Visão computacional - 2020/2

Relatório PS3

Tiago Araújo Mendonça

[Código fonte]

Em todas as questões (exceto a 4) é opcional passar um parâmetro na linha de comando para especificar a imagem utilizada, se não for usado, uma imagem padrão é utilizada.

Questão 1

Nesta questão, definir o limiar T como 127. A geração da imagem binária foi realizada por meio da função cv.threshold. A partir da imagem binária, foi detectada a lista de contornos da imagem por meio da função cv.findContours.

Ao invés de solicitar que o usuário escolha a opção desejada, decidi sempre exibir a contagem de pixels pretos e brancos, além disso, a funcionalidade de clicar para ver detalhes de um polígono fica sempre disponível. Desta forma consigo atender as duas opções que o usuário poderia escolher. Ao rodar o algoritmo, primeiro é exibida a contagem de pixels, ao fechar esta imagem, a imagem com a interação é exibida, onde o usuário pode clicar para ver área, perímetro e diâmetro, para fechar esta imagem, basta teclar ESC.

A área foi calculada com a função cv.contour Área; O perímetro foi calculado com a função cv.arcLength; O diâmetro foi calculado por meio da fórmula $\sqrt{\frac{4 \times área}{\pi}}$. Estas informações são exibidas no terminal quando o usuário seleciona um objeto/área.

Questão 2

O primeiro passo nesta questão é gerar as 30 imagens suavizadas S(n) e as 30 imagens de resíduo R (n). Isso é realizado pelo seguinte esquema:

$$S^{(0)} = I$$
$$S^{(n)} = S\left(S^{(n-1)}\right) \quad \text{for } n > 0$$
$$R^{(n)} = I - S^{(n)}$$

.

O segundo passo é calcular todas as matrizes de coocorrência, que segue a seguinte formulação:

$$\mathbf{C}_I(u, v) = \sum_{p \in \Omega} \sum_{q \in A \wedge p+q \in \Omega} \begin{cases} 1 & \text{if } I(p) = u \text{ and } I(p+q) = v \\ 0 & \text{otherwise} \end{cases}$$

A adjacência para esta questão foi definida como apenas o pixel de baixo do pixel alvo, porém o código foi estruturado para que isso possa ser alterado com facilidade. A implementação da obtenção destas matrizes foi realizada de um forma um pouco diferente para otimizar o código:

1. Todas as matrizes são inicializadas com todos os valores sendo 0
2. Para cada pixel **u** da imagem:
    a. Para cada pixel **v** adjacente à **u** (e existente na imagem):
        i. Somar 1 na matriz de co-ocorrência da imagem original na posição (**u,v**)
        ii. Somar 1 em cada uma das matrizes de co-ocorrência das 30 imagens suavizadas obtidas, sempre na posição (**u,v**)
        iii. Somar 1 em cada uma das matrizes de co-ocorrência das 30 imagens de resíduos obtidas, sempre na posição (**u,v**)

O terceiro passo consiste em calcular a homogeneidade e a uniformidade de cada uma das 61 imagens, este cálculo é realizado utilizando as matrizes de co-ocorrência correspondentes.

Homogeneidade é dada por:

$$M_{hom}(I) = \sum_{u,v \in \{0,1,\ldots,G_{\max}\}} \frac{\mathbf{C}_I(u,v)}{1+|u-v|}$$

Uniformidade é dada por:

$$M_{uni}(I) = \sum_{u,v \in \{0,1,\ldots,G_{\max}\}} \mathbf{C}_I(u,v)^2$$

Por fim, são exibidos:

1. A imagem original, a imagem suavizada S(30), a imagem de resíduo R(30
2. As matrizes de co-ocorrência de cada uma das imagens do item anterior
3. A homogeneidade e uniformidade da imagem original
4. Os gráficos de homogeneidade e uniformidade distribuídos pelas 30 imagens suavizadas
5. Os gráficos de homogeneidade e uniformidade distribuídos pelas 30 imagens de ruídos

Vale lembrar que este algoritmo demora bastante a executar, por isso é exibida uma estimativa de porcentagem de conclusão no terminal. Optei por não normalizar os vetores de homogeneidade e uniformidade (exceto da imagem original) pois isso estava aumentando ainda mais o tempo de execução. Isto não é muito necessário pois a exibição dos gráficos pelo pyplot já acontece de uma forma que visualmente o resultado é muito similar ao normalizado.

Com os gráficos gerados, é fácil perceber que a homogeneidade e uniformidade aumentam quanto mais a imagem foi suavizada, o que faz sentido pois a suavização faz com que os pixels adquiram valores similares aos valores de seus vizinhos. Nas primeiras suavizações este aumento é maior, mas depois estes parâmetros crescem aos poucos.

Também podemos perceber que para as imagens de resíduos, logo nas primeiras imagens geradas existe uma queda brusca na uniformidade e homogeneidade, que segue diminuindo aos poucos para os próximos resíduos.

Questão 3

Nesta questão utilizei 127 como limiar para gerar a imagem binária; utilizei 100 como o limiar para o diâmetro de objetos aceitos na obtenção do contorno.

Primeiro o algoritmo gera a imagem binária e detecta os contornos da imagem. Em seguida, o algoritmo remove os contornos que não atingem o limiar escolhido. Os contornos restantes são exibidos com um contorno verde acima da imagem original.

Ao clicar em um contorno, são exibidos na imagem seu centróide e eixo principal, e o terminal exibe sua excentricidade. Para fechar a imagem, basta teclar ESC.

Cálculo do momento:

$$m_{a,b}(S) = \sum_{(x,y)\in S} x^a y^b \cdot I(x,y)$$

Cálculo do centróide utilizando o momento:

$$x_S = \frac{m_{1,0}(S)}{m_{0,0}(S)} \quad \text{and} \quad y_S = \frac{m_{0,1}(S)}{m_{0,0}(S)}$$

O cálculo eixo principal se baseia na fórmula abaixo para obter as coordenadas (x1, y1) e (x2, y2) que são utilizadas para exibir o eixo:

$$\tan\big(2 \cdot \theta(S)\big) = \frac{2\mu_{1,1}(S)}{\mu_{2,0}(S) - \mu_{0,2}(S)}$$

A excentricidade é calculada utilizando o momento, conforme a fórmula abaixo:

$$\varepsilon(S) = \frac{[\mu_{2,0}(S) - \mu_{0,2}(S)]^2 - 4\mu_{1,1}(S)^2}{[\mu_{2,0}(S) + \mu_{0,2}(S)]^2}$$

Questão 4

Esta questão possui duas opções para sua execução na linha de comando:

1. **-d** <valor> => define a densidade dos pixels gerados fora das linhas
2. **-n** <valor> => define o número de linhas geradas

Outros parâmetros são editáveis apenas via código, basta alterar a declaração da variável. A configuração padrão para algoritmo é a seguinte:

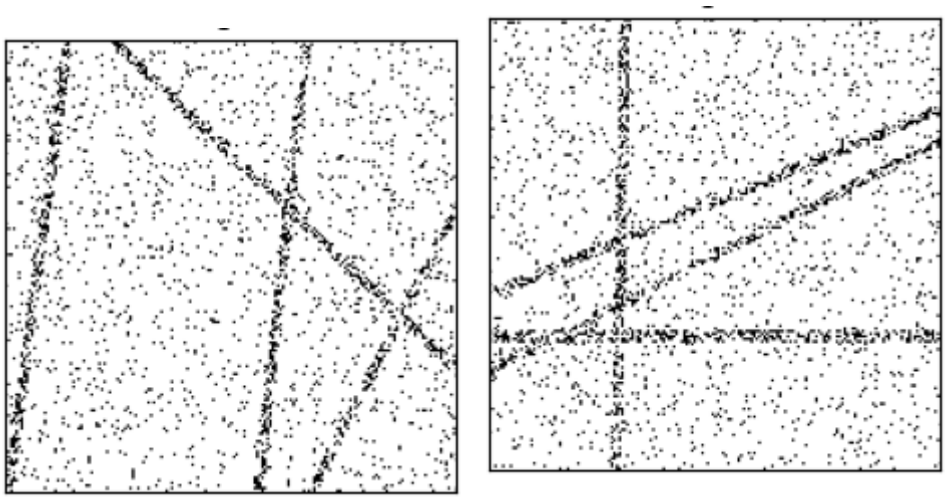
1. Densidade do plano de fundo **DF** = 5%
2. Densidade das proximidades das linhas **DL** = 40%
3. Número de linhas **N** = 4
4. Alcalce das linhas **A** = 2
5. Tamanho da imagem gerada = 175 x 175

O primeiro passo nesse algoritmo é gerar as linhas, para isso são gerados **N** pares aleatórios de pontos, que representam as retas.

O segundo passo é preencher a imagem:

1. Define inicialmente todos os pixels como pretos
2. Para cada pixel **P3** da imagem:
    a. Para cada para de pontos (**P1**, **P2**) do conjunto de linhas:
        i. Calcula a distância entre o ponto **P3** e a reta (**P1**, **P2**)
        ii. Se a distância está dentro do alcance **A**, é considerado como um “pixel de linha”
    b. Se **P3** foi considerado como um pixel de linha, tem probabilidade (1 - **DL**) de definir como branco
        i. Caso contrário, tem probabilidade (1 - **DP**) de definir como branco

Exemplos de imagens geradas:



Depois de gerar a imagem, é utilizada a função cv.HoughLinesP com limiar *threshold* 250 e com os parâmetros padrão para detecção das linhas. A imagem original é exibida ao lado da imagem que representa o resultado obtido pelo detector. Este algoritmo também é um pouco demorado, o terminal exibe o progresso estimado.