Processamento de Imagens
Métodos de Aguçamento e Detecção de Bordas

Caroline de Sala Borba[1], **Vinicius Gasparini**[1]

[1]Departamento de Ciência da Computação
Universidade do Estado de Santa Catarina (UDESC)
Centro de Ciências Tecnológicas – Joinville – SC – Brasil

carolinesala16@gmail.com, v.gasparini@edu.udesc.br

1. Introdução

O presente trabalho tem como objetivo final o aguçamento e a detecção de bordas dos objetos presentes em imagens. Para tal, será utilizado uma técnica de detecção de fronteiras por meio dos descritores gradientes que os contornos geram. Na implementação desse método é requerido que decompúnhamos as derivadas direcionais da imagem por meio do operador de Sobel.

2. Fundamentação

2.1. Filtro de Realce

Segundo [Solomon and Breckon 2013], "o objetivo básico da aplicação de realce é processar a imagem de modo que possamos ver e avaliar a informação visual nela contida com maior clareza". Sendo assim, o realce é um processo muito subjetivo, pois depende da informação que o observador espera extrair da imagem.

Dentre as principais aplicações de filtros de realce citamos a remoção de ruídos, efeito de *blurring* e agudização de bordas da imagem. Sendo este último, o foco deste presente trabalho. Para entender um pouco melhor como ocorre o processo de realce de bordas, precisamos compreender o conceito de vizinhança de *pixels* no contexto de processamento de imagens.

"Central à questão de definição de vizinhança local é a noção de conectividade de *pixels*, ou seja, a determinação de quais *pixels* estão conectados a outros. Uma conectividade 4 implica que somente os *pixels* que se encontram ao N, O, L, S do pixel em questão estão conectados. Contudo, se os *pixels* nas diagonais também estiverem conectados, devem ser considerados; nesse caso, teremos conectividade 8" [Solomon and Breckon 2013].

2.2. Filtragem pela média

Um dos principais usos da filtragem é para a remoção de ruídos. Tal qual existem diversos tipos de ruídos conhecidos, existe uma gama de estratégias para a redução dessas anomalias nas imagens. O filtro de média realiza uma suavização baseada na média das intensidades entre os *pixels* da vizinhança. Seja uma vizinhança de tamanho $N \times M$, cada *pixel* recebe peso $W_k = 1/(NM)$, fazendo portanto com que a soma dos pesos no *pixel* x, y é igual à 1, quaisquer que sejam os tamanhos de vizinhança.

Tal procedimento faz com que localmente a imagem fique mais suavizada, eliminando alguns tipos mais simples e menores de ruídos como sal e pimenta e ruído gaussiano. Essa filtragem traz resultados simplórios quando comparado com outros métodos

de mesmo objetivo, mas é amplamente utilizado de modo a preparar para outros procedimentos.

2.3. Detecção de bordas por gradientes

[Solomon and Breckon 2013] ressalta que além da remoção de ruído, duas outras importantes aplicações de filtragem em imagens são na extração de características e no realce de características. Para trabalhar nesse sentido, precisamos utilizar estratégias mais robustas de análise das imagens como por exemplo o uso de filtros diferenciais.

Uma borda pode ser definida com uma descontinuidade no nível de cinza ou na textura da imagem. Para facilitar a identificação dessas mudanças abruptas recomenda-se o uso de filtros de realce para limpar informações repetitivas e sujeiras.

Descontinuidades poderão ser identificadas utilizando operadores diferenciais. Esses operadores diferencias podem ser de primeira ordem e de segunda ordem. Um exemplo de filtro diferencial de primeira ordem é o filtro de Sobel.

O filtro de Sobel, em comparação com os semelhantes filtros de Prewitt e Roberts, se destaca pela suavização das bordas da vizinhança através da média gaussiana que ocorre nos limites. Para a obtenção das derivadas é realizada uma convolução entre a matriz de vizinhança e as duas matrizes de Sobel (Eq. 1 e 2).

$$Sobel_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{1}$$

$$Sobel_y = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{2}$$

Sob posse das derivadas X e Y da imagem, podemos construir o vetor gradiente através da definição da magnitude $|G|$ (3) e a orientação θ (4).

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{3}$$

$$\theta = tan^{-1}(\frac{G_y}{G_x}) + \frac{1}{4}\pi \tag{4}$$

Para a detecção de borda com base no vetor gradiente, podemos utilizar duas estratégias para a concepção da imagem binária. Uma das estratégias se baseia em dado o valor de magnitude do *pixel* de coordenada x, y, o nível de cinza será 255 caso este seja maior que uma porcentagem da maior magnitude da vizinhança, senão será 0 (5). Outra estratégia possível é olhar para a direção do vetor gradiente do *pixel* x, y. O nível de cinza será 255 caso o *pixel* central seja vencedor entre suas conexões na direção do diretor do vetor. Esse comportamento pode ser melhor entendido na Figura 1.

Sendo T a porcentagem limiar e M a magnitude máxima da vizinhança de x,y

$$I_s(x, y) = 255 \; se \; |G|_{x,y} \geq T * M \; e \; I_s(x, y) = 0 \; se \; |G|_{x,y} < T * M \tag{5}$$

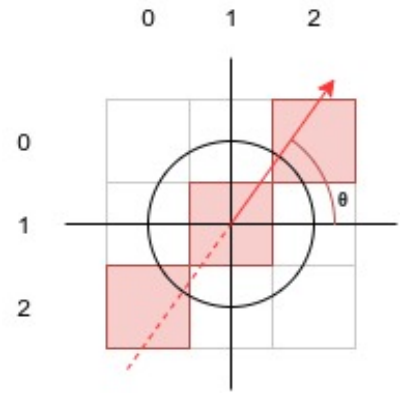

Figura 1. No exemplo, os *pixels* a ser comparados estão destacados em vermelho e são determinados pela direção de θ.

3. Desenvolvimento

3.1. Algoritmo de Janela

Para recortar uma matriz de vizinhança a um *pixel* fez-se necessário modelar um algoritmo que dado uma posição (x, y) e um tamanho de janela S seja retornado uma matriz M^S contendo no centro o *pixel* (x, y).

```
class Window:
    def __init__(self, img):
        self.img = np.array(img)

    def cut_window(self, x, y, window_fullsize = 3):
        h_old, w_old = len(self.img), len(self.img[0])

        window_size = window_fullsize // 2

        left_limit = x  if x - window_size < 0 else window_size
        right_limit = h_old if x + window_size >= h_old else
                                            window_size
        upper_limit = y if y - window_size < 0 else window_size
        bottom_limit = w_old if y + window_size >= w_old else
                                            window_size

        cropped = self.img[
            y - upper_limit : y + bottom_limit + 1, x - left_limit : x
                                                + right_limit + 1,
        ]
        return cropped
```

Algoritmo 1. Classe *Window* para criação de janelas

3.2. Filtro de Realce

O filtro de realce aqui implementado segue algoritmo 2 descrito por [Solomon and Breckon 2013]. Inicialmente o algoritmo realiza uma varredura pela imagem recortando janelas de tamanho 3, calcula a média entre todos os *pixels* dessa janela e aplica a diferença sob o ponto central (6). Agora com a transformação do *pixel* realizada, a equação 7 descreve como

esse filtro é aplicado mediante um intensificador c. Após esse processo é realizado um ajuste para manter o nível de cinza dentro da escala 0-255, descritos pelas equações 8 e 9.

$$m(x,y) = f(x,y) - \bar{f}(x,y) \tag{6}$$

$$g(x,y) = f(x,y) + c * m(x,y) \tag{7}$$

$$g_m(x,y) = g(x,y) - min(g(x,y)) \tag{8}$$

$$g_a(x,y) = 255\left[\frac{g_m(x,y)}{max(g_m(x,y))}\right] \tag{9}$$

```
class Intensify:
    def __init__(self, matrix):
        self.height, self.width = len(matrix), len(matrix[0])
        self.matrix = matrix

    def mean_filter(self, c):
        window_object = Window(self.matrix)
        mean_matrix = []

        for y in range(self.height):
            row = []
            for x in range(self.width):
                janela = window_object.cut_window(x,y)
                row.append( np.sum(janela) // np.size(janela) )
            mean_matrix.append( row )
        mean_matrix = self.matrix - mean_matrix
        return self.matrix + c*mean_matrix

    def adjust(self, mean_matrix):
        adjusted = []
        mini, maxi = np.min(self.matrix), np.max(self.matrix)

        for y in range(self.height):
            row = []
            for x in range(self.width):
                gm = mean_matrix[y][x] - mini
                ga = int( 255 * (gm / maxi) ) if gm > 0 else 0
                row.append( ga )
            adjusted.append( row )
        return adjusted
```

Algoritmo 2. Classe *Intensify* para realce e ajuste da imagem para a escala 0-255.

3.3. Derivada de Sobel

Para decomposição da imagem, já com reforço de nitidez, em suas derivadas é necessário que todos os *pixels* e suas respectivas vizinhanças sejam transformados pelos operadores de Sobel já descritos no capítulo 2. O Algoritmo 3 documenta a rotina dessa operação. Como retorno é esperado que o objeto da classe *Sobel* possua em seus atributos a componente em x e y da imagem.

```
class Sobel:
    def __init__(self, matrix):
        self.height, self.width = len(matrix), len(matrix[0])
        self.matrix = np.zeros((self.height+2, self.width+2))
        self.matrix[1:-1,1:-1] = matrix
        self.x_component = []
        self.y_component = []
        self.x_sobel = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
        self.y_sobel = np.array([[-1,-2,-1],[0,0,0],[1,2,1]])

    def generate_x_y(self):
        window_object = Window(self.matrix)

        for y in range(1,self.height-1):
            row_x, row_y = [],[]
            for x in range(1,self.width-1):
                janela = window_object.cut_window(y,x)
                row_x.append( np.sum(janela * self.x_sobel))
                row_y.append( np.sum(janela * self.y_sobel))
            self.x_component.append( row_x )
            self.y_component.append( row_y )
```

Algoritmo 3. Classe *Sobel* para extrair as derivadas da imagem.

3.4. Processo de Detecção de Bordas

Para essa atividade foi requerido dois métodos de detecção de borda. Por meio de um limiar sob a magnitude do vetor gradiente e outro método sob a direção do vetor gradiente. A classe *Gradient* (Algoritmo 4) possui um método *calculate* que é responsável por aplicar as transformações (3) e (4) por todos as *pixels* da imagem. Ao fim desse processo, obtemos duas novas matrizes, uma de magnitude e outra de direções. Cada célula dessas matrizes correspondem as informações de cada *pixel* da imagem realçada.

O método de detecção de bordas por um limiar está representado pelo método *filter_by_threshold* do objeto *Gradient*. Este método recebe o limiar por parâmetro.

O método de detecção de bordas pela direção do vetor gradiente está representado pelo método *filter_by_direction*. Naturalmente esse procedimento não opera com base em um limitante local ou global para coloração ou não da imagem binária, mas os autores optaram pela inclusão dessa regra para obtenção de uma imagem final mais precisa. Adicionalmente fez-se necessário a definição de uma função que dada a direção do vetor gradiente fosse retornado quais as posições da vizinhança deveriam ser consultadas (Algoritmo 5).

```
class Gradient:
    def __init__(self, x_component, y_component):
        self.x_component = x_component
        self.y_component = y_component
        self.height, self.width = len(self.x_component), len(self.
                                                y_component)
        self.magnitude = np.zeros((self.height, self.width))
        self.direction = np.zeros((self.height, self.width))

    def calculate(self):
        for y in range(self.height):
            for x in range(self.width):
                self.magnitude[y][x] = (
                    self.x_component[y][x] ** 2
                    + self.y_component[y][x] ** 2
                ) ** (1 / 2)
                self.direction[y][x] = np.degrees(
                    np.arctan(
                        self.y_component[y][x] /
                        (self.x_component[y][x] + epsilon)
                    )
                )

    def filter_by_direction(self, threshold):
        new_img_i = np.zeros((self.height, self.width))
        window_object = Window(self.magnitude)
        maxii = np.max(self.magnitude)
        for y in range(2, self.height - 2):
            for x in range(2, self.width - 2):
                janela = window_object.cut_window(y, x)
                direction = get_direction(self.direction[y][x])
                a, b = direction[0], direction[1]
                maxi = max(
                    self.magnitude[y + a[0]][x + a[1]],
                    self.magnitude[y + b[0]][x + b[1]],
                )
                if self.magnitude[y][x] > maxi:
                    new_img_i[y][x] = (
                        255 if
                        (self.magnitude[y][x] / maxii)* 255 > threshold
                        else 0
                    )
        return new_img_i

    def filter_by_threshold(self, threshold):
        new_img = np.zeros((self.height, self.width))
        maxi = int(np.max(self.magnitude) * threshold)
        for y in range(2, self.height - 2):
            for x in range(2, self.width - 2):
                if self.magnitude[y][x] >= maxi:
                    new_img[y][x] = 255
        return new_img
```

Algoritmo 4. Classe *Gradient* contendo os dois métodos de detecção de borda.

```
def get_direction(angle):
    angle = round(angle) % 360
    if 22.5 >= angle >= 0 \
    or 360 >= angle > 337.5 \
    or 202.5 >= angle > 157.5:
        return [[1, 2], [1, 0]]
    elif 67.5 >= angle > 22.5 or 247.5 >= angle > 202.5:
        return [[0, 2], [2, 0]]
    elif 112.5 >= angle > 67.5 or 292.5 >= angle > 247.5:
        return [[0, 1], [2, 1]]
    elif 157.5 >= angle > 112.5 or 337.5 >= angle > 292.5:
        return [[0, 0], [2, 2]]
```

Algoritmo 5. Função para detecção da direção do vetor gradiente.

4. Experimentação

O primeiro experimento corresponde a questão 2 da atividade.

Figura 2. Imagem original e após realce de nitidez com *C=5*.

O segundo experimento é sob a decomposição da imagem em suas derivadas através do operador de Sobel.

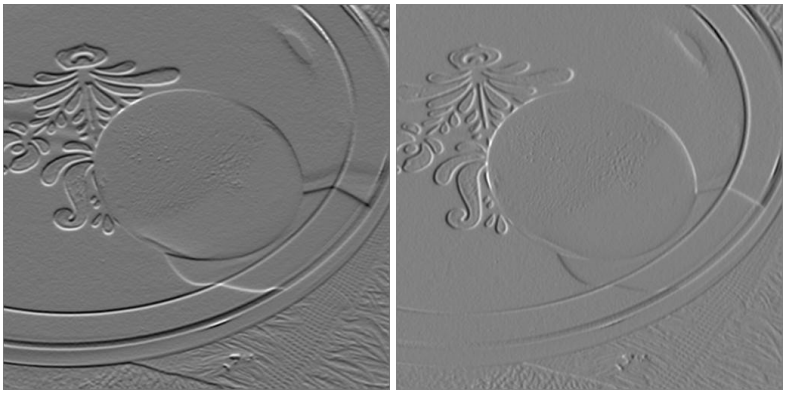

Figura 3. Derivada em X e derivada em Y da imagem realçada por meio do operador de Sobel.

Sob posse das derivadas, geramos o vetor gradiente e assim aplicamos o filtro de detecção de borda por limiar de máximo global em relação a magnitude.

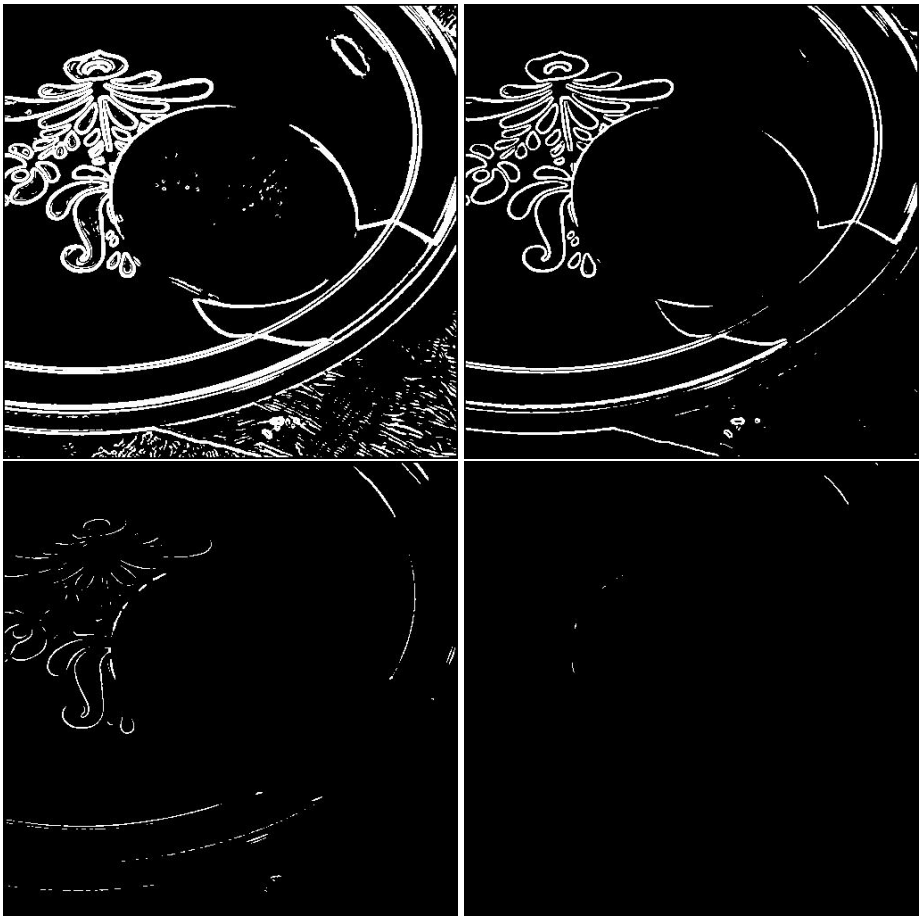

Figura 4. Da esquerda para a direita, de cima para baixo, temos a aplicação de limiar 5%, 10%, 20% e 33%

Por fim, a aplicação do filtro de detecção através do uso da direção do vetor gradiente.

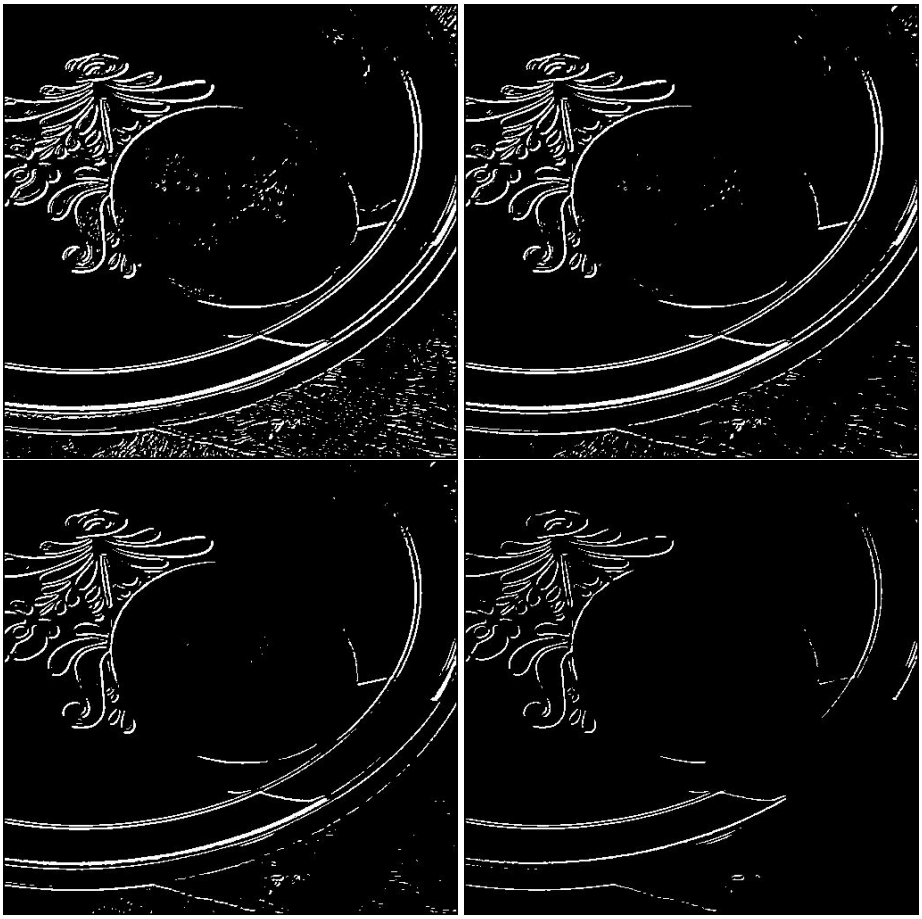

Figura 5. Da esquerda para a direita, de cima para baixo, temos a aplicação de limiar 10%, 15%, 20% e 33%.

5. Análise dos Resultados

Nosso objetivo principal era o realce e a detecção de bordas. Este objetivo foi cumprido pelos métodos aqui propostos.

Do filtro de aguçamento de bordas, os resultados obtidos com $C = 5$ nos mostram que essa é uma técnica simples que performa muito bem quando calibrada a intensidade (Figura 2).

Quanto aos dois métodos de detecção de borda, ao comparar lado a lado os melhores resultados (aqueles com menos ruídos e bordas melhores definidas) vemos que o método que faz uso do vetor diretor possui leve desvantagem (Figura 6). A quantidade de ruídos é maior e a definição das bordas está mais singela.

Figura 6. Na esquerda, detecção de borda apenas pelo limiar com base no máximo global de 5%. Na direita, detecção de borda pelo uso do vetor diretor sob limiar local de 15%.

6. Conclusão

O processo de realce se mostrou eficiente ao garantir um reforço nas bordas da imagem experimentada. Quanto aos métodos de detecção de borda, conforme apontam os experimentos, o método que considera um limiar com base no máximo global se mostrou mais eficiente para essa imagem estudada. A adição de uma regra limitante local ao procedimento que considera a direção do vetor gradiente não foi o suficiente para produzir um resultado com melhor definição.

Referências

[Solomon and Breckon 2013] Solomon, C. and Breckon, T. (2013). *Fundamentos de processamento digital de imagens*. ProQuest Ebook Central.