Processamento de imagem- Modelos de Perona-Malik e Mumford-Shah

Victor Bastos Canut Costa[1,†]

[1]*Instituto de Fisica de São Carlos - USP*

Esse texto foi compilado em Junho 27, 2024

Abstract

O processamento de imagens é uma área recente da ciência, tendo surgido apenas no último século, com a criação dos computadores, e ganhando popularidade com o surgimento da câmera digital. Apesar de sua origem ser apenas para a melhor visualização de fotos devido à limitação dos computadores da época, com o avanço das tecnologias essa área também cresceu. Essa área deixou de docar apenas na visualização e passando a almejar a manipulação dos pontos de interesse da imagem. Atualmente, por meio dessa área, podemos manipular imagens digitais conforme a necessidade, permitindo receber benefícios muito maiores em troca de percas que não fazem muito impacto. Por exemplo, câmeras de segurança focam em detectar movimento, sacrificando uma visão de cores e alguns detalhes para ter um destaque maior quando alguma movimentação anômala for detectada. Para chegar nesse ponto, o processamento de imagem passou por saltos de seu desenvolvimento. Nesse texto, um pouco mais é inicialmente introduzido, destacando a origem do processamento digital mesmo, e em seguida são explicadas as principais características dos métodos de Perona-Malik e de Mumford-Shahh, destacando algumas das suas principais equações, que constituem a base do seu processo. Em seguida, é explicado como foram feitos os códigos, passo a passo, e são feitas ressalvas quanto aos dois. Os códigos são inseridos logo em seguida, e, depois deles, são mostrados os resultados (ou a falta deles). Por fim, uma conclusão generalizando o trabalho é feita.

Keywords: *Perona-Malik, Mumford-shah, processamento de imagens, difusão anisotrópica*

1. Introdução

Quando nos referimos ao processamento de imagens digitais, nos referimos a métodos matemáticos utilizados para compensar a falha dos equipamentos tecnológicos que as produzem, em uma tentativa de salvar os detalhes de alguma forma. O primeiro êxito dessa área foi no processo do melhoramento de imagens da lua em 1964, no qual imagens que antes eram borrões se tornaram as primeiras imagens espaciais da Lua. O processamento digital de imagens tem evoluído desde então, com o propósito de melhorar a visualização de imagens em processos].

Com a popularização dos computadores, o processamento digital de imagens se fortaleceu, com diversas propostas de processamento surgindo. Uma dessas foi a difusão gaussiana, a qual estabelecia a média dos valores dos píxeis de uma imagem usando uma curva de gauss para isso. Posteriormente, outros métodos surgiram, os quais são tanto mais populares quanto mais eficientes no processamento de imagem. Dois desses métodos que se destacam até hoje são o método da anisotropia difusa, também chamado de método de Perona-Malik, o qual utiliza o fluxo dos píxeis da imagem para conseguir aplicar um filtro em regiões dependendo da quantidade de informação nela, e o funcional de Mumford-Shah, o qual é um processo que trabalha mais com a energia da imagem, de modo a suavizar a curva de energia. Nesse trabalho, serão apresentadas as principais ideias por trás dos processos de Perona-Malik e Mumford-Shah, e será feita uma tentativa de replicá-los, de modo a observar como que, ao se alterar padrões de suas equações, o processamento das imagens também é alterado

2. Difusão anisotrópica

Um dos modelos mais utilizados para processamento de imagens é o método de Perona-Malik, uma técnica de processamento que almeja a redução de ruído sem que o essencial da imagem seja perdido. Proposta em 1987, ele é composto por uma equação não linear de difusão parcial, a qual possui um coeficiente de difusividade não-homogêneo. A partir da aplicação dessa equação, podemos modificar uma imagem, de modo a reduzir o ruído, sem perder muitos dos aspectos principais que tornam a imagem identificável. Para aprofundar mais em como funciona exatamente o processo, será necessário, primeiro, explicar sobre a difusão gaussiana.

2.1. Difusão gaussiana

A difusão gaussiana é um processo de processamento de imagem que tenta reduzir o ruído gaussiano (ruído o qual a função densidade de probabilidade tem o formato de uma curva gaussiana) tentando "normalizar" o píxel com o do lado. O princípio dessa difusão está na equação 2.1, que fornece uma curva gaussiana como resultado. A informação transmitida por essa curva é a de que os pixeis mais centrais possuem mais "detalhes" enquanto os pixeis de canto possuem menos detalhes, de modo que normalização colocaria eles todos na média, borrando a figura um pouco, mas garantindo uma perca do ruído dela

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

na qual σ representa o desvio padrão da curva.

O problema desse método é a perca de detalhes que esse processo de equalização acaba gerando, devido a essa associação com píxeis vizinhos[4]. Como se está tomando a média dos valores de uma gaussiana, questões como a luminosidade e detalhes menos destacados de imagens acabam sendo perdidos. Por isso, esse método, apesar de ser a base do processamento de imagem, necessita de uma correção.

2.2. Equação de Perona-Malik

O problema com a difusão gaussiana para filtragem de imagens digitais é a sua homogenização, que acaba por gerar uma difusão da imagem em partes que não são tão interessantes, por gerar uma perca de informação[5]. Para resolver esse problema, entra a equação proposta por Pietro Perona e Jitendro Malik, uma equação utilizada em conjunto com o conhecimento prévio de difusão para montar um algoritmo que visa a preservação de detalhes mais importantes e a detecção de bordas da imagem[1]. Em essencial, a equação de Perona-Malik faz uma modificação na equação de calor, adicionando a ela o coeficiente de difusividade dependendo do local da imagem, sendo essa medida feita conforme a norma do gradiente da região em análise. Em regiões com maior inomogeneidade, é medido a presença de um gradiente maior, o que apresentaria uma concentração maior de detalhes, e por isso o coeficiente de difusão utilizado seria menor. Para regiões mais homogêneas, a norma do gradiente será menor, indicando uma presença menor de detalhes, aplicando um fator maior do coeficiente de difusão para esses pontos. Partindo desse princípio, a formulação da difusão de Perona-Malik é descrita pela equação 1

$$\frac{\partial I}{\partial t} = \nabla(c(|\nabla I|^2)\nabla I) \quad (1)$$

$$\frac{\partial I}{\partial n} = 0 \quad (2)$$

Ambas valendo para um domínio limitado pela imagem, onde "I" representa a intensidade da imagem e "t" representa o passo no tempo. A resolução dessas equações é feita em um processo iterativo de setores, somados para compor a imagem final. Entretanto, um fator importante nesse problema é o fator "c", que corresponde ao coeficiente de condutância, e que depende do gradiente da imagem. Para se fazer o processamento de imagem, é necessário determinar o valor de c. Perona-Malik propuseram duas possíveis soluções [4] para o valor de c 3 4

$$c = e^{-\frac{s^2}{\kappa^2}} \quad (3)$$

$$c = \frac{1}{1 + \frac{s^2}{\kappa^2}} \quad (4)$$

Onde "s" é o gradiente da imagem, anteriormente usado e escrito dessa maneira para facilitar a escrita, e κ é o coeficiente de condução, o qual determina o quão bem a difusão vai ocorrer, principalmente nas bordas, sendo que κ deve ser positivo. O fluxo desse problema também é importante para explicar o porquê desse processo ser chamado de anisotrópico. O fluxo é dado pela equação 5

$$\Phi = s \cdot c(s^2) \quad (5)$$

Utilizando esse fluxo, a equação 1 pode ser reescrita da forma 6

$$\frac{\partial I}{\partial t} = \Phi'(s) I_{\nu\nu}(c(s^2) I_{\xi\xi}) \quad (6)$$

onde ν e ξ são coordenadas das direções paralelas e perpendiculares ao gradiente de I, respectivamente. Analisando essa equação, nota-se que a difusão ocorrerá em paralelo ao contorno de I e perpendicularmente haverá uma difusão em sentido oposto, causando com que as pontas tenham mais detalhes. Por causa disso, esse processo pode ser dito como uma difusão anisotrópica e explica o porquê dos centros serem mais borrados que as pontas nas imagens que passam por esse processo.

3. Mumford-Shah

O processo de Mumford-Shah começa a se diferenciar em sua concepção, uma vez que, enquanto o processo Perona-Malik [6] foi já proposto para ser uma versão melhorada de outros processos de processamento de imagem. O processo de Mumford-Shah, também chamado de funcional de Mumford-Shah, foi uma proposta de otimização da segmentação de imagens. A partir dele, foram derivados outros processos, como um de melhoria de imagem e detecção de bordas[3] [2]. O processo se baseia em minimizar a função energia de uma imagem, de modo a deixá-la mais suave a partir disso. Há uma perca de detalhes nas imagens, em prol de uma maior homogenização, não diferente do processo de Perona-Malik, apesar de os dois métodos abordarem por frentes diferentes a solução desse problema. A energia a ser minimizada é descrita na equação 7

$$E_{MS}(u,K) = \int_{\Omega/K} (u - g^2)dx + \int_{\Omega/K} |\nabla u|^2 dx + H^{n-1}(K) \quad (7)$$

Pode se reparar que esse é um processo extremamente complicado de se implementar no computador. Por causa disso, pode ser utilizada uma aproximação de Ambrosio-Tortorelli [7] [8] para a equação 7, de modo a obtermos:

$$E_{AM}(u,K) = \int_{\Omega} (\beta(u-f^2) + \alpha(v^2|\nabla u|^2) + \frac{1}{2}(\rho|\nabla v|^2 + \frac{1-v^2}{\rho}))dx \quad (8)$$

Onde α e β são parâmetros de regularização positivos, u é a imagem original que passará pelo processamento, v é uma variável auxiliar que representa as funções de contorno e ρ tendendo a zero. Usando um processo de somatório para essas funções.

4. Métodos

Nesse trabalho, foram examinados os dois métodos e foi feita uma tentativa de replicá-los em códigos de python, sem se afastar muito do que foi visto em sala. Para o código de Perona-Malik, foi inicialmente definida a função que regeria todo o código, com diversas variáveis atribuídas, as quais foram já descritas anteriormente enquanto se explicava o processo, para que caso seja necessário, essas possam ser alteradas. A leitura da imagem inicial é feita, e em seguida são montadas matrizes de convolução no formato de float (para poderem interagir com a imagem) para trabalhar separadamente com setores da imagem. Utilizamos, em seguida, um loop "for" para repetir o processo e simular uma integral, e dentro dele determinamos o valor do gradiente da imagem pela convolução dela pelas matrizes preparadas anteriormente. Por fim, usamos uma condição if para determinar qual o tipo de coeficiente de condutividade será utilizado. No fim, é só a aplicação desse processamento em uma imagem, com os parâmetros necessários determinados anteriormente, e são mostradas a imagem original e pós-processo no fim.

Os dois códigos seguintes são um pouco mais complicados, principalmente devido aos seus resultados. Uma mesma base foi usada, onde a função é definida anteriormente, apesar de que nesse segundo caso foi necessário definir uma função antes da principal, visando nos aproximar da equação 7. Definimos uma função que retorna valor do gradiente da imagem e normalizamos a figura, além de fazer uma cópia para não afetar a original. Tomamos o valor do gradiente de u e, com ele, fazemos operações em um loop "for", novamente tentando contornar o problema da integral. Utilizamos os valores dos gradientes para calcular um novo valor de u, com mu sendo uma variável que rege as pontas, enquanto as outras são análogas ao que já foram usadas. Aplicamos essa função no resultado e obtemos uma imagem com tons de cinza mais deturpados, mas ainda com os ruídos que gostaríamos de retirar. Mesmo com esse processo sendo refeito, não foi possível atingir o valor desejado, então foi feito um segundo código. Esse segundo código conta com ideias e passagens semelhantes, as quais estão explicadas nele, almejando agora resolver pela equação 8. O processo foi feito, com uso de um for e calculando gradientes e divergentes das principais variáveis da equação. Entretanto, no fim, as imagens não saem de maneira ideal, sendo a imagem sem ruído apenas imagem monocromática cinza. Entretanto, esse segundo programa conseguiu fornecer o contorno de imagens de forma satisfatória, ao menos, não sendo um fracasso completo.

5. Codes

```
1
2   import numpy as np
3   import cv2
4   import matplotlib as mp
5   import matplotlib.pyplot as plt
6   from scipy.ndimage import convolve
7   from skimage import io, transform
8
9   def perona_malik(img, num_iter, delta_t, kappa,
        option=1):
10      """
11      Processo de difus o anisotr pica de Perona
        -Malik.
```

```
    Parameters:
    img : imagem (array).
    num_iter : n mero de itera  es (int).
    delta_t : passo/tempo de integra  o (float).
    kappa : Condutividade (float).
    option : op  o de fun  o de difus o (int).
         1: g = exp(-(grad/kappa)**2)
         2: g = 1 / (1 + (grad/kappa)**2)

    """

    diffused_img = img.astype(float)

    # Matrizes de convolu  o , necess rias para trabalhar com setores diferentes da imagem
    hN = np.array([[0, 1, 0],
                   [0, -1, 0],
                   [0, 0, 0]], dtype=float)

    hS = np.array([[0, 0, 0],
                   [0, -1, 0],
                   [0, 1, 0]], dtype=float)

    hE = np.array([[0, 0, 0],
                   [0, -1, 1],
                   [0, 0, 0]], dtype=float)

    hW = np.array([[0, 0, 0],
                   [1, -1, 0],
                   [0, 0, 0]], dtype=float)

    for i in range(num_iter):
        # c lculo dos gradientes
        nablaN = convolve(diffused_img, hN)
        nablaS = convolve(diffused_img, hS)
        nablaE = convolve(diffused_img, hE)
        nablaW = convolve(diffused_img, hW)

        # C lculo da fun  o de difus o
        if option == 1:
            cN = np.exp(-(nablaN / kappa) ** 2)
            cS = np.exp(-(nablaS / kappa) ** 2)
            cE = np.exp(-(nablaE / kappa) ** 2)
            cW = np.exp(-(nablaW / kappa) ** 2)
        elif option == 2:
            cN = 1.0 / (1.0 + (nablaN / kappa) ** 2)
            cS = 1.0 / (1.0 + (nablaS / kappa) ** 2)
            cE = 1.0 / (1.0 + (nablaE / kappa) ** 2)
            cW = 1.0 / (1.0 + (nablaW / kappa) ** 2)

        diffused_img += delta_t * (cN * nablaN + cS * nablaS + cE * nablaE + cW * nablaW)

    return diffused_img

img1 = cv2.imread('/content/drive/MyDrive/Images/Lunch_atop_a_Skyscraper.png', cv2.IMREAD_GRAYSCALE)

num_iter = 25
delta_t = 1/7
kappa = 15
option = 1

img1final = perona_malik(img1, num_iter, delta_t, kappa, option)

plt.imshow(img1, cmap='gray')
plt.title('Original')
plt.show()

plt.imshow(img1final, cmap='gray')
plt.title('Perona-Malik')
plt.show()
```

```

import numpy as np
import cv2
import matplotlib.pyplot as plt

def gradient_magnitude(img):
    gx = np.gradient(img, axis=0)
    gy = np.gradient(img, axis=1)
    return np.sqrt(gx**2 + gy**2)

def mumford_shah_denoising(img, num_iter, lambda_, mu, dt):
    img = img.astype(np.float32) / 255.0
    u = img.copy()

    gu= np.gradient(u)

    for _ in range(num_iter):
        u_old = u.copy()
        gradient_u = gradient_magnitude(u)

        u = u - dt * (u - img - lambda_ * (u - img / 255) * (1 - gradient_u) - mu * gradient_u * gu)

        if np.linalg.norm(u - u_old) < 1e-3:
            break

    return (u * 255).astype(np.uint8)


input_image = cv2.imread('/content/drive/MyDrive/Images/Noisy-image.png', cv2.IMREAD_GRAYSCALE)

num_iter = 100
lambda_ = 0.1
mu = 0.1
dt = 0.01


denoised_image = mumford_shah_denoising(input_image, num_iter, lambda_, mu, dt)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Original')
plt.imshow(input_image, cmap='gray')
plt.subplot(1, 2, 2)
plt.title('Mumford-Shah')
plt.imshow(denoised_image, cmap='gray')
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
import imageio

def compute_gradients(u):
    grad_u_x = np.roll(u, -1, axis=1) - u
    grad_u_y = np.roll(u, -1, axis=0) - u
    return grad_u_x, grad_u_y

def compute_divergence(grad_u_x, grad_u_y):
    div_u = np.roll(grad_u_x, 1, axis=1) - grad_u_x + np.roll(grad_u_y, 1, axis=0) - grad_u_y
    return div_u

def ambrosio_tortorelli_denoising(img, alpha, epsilon, dt, iterations):
    u = img.copy()
    v = np.ones_like(img)

    for _ in range(iterations):

```

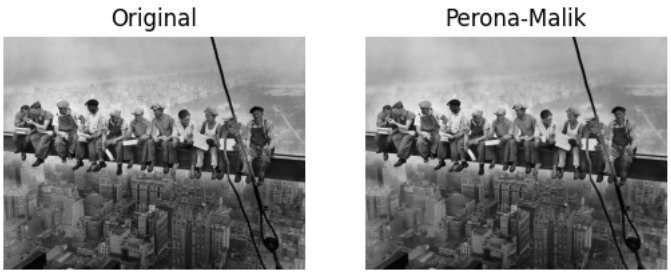

Figure 1. Imagem oriunda de Perona-Malik. Fonte: do autor

```
        grad_u_x, grad_u_y = compute_gradients(u
    )
        grad_v_x, grad_v_y = compute_gradients(v
    )

        div_u = compute_divergence(grad_u_x,
    grad_u_y)
        div_v = compute_divergence(grad_v_x,
    grad_v_y)

        u += dt * (div_u - (u - img) + alpha * v
     * (img - u))

        v += dt * (epsilon * div_v - alpha * (
    img - u)**2 * v)
        v = np.clip(v, 0, 1)
    return u, v

img = imageio.imread('/content/drive/MyDrive/
    Images/folhas.png', mode='L')


img = img / 255.0

alpha = 5
epsilon = 1e-3
dt = 0.1
iterations = 50

denoised_img, v = ambrosio_tortorelli_denoising(
    img, alpha, epsilon, dt, iterations)

plt.figure(figsize=(18, 6))

plt.subplot(1, 3, 1)
plt.title('Original Image')
plt.imshow(img, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.title('Denoised Image')
plt.imshow(denoised_img, cmap='gray')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.title('Edge Indicator')
plt.imshow(v, cmap='gray')
plt.axis('off')

plt.show()
```

6. Resultados gerados

Percebemos, aqui, que foi um sucesso o processo de Perona-Malik, e é possível notar como a variação de cada uma das variáveis presentes afeta o resultado, seja o passo reduzindo a luminosidade da figura 3, seja o coeficiente de condução tornando ela mais borrada 4. Podemos dizer que o código para Perona-Malik foi um sucesso, e podemos notar o quão maleável é essa ferramenta graças a isso. Entretanto, um resultado notável está na figura 4, onde o último resultado, após 5000 iterações, começou a perder a forma, algo que não era o esperado

Figure 2. Imagem oriunda de Perona-Malik, ao se alterar o valor de κ. Fonte: do autor

Figure 3. Imagem oriunda de Perona-Malik, ao se alterar o valor de d_t Fonte: do autor

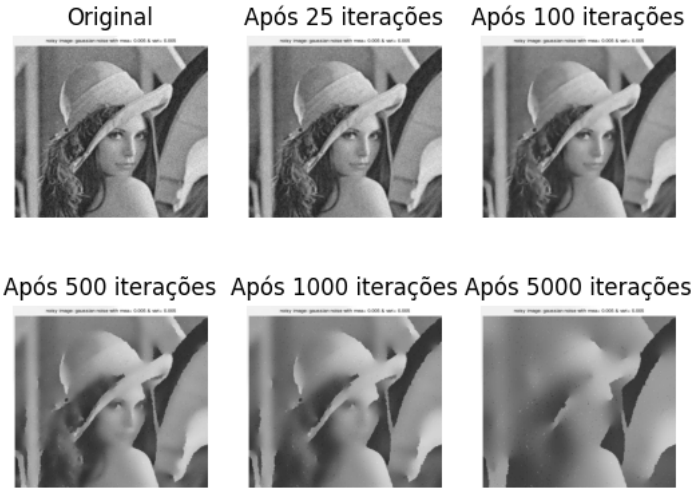

Figure 4. Imagem oriunda de Perona-Malik, ao se alterar o número de iterações. Fonte: do autor

Figure 5. Imagem oriunda de Perona-Malik, ao se alterar a equação do coeficiente de condutividade. Fonte: do autor

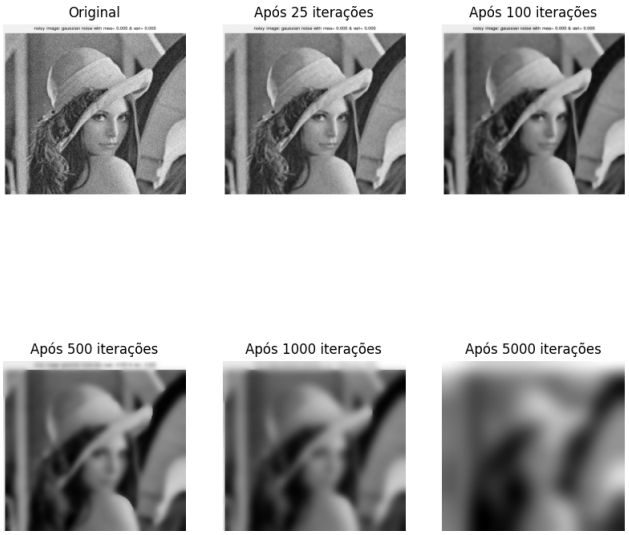

Figure 6. Imagem oriunda de Perona-Malik, ao se alterar o número de iterações, utilizando equação 4 ao invés de 3. Fonte: do autor

conforme a teoria de Perona-Malik. Como o código não possui muitas variáveis além daquelas que afetam o gradiente, pressupõe-se que o problema provavelmente está no fato de, ao tratar de vizinhanças, um píxel na posição central acabaria sofrendo maior deformação nas várias iterações, justificando assim a perca da forma do chapéu na imagem. Para verificar se esse caso ocorreria em outra situação, foi montada a figura 6, a qual combina o número de iterações com o método do segundo coeficiente proposto por Perona e Malik. O resultado esperado era uma imagem borrada e que perderia suas bordas, conforme foi visto nos casos individuais anteriores. Entretanto, o que se notou foi que, apesar da imagem ter ficado extremamente difícil de distinguir detalhes, por estar extremamente borrada, ela a forma dela não sofreu como no primeiro caso de múltiplas iterações, o que leva a crer que, talvez, o problema esteja em utilizar o coeficiente exponencial 3 e não o coeficiente aproximado 4.

No que se trata de Mumford-Shah, por outro lado, o procedimento de retirada de ruído não teve sucesso. As imagens geradas para os

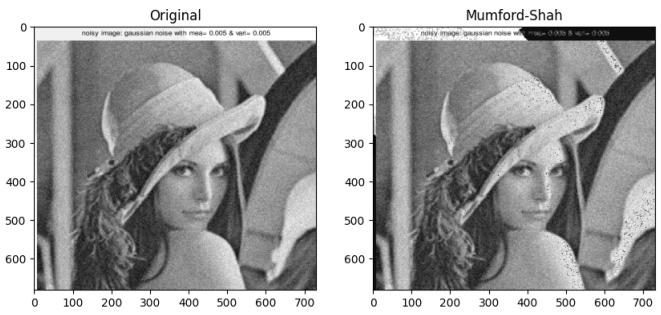

Figure 7. Imagem oriunda do primeiro código de Mumford-Shah. Fonte: do autor

Figure 8. Imagem oriunda do segundo código de Mumford-Shah. Fonte: do autor

Figure 9. Imagem oriunda do segundo código de Mumford-Shah, analisando somente o contorno. Fonte: do autor

dois códigos, tanto aquele baseado em 7 quanto o segundo, baseado em 8, não obtiveram os resultados desejados. Como podemos ver pela figura 7, o primeiro código não fez o processo de forma adequada, gerando uma imagem mais ruidosa do que originalmente tínhamos. Já para o segundo código, nota-se pela imagem 8 que o processo também não funcionou ao utilizar a equação de Ambrosio-Tortorelli, fornecendo apenas uma imagem monocromática de cinza. Entretanto, dessa equação, podemos também tirar uma variável que denota o contorno das imagens, e, como foi mostrado pela imagem 9, podemos tirar algumas informações da maneira como alguns fatores afetam esse contorno. Como o contorno deu um resultado minimamente satisfatório, especula-se que o motivo de falha do programa esteja nos cálculos relacionados a variável "u". Outra possibilidade está no fato de, ao contrário de Perona-Malik, Mumford-Shah (e consequentemente Ambrosio-Tortoreli) é um processo muito mais difícil de ser executado, principalmente devido a sua integral mais complexa. Por causa disso, a adoção de um loop de interações para esse processo, bem como a adoção de uma escrita mais simplificada são fatores que provavelmente pesaram muito para que esse erro ocorresse. Seria necessário um código bem mais complexo para poder fazer uma adaptação realmente adequada desse processo. Mesmo assim, conseguimos notar que, quanto maior o alfa, mais preciso é o contorno que o programa consegue encontrar.

7. Conclusão

Nesse trabalho foram apresentados os métodos de processamento de imagem de Perona-Malik e de Mumford-Shah, bem como discutidos alguns de seus agregados. Também houve uma tentativa de replicar os códigos e, enquanto sucesso foi obtido quando se referindo ao processo de Perona-Malik, o mesmo não pode ser dito acerca do processo de Mumford-Shah, sendo que apenas a detecção de bordas deste foi obtida. Foi notado que, no que se refere a Perona-Malik, a variação de cada parâmetro fornece um resultado diferente, com a opção 1 de coeficiente de condutividade sendo favorecido. Também foi determinado que o quanto mais iterações houverem, menor a qualidade final, apesar de termos obtido um resultado inesperado, com perca de forma, e foi testado que, ao menos para esse caso, a adoção do coeficiente descrito na equação 4 seria a ideal, não sofrendo com o problema de perca de bordas. Foi visto também que um passo menor favorece mais os tons de cinza da imagem, e os coeficientes de condução mais altos acabam por borrar a mesma.

Por outro lado, foi visto que a aplicação do Mumford-Shah não funcionou, ao menos no processo de retirada de ruído, e acredita-se que o principal problema esteja na aplicação dos gradientes no fator "u", gerando diferentes problemas dependendo do código utilizado. No entanto, foi visto que, ao menos parcialmente, o funcional de Ambrosio-Tortorelli funcionou, conseguindo diferenciar bordas e

mostrando como o fator multiplicativo afeta a precisão desse processo. Outro motivo especulado como causa dos problemas é a simplificação de um processo complexo, que possui uma integral que não poderia ser reduzida a apenas diversas iterações.

References

[1] F. Voci, S. Eiho, N. Sugimoto, and H. Sekibuchi, "Estimating the gradient in the perona-malik equation," *IEEE Signal Processing Magazine*, vol. 21, no. 3, pp. 39–65, 2004.

[2] J. B. Florindo, S. H. M. Soares, L. A. V. de Carvalho, and O. M. Bruno, "Mumford–shah algorithm applied to videokeratography image processing and consequences to refractive power values," *Computer methods and programs in biomedicine*, vol. 87, no. 1, pp. 61–67, 2007.

[3] P. Guidotti, "Some anisotropic diffusions," *Ulmer Seminare*, vol. 14, pp. 215–221, 2009.

[4] M. Wielgus, *Perona-malik equation and its numerical properties*, 2014. arXiv: 1412.6291 [cs.NA]. [Online]. Available: https://arxiv.org/abs/1412.6291.

[5] P. Guidotti, "Anisotropic diffusions of image processing from perona–malik on," in *Variational Methods for Evolving Objects*, vol. 67, Mathematical Society of Japan, 2015, pp. 131–157.

[6] D. Vicente, "An anisotropic mumford–shah model," *Journal of Mathematical Analysis and Applications*, vol. 447, no. 1, pp. 181–205, 2017, ISSN: 0022-247X. DOI: https://doi.org/10.1016/j.jmaa.2016.10.008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022247X16305947.

[7] N. Bonneel, D. Coeurjolly, P. Gueth, and J.-O. Lachaud, *Mumford-shah mesh processing using the ambrosio-tortorelli functional*, 2018. arXiv: 1806.05999 [cs.GR]. [Online]. Available: https://arxiv.org/abs/1806.05999.

[8] I. Fonseca, L. M. Kreusser, C.-B. Schönlieb, and M. Thorpe, Γ*-convergence of an ambrosio-tortorelli approximation scheme for image segmentation*, 2023. arXiv: 2202.04965 [math.OC]. [Online]. Available: https://arxiv.org/abs/2202.04965.