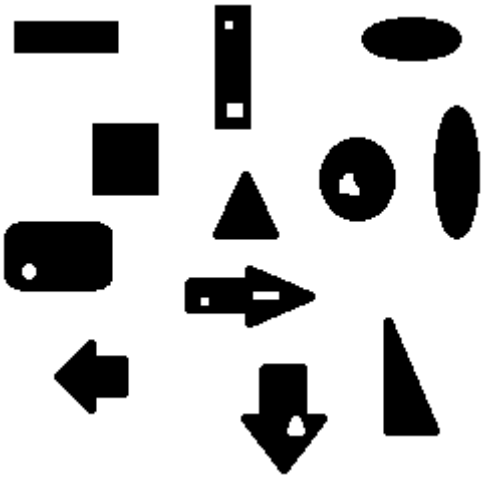

Figura 2 – Imagem 1

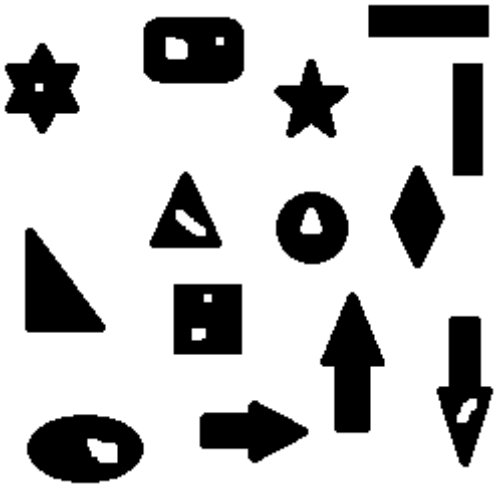

Figura 3 – Imagem 2

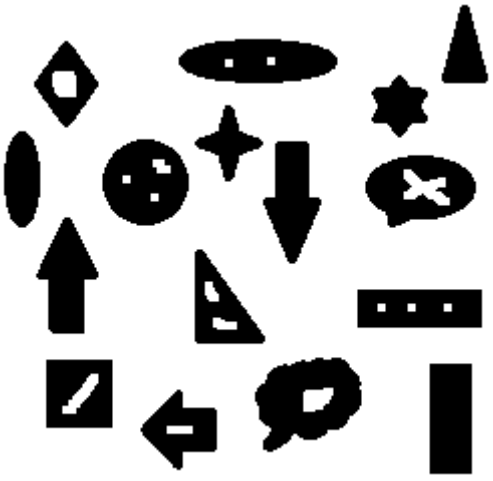

Figura 4 – Imagem 3

| | Total de objetos encontrados | Total de objetos com furos | Total de objetos sem furos | Tempo gasto (s) |
|---|---|---|---|---|
| Imagem 1 | **12** | **5** | **7** | **56.770** |
| Imagem 2 | **14** | **7** | **7** | **70.662** |
| Imagem 3 | **16** | **9** | **7** | **60.229** |

Tabela 1 - Resultados

# 5. Aplicações

Sendo o produto deste trabalho um algoritmo razoavelmente genérico, feitas as modificações necessárias, ele pode ser reutilizado em diferentes contextos.

Uma possível aplicação seria um software para controle de qualidade industrial. Imagine uma linha de produção em que os itens devem ter uma pintura uniforme e monocromática – itens com falhas na pintura são considerados defeituosos e devem receber um tratamento especial. Esse é um caso especial do problema resolvido pelo algoritmo, em que as falhas na pintura são os “furos”. Assim, o programa poderia ser utilizado para discernir os objetos corretos dos defeituosos. Para tal, seria preciso apenas deixar a imagem capturada em conformidade com a entrada esperada pelo programa, que é uma imagem binária. Uma possível abordagem seria utilizar um filtro de limiarização, mapeando as cores da imagem bruta para preto/branco de acordo com níveis de intensidade pré-definidos.

Analogamente, podemos generalizar e dizer que esse algoritmo serve para qualquer problema que envolva identificar “descontinuidades” em um objeto, ou seja, ele pode verificar se um objeto tem furos, se um objeto tem o número correto de furos, se o número de objetos com/sem furos está de acordo com o esperado, enfim. Dito isso, ele pode ter diversas aplicações.

# 6. Considerações Éticas

O que podemos dizer a respeito da ética em um programa como esse? Ele é ético ou antiético? Sua existência representa algum risco à sociedade?

Ora, uma vez que o programa é extremamente genérico, não pode haver nada de inerentemente controverso sobre ele, uma vez que não possui um fim em si mesmo. Assim, não é possível dar juízo de valor quanto à sua existência propriamente dita – ele é apenas tão ético ou antiético quanto a sua aplicação final.

Dito isso, vamos pensá-lo no contexto de suas aplicações. Já que citamos exemplos de aplicação em linhas de produção, sigamos por esse fio. Se o bem produzido nessa esteira não for de natureza controversa, então não há com que se preocupar. Mas e se for? E se a linha de produção em questão estiver fabricando armas ou entorpecentes? Certamente, cabe uma discussão mais complexa.

Será que a possibilidade desse tipo de aplicação justifica a destruição ou proibição do algoritmo? Cremos que não – sendo ele genérico e aplicável em diferentes contextos, não sendo antiético em si mesmo, pensamos que sua existência não represente um problema. Caso contrário, estaríamos combatendo tecnologias como a pólvora, o avião, o telefone e o computador, que, apesar de serem úteis para a manutenção do bem estar social, também são empregadas em fins escusos.

Porém, nos limitamos a dizer que, sendo o algoritmo genérico e aplicável para qualquer fim, não pode ser antiético em si mesmo, e portanto sua existência não apresenta, inerentemente, um problema. Caso contrário, estaríamos combatendo tecnologias como a pólvora, o avião, o telefone e o computador, que, apesar de serem muito úteis para a manutenção do bem estar social, também são empregadas em fins escusos.

Ou seja – acreditamos que esse software está mais suscetível a agregar do que a agredir a sociedade. Não obstante, devemos permanecer vigilantes a aplicações maliciosas que possam surgir, prontamente combatendo-as.

# 7. Conclusão

Neste trabalho, implementamos um algoritmo para detecção de objetos em imagens binárias usando técnicas de processamento de imagens aprendidas durante o curso.

Uma vez implementado o algoritmo, refletimos sobre algumas aplicações que ele poderia ter no mundo real, principalmente no contexto de software de controle de qualidade em linhas de produção. A partir dessas aplicações, debruçamo-nos sobre as implicações éticas desse programa, concluindo que não há nada de inerentemente mau sobre ele.

Dessa forma, o trabalho representou um desafio técnico muito interessante que serviu para aplicar e consolidar os conhecimentos adquiridos durante o curso.

# 8. Referências

GONZALES, R. WOODS, R. *Digital Image Processing*. 2th ed. New Jersey: Prentice Hall, 2001.

Slides da professora Beatriz Trinchão Andrade do Departamento de Computação da Universidade Federal de Sergipe.

*Flood Fill*. Wikipedia. Disponível em: https://en.wikipedia.org/wiki/Flood_fill.