

# GRABCUT PARA SEGMENTAÇÃO DE OBJETOS

1<sup>st</sup> João Eduardo Santo Farias

Departamento de Engenharia da Computação  
Instituto Federal de São Paulo  
Birigui, SP, Brasil  
j.farias@aluno.ifsp.edu.br

2<sup>nd</sup> Luiz Henrique Botega Cervantes

Departamento de Engenharia da Computação  
Instituto Federal de São Paulo  
Birigui, SP, Brasil  
luiz.cervantes@aluno.ifsp.edu.br

**Resumo—**O *GrabCut* é um algoritmo projetado por Carsten Rother, Vladimir Kolmogorov e Andrew Blake da Microsoft Research Cambridge do Reino Unido. O termo apareceu pela primeira vez no artigo "GrabCut": extração interativa de primeiro plano usando cortes de gráficos iterados de autoria deles cuja proposta era extrair o primeiro plano (*foreground*) com interação mínima por parte do usuário.

**Palavras-chave:** GrabCut, algoritmo, extração, Python, segmentação.

**Abstract—**GrabCut is an algorithm designed by Carsten Rother, Vladimir Kolmogorov and Andrew Blake of Microsoft Research Cambridge UK. The term first appeared in the article "GrabCut": interactive foreground extraction using iterated graph cuts of their own whose purpose was to extract the foreground with minimal user interaction.

**Keywords:** GrabCut, algorithm, extraction, Python, segmentation.

## I. INTRODUÇÃO

A segmentação de objetos sempre foi um dos problemas na área de processamento de imagens, baseado nisso foram criados diversos algoritmos para tentar facilitar o processo. Um desses algoritmos foi o *GrabCut* da biblioteca *OpenCV*, algoritmo cujo propósito principal era segmentar objetos com a menor interação possível do usuário. Seu uso se baseia em fazer um retângulo na região a ser segmentada que o algoritmo faz o resto, baseando-se somente nas cores presentes da região destacada. Dito isso, este artigo busca apresentar o funcionamento do algoritmo, suas falhas e possíveis soluções para as mesmas através de exemplos com imagens.

## II. GRABCUT

O algoritmo *GrabCut* foi projetado por Carsten Rother, Vladimir Kolmogorov e Andrew Blake da Microsoft Research Cambridge, Reino Unido. em seu artigo, "GrabCut": extração interativa de primeiro plano usando cortes de gráficos iterados. O motivo de seu surgimento foi a necessidade de um algoritmo destinado a extração de primeiro plano, ou *foreground*, com interação mínima do usuário, e o resultado foi *GrabCut*.

### A. Inicialização

Primeiro é necessário receber os parâmetros da classe normal, para que o algoritmo analise suas distribuições. Para isso, o usuário realizar a seleção da área que o objeto se encontra, vão se criar duas regiões, a *foreground* e o *background*.



Fig. 1. Seleção do objeto pelo usuário.

### B. Modelo gaussiano

Quando o algoritmo recebe as regiões, será feita a implementação de um modelo estatístico, o modelo de mistura gaussiana. Esse modelo consiste em agrupar elementos ou amostras muito parecidas, o que na situação em questão, seria levado em consideração as cores de cada *pixel*. Quando o modelo percebe que existem cores parecidas, ele as agrupa em um conjunto, numa gaussiana.

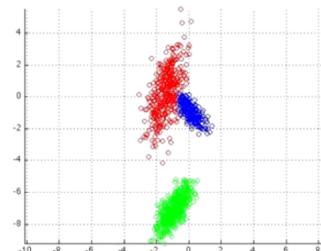


Fig. 2. Exemplo de um modelo de mistura gaussiana.

### C. Gráfico de uma gaussiana

Como uma gaussiana representa modelos com desvio padrão relativamente baixo e muito próximo da média, ela tende agrupar realmente somente as amostras parecidas.

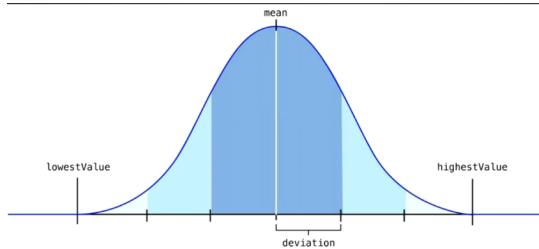


Fig. 3. Exemplo de um gráfico de gaussiana.

O modelo em si consiste em várias gaussianas que vão realizar a separação dos conjuntos. O princípio é que cada um seja muito parecido e que o desvio padrão seja o menor possível, pois resultará em uma menor variância na distribuição.

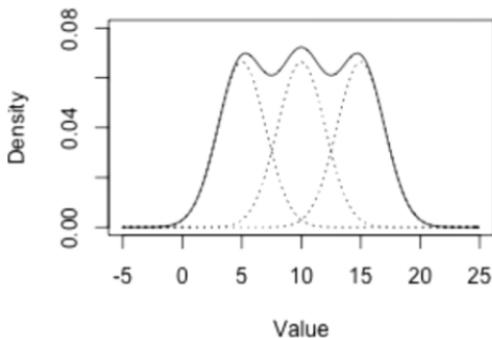


Fig. 4. Exemplo da mesclagem de gaussianas.

Porém, em situações que *pixels* do objeto são muito parecidos com os do *background*, o modelo gaussiano pode não ser suficiente para realizar a segmentação. Dessa forma, o Grabcut da uma penalidade para elementos vizinhos: *pixels* vizinhos tendem a permanecer juntos, assim como *pixels* de mesma cor distantes tendem a ser de conjuntos diferentes. Assim, é necessário conciliar um meio termo entre a distribuição de cores e a vizinhança.

### D. Construção do Grafo

Para obter a segmentação, o grabcut aproveita a estrutura gráfica da imagem. Cada *pixel* possui vários *links*: um “n-link” para cada um dos seus 4 vizinhos diretos e dois “t-link”, um representando a origem (ou objeto) e outro o *background*. Com esses *links*, é possível realizar a classificação de cada grafo por valor.

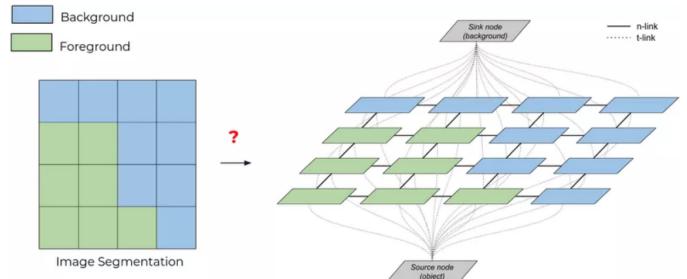


Fig. 5. Construção do grafo.

Quanto aos valores atribuídos a cada *pixel*, leva-se em conta como os links estão manejados. Para os “n-links”, o custo está associado à dissimilaridade entre os valores dos *pixels*. Se os *pixels* vizinhos são semelhantes, o custo é baixo; se forem diferentes, o custo é alto. Os ”t-links” refletem a probabilidade de um *pixel* pertencer ao *foreground* ou ao *background*, conforme estimado pelos Modelos de Mistura Gaussiana: quanto maior a probabilidade de estarem no *foreground*, receberão valores mais baixos para se conectar ao nó de origem.

### E. Cálculo

Os valores atribuídos a cada *pixel* serão computados em fórmulas que serão utilizadas no próximo passo para a realização da segmentação. Com os dados dos *links*, temos um termo designado para cada um.

1) *Termo de dados (V)*: Leva em consideração o peso ”n-link” entre *pixels* cortados pela segmentação. Possui a seguinte fórmula que será utilizada futuramente:

$$V(\underline{\alpha}, \mathbf{z}) = \gamma \sum_{(m,n) \in \mathcal{C}} [\alpha_n \neq \alpha_m] \exp -\beta \|z_m - z_n\|^2$$

Fig. 6. Fórmula ”n-link”.

”Alpha” representa os rótulos dos *pixels* e ”z” representa as intensidades dos *pixels*.

2) *Termo de Suavidade (U)*: O termo de suavidade leva em consideração a modelagem da cor de fundo. Possui a seguinte fórmula que será utilizada futuramente:

$$U(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z}) = \sum_n D(\alpha_n, k_n, \underline{\theta}, z_n)$$

Fig. 7. Fórmula ”t-link”.

”Alpha” representa os rótulos dos *pixels*, ”k” e ”theta” são parâmetros do modelo de mistura gaussiana, e ”z” representa as intensidades dos *pixels*.

### F. MinCut

Finalmente, o MinCut será utilizado para separar os *pixels* que representam o *foreground* dos que representam o *background*, contribuindo assim para a segmentação da imagem.

Uma vez que os pesos são definidos, a função de custo ou função de energia é a soma desses pesos sobre o grafo.

$$E(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z}) = U(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z}) + V(\underline{\alpha}, \mathbf{z}),$$

Fig. 8. Fórmula "MinCut".

"Alpha" representa os rótulos dos *pixels*, "k" e "theta" são parâmetros do modelos de mistura gaussiana, e "z" representa as intensidades dos *pixels*.

#### G. Refinamento

Caso a segmentação ocorra como o esperado, o processo é finalizado. Caso contrário, o usuário poderá fazer ajustes manuais, como uma espécie de tratamento manual (Fig. 16). Após as considerações necessárias, o algoritmo irá realizar uma última análise e a imagem estará segmentada corretamente.

### III. EXEMPLOS

A seguir será analisada a eficiência do algoritmo em algumas situações práticas.

#### A. Código usando algoritmo padrão sem tratamento

No primeiro teste, foi criado um código em Python utilizando o algoritmo de maneira padrão, passando como parâmetros a imagem e alguns vetores. O propósito inicial era verificar a eficiência do *GrabCut* e os resultados foram os seguintes:



Fig. 9. Conjunto de objetos em um fundo branco.

#### B. Conclusões do código usando algoritmo padrão sem tratamento

Com base nos resultados adquiridos notamos que, pela primeira imagem possuir um fundo branco, o algoritmo se mostrou extremamente eficiente em segmentar a área escolhida, porém na imagem com um fundo mais complexo, o



Fig. 10. Objeto da imagem escolhido para ser aplicado o GrabCut.

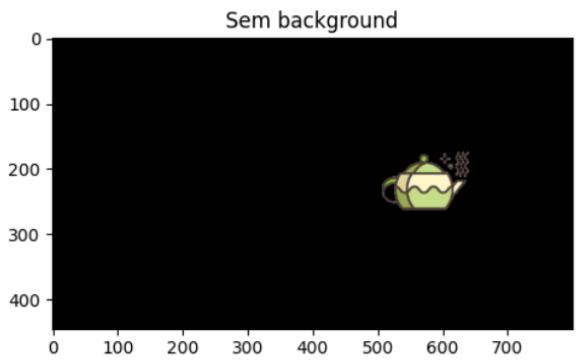


Fig. 11. Resultado da execução do algoritmo.



Fig. 12. Pessoa andando com um cenário mais complexo.

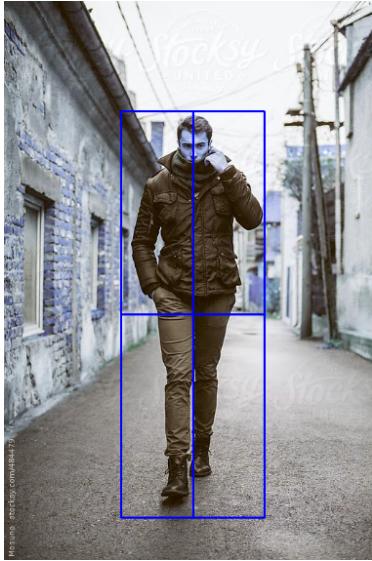


Fig. 13. Aplicar o GraCut na pessoa.

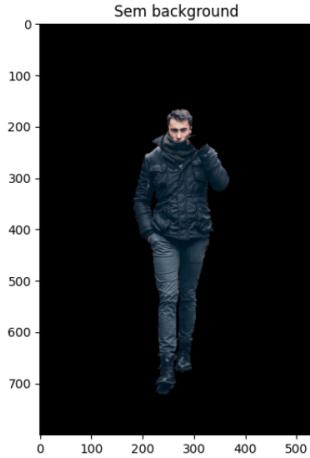


Fig. 14. Resultado da silhueta da pessoa.

resultado se mostrou suscetível à falhas. Buscando contornar o problema encontrado, utilizamos um código que aplica o algoritmo *GrabCut* juntamente com uma ferramenta manual de tratamento de erros.

#### C. Código usando algoritmo padrão com tratamento

O próximo código possui um tratamento a mais, ele disponibiliza uma edição no qual o usuário consegue desenhar riscos na imagem em que foi aplicado o *GrabCut* com o propósito de manter ou retirar alguma região. As regiões a serem mantidas são marcadas de verde e as serem retiradas são vermelhas, ao marcar uma região e apertar a tecla de confirmação, é executado novamente o algoritmo *GrabCut* considerando as regiões citadas previamente, obtendo os seguintes resultados:

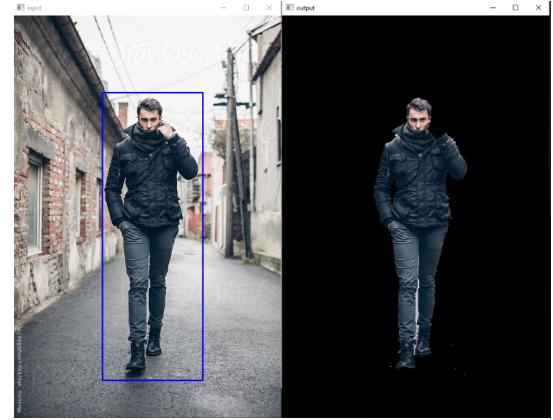


Fig. 15. Lado a lado da pessoa andando com o GrabCut aplicado.



Fig. 16. Resultado tratado manualmente.



Fig. 17. Seleção da região do carro que irá receber o GrabCut.

#### D. Conclusões do código usando algoritmo padrão com tratamento

A seleção de áreas de maneira manual, apesar de ser mais trabalhoso para o usuário, se mostrou bastante eficiente para contornar as falhas do *GrabCut* padrão. A complexidade da correção manual depende do resultado do *GrabCut* padrão, como podemos ver na imagem 3, o carro preto possui a sombra escura e o brilho dele tem cor similar ao céu, o que faz com que o algoritmo que funciona, principalmente, baseado em cor, apresente um comportamento pouco preciso.



Fig. 18. Contorno do carro realizado automaticamente pelo GrabCut.

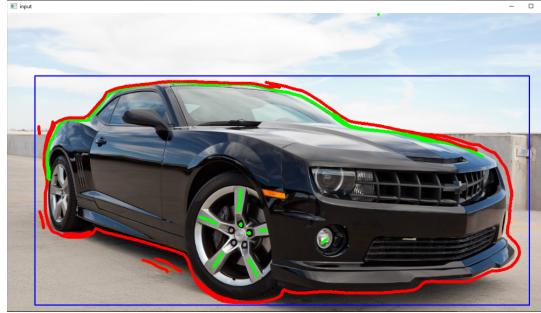


Fig. 19. Grabcut e tratamento manual das regiões imperfeitas.



Fig. 20. Contorno final do carro.

#### IV. CONCLUSÃO

Baseado nas pesquisas e nos resultados analisados, podemos concluir que o algoritmo por si tem a capacidade de apresentar um bom resultado na seguimentação de imagens dependendo da aplicação e, quando não, pode-se aplicar tratamentos variados que podem contornar as falhas para se atingir o objetivo desejado.

#### REFERENCES

- [1] PROST, Julie. GrabCut for Automatic Image Segmentation [OpenCV Tutorial]. Acessado em: Dez. 1 de 2023.
- [2] G. Zhang. What is the kernel trick? Why is it important?. Acessado: Nov. 25, 2023. [Online].
- [3] Visão Computacional. Identificação, Detecção, Reconhecimento e Segmentação de Imagem e Objetos. Acessado em: Dez. 1 de 2023. [Online].
- [4] OpenCV. Interactive Foreground Extraction using GrabCut Algorithm, 2023. Acessado em: Nov. 23 de 2023. [Online]