Documentação de código PSE

Grupo:

Gabriel Campos

Vítor França

Mateus Possas

Luigi Sorrentino

1 - Introdução:

A documentação está separada com base nas classes utilizadas no PSE, ou seja, cada capítulo dessa documentação representa uma classe do PSE de modo que seu conteúdo seja único e exclusivo relacionado ao seu papel e funcionamento no PSE.

Os arquivos das classes utilizadas no PSE se encontram na pasta src/processamentoimagem, todos os outros arquivos são gerados automaticamente pela IDE ou são arquivos pertencentes a bibliotecas utilizadas no desenvolvimento.

2 - CalculaDistancia

A classe CalculaDistancia, é responsável pelos cálculos de distância entre dois pontos.

Variáveis:

* private int posx1: valor de X do 1º ponto no cálculo de distância
* private int posx2: valor de X do 2º ponto no cálculo de distância
* private int posy1: valor de Y do 1º ponto no cálculo de distância
* private int posy2: valor de Y do 2º ponto no cálculo de distância

Métodos:

* CalculaDistancia(int posx1, posx2, posy1, posy2)
  + Este método é o construtor da classe, recebendo como parâmetro os valores dos pontos que terão as distâncias calculadas. Os valores das variáveis da classe são atualizadas com base nos valores fornecidos ao instanciar a classe.
* Get/Set das variáveis:
  + Para cada uma das variáveis da classe, existe um par de métodos denominados get<variavel>, set<variavel>, responsáveis respectivamente por retornar o valor da variável e definir o valor da variável com base em um valor fornecido na criação da classe. Estes métodos são necessários pelo fato das variáveis da classe são privadas.
* distEuclid(int posx1, posx2, posy1, posy2):
  + Realiza o cálculo e retorna a distância euclidiana dos pontos informados através dos parâmetros.
* distCityBlock(int posx1, posx2, posy1, posy2):
  + Realiza o cálculo e retorna a distância city block entre os pontos informados através dos parâmetros.
* distChessboard(int posx1, posx2, posy1, posy2):
  + Realiza o cálculo e retorna a distância chessboard entre os pontos informados através dos parâmetros.

3 - Gráficos:

A classe de gráficos, é responsável pela geração e exibição dos histogramas gerados pelo usuário através da utilização do java JPanel e JFrame.

Variáveis:

* javax.swing.JPanel jPanel1: jPanel utilizado para representação do histograma da imagem original
* javax.swing.JPanel jPanel2: jPanel utilizado para representação do histograma da imagem manipulada

Métodos:

* Graficos():
  + O construtor da classe realiza a chamada dos métodos initComponents() e grafico(), de modo a gerar e exibir os gráficos para o usuário
* grafico():
  + Através da conversão dos jPanels da classe para a classe Imagem, o método realiza a chamada do método atribuiImagem que permitirá a exibição dos histogramas das imagens que estão sendo calculados os gráficos (Original e manipulada), através da adaptação dos tamanhos dos gráficos para o tamanho do jPanel em que o gráfico será exibido.
* initComponents():
  + Este método é responsável por iniciar os componentes dos gráficos. Cada jPanel se torna uma instância da classe Imagem, a qual possuem a imagem original e imagem manipulada. A partir dos valores disponíveis nos jPanels com instâncias das classes de Imagem, os valores de cada imagem são calculados e atribuídos aos seus respectivos histogramas.

4 - Mergesort:

A classe mergesort, possui a implementação do algoritmo de ordenação mergesort, o qual recebe um vetor de números inteiros e realiza a ordenação dos valores contidos no vetor de forma recursiva.

Esta classe não possui nenhuma variável e a execução do algoritmo de ordenação é iniciada ao realizar chamada da função mergesort(int[] vetor, int p, r) a qual recebe o vetor a ser ordenado, bem como os valores necessários para as comparações de valores. Esta função faz a chamada de intercala1(int[] vetor, int p, q, r) que será responsável por realizar a ordenação dos valores nos intervalos calculados pelas chamadas do método mergesort.

5 - Histograma:  
 A classe histograma, como o próprio nome infere, é responsável pela criação dos histogramas da imagem, onde a partir dos parâmetros passados para a classe a mesma será responsável por criar os histogramas a serem exibidos no PSE.

Esta classe não possui variáveis locais, somente métodos. Além disso ela estende a classe ApplicationFrame.

Métodos:

* Histograma(String title, double[] vet):
  + O construtor da classe recebe como parâmetro o título do histograma ser criado e em forma de um vetor double os dados que serão utilizados na representação em forma de histograma. A partir do vetor de valores, uma variável “xySeriesCollection” é criada para que os valores possam ser utilizados como fonte de dados para o histograma. Após a geração do gráfico, o mesmo será salvo em formato “.png” para posterior utilização e exibição.
* createChart(IntervalXYDataset dataset, String title):
  + Este é o método responsável pela criação do histograma, como dito no método construtor os dados são convertidos para “xySeriesCollection”, e os mesmos são passados como parâmetro para essa função, dessa forma os dados que antes eram um vetor podem ser utilizados como os valores presentes no histograma. Além dos dados, o título também é informado. Este método cria um gráfico de barras e o retorna para o construtor, para que o mesmo o salve localmente para ser utilizado.

6 - Imagem:

A classe imagem é a classe responsável por conter todas as iterações referentes a utilização da imagem no PSE, indo desde sua exibição em um jPanel até a execução dos métodos de filtragem e segmentação da imagem. Por ser responsável em exibir as imagens no PSE, esta classe estende jPanel, para que a utilização dos seus métodos possam ser personalizados de acordo com as necessidades do programa, bem como facilitar a utilização das funções provenientes desta classe.

Variáveis:

* BufferedImage figura: está variável contém a imagem que está sendo exibida no PSE no momento (tanto original, quanto manipulada - uma instância da classe para cada)
* int IMG\_WIDTH: largura da imagem que será exibida no PSE
* int IMG\_HEIGHT: altura da imagem que será exibida no PSE
* int reset: indica se a imagem da instância foi alterada, 0 representa que a imagem continua a mesma 1 representa que houve alteração na imagem

Métodos:

* Imagem():
  + O construtor da classe possui apenas a chamada para o construtor da classe jPanel, dessa forma ao instanciar um objeto do tipo Imagem, seu construtor será o mesmo da classe jPanel.
* paintComponent(Graphics g):
  + Este método faz sobrescreve o método da classe jPanel, e será responsável por preencher o jPanel da instância com a figura que está sendo utilizada.
* atribuiImagem(BufferedImage \_buffer):
  + Este método recebe a imagem que será utilizada na instância, faz o ajuste com relação aos tamanhos do jPanel onde ocorrerá a exibição da imagem (IMG\_WIDTH, IMG\_HEIGHT). Após o ajuste de tamanho da imagem para que a mesma tenha o mesmo tamanho do jPanel, ocorre o redesenho do jPanel onde a imagem será exibida, desta forma o jPanel passará a exibir a figura passada como parâmetro.
* getImageOriginal():
  + Retorna a figura da classe
* setImageOriginal():
  + Faz com que a imagem se torne novamente a imagem original (sem as manipulações já realizadas)
* getReset():
  + Retorna o valor da variável reset, para indicar se houve ou não alteração da imagem
* setReset(int valor):
  + Define o valor da variável reset
* retornaPixel(int posx, posy):
  + A partir das coordenadas passadas, retorna o valor RGB do pixel que foi solicitado. Caso o pixel não seja válido na imagem, retorna o valor -999999.
* resizeImage(BufferedImage originalImage, int type):
  + Este é o método responsável pelo ajuste do tamanho da imagem com o jPanel que foi mencionado no método atribuiImagem. A partir da imagem original passada por parâmetro, os ajustes de tamanho são realizados de modo a evitar alteração na imagem e em seguida a imagem com novas dimensões são retornadas.
* vizinhanca4(int x, y, Color cor):
  + Este método é responsável por realizar a execução dos filtros morfológicos com a utilização do elemento estruturante de 4 vizinhos (formato de cruz). A execução deste modelo é feita de forma recursiva. Inicialmente é verificado se o pixel onde está sendo dado como centro do elemento estruturante faz parte da imagem (valores entre X e Y mínimos e máximos), com o pixel pertencente a imagem, é verificado se o mesmo possui a cor que está sendo verificada (hit), caso verdadeiro o valor do pixel é alterado e o método é chamado para seus 4 vizinhos pertencentes ao elemento estruturante ((x,y+1), (x,y-1), (x+1,y),(x-1,y). O elemento estruturante percorre toda a imagem, colocando todos os pixels da imagem no centro do elemento, ou seja, todos os pixels da imagem são analisados com o elemento estruturante.

Elemento estruturante:

1

1x1, onde x é o pixel que está sendo analisado

1

* vizinhanca8(int x, y, Color cor):
  + Este método possui funcionamento similar ao método vizinhanca4. Ele difere com relação ao tamanho do elemento estruturante, que neste caso leva em conta todos os 8 vizinhos do pixel. Sua execução também é feita recursivamente. O elemento estruturante percorre toda a imagem, colocando todos os pixels da imagem no centro do elemento, ou seja, todos os pixels da imagem são analisados com o elemento estruturante.

Elemento estruturante:

111

1x1, onde x é o pixel que está sendo analisado

111

* escalaCinza():
  + Este é o método responsável por transformar a imagem RGB em escala de cinza. É feita verificação de cada pixel da imagem, e a cada iteração é calculado o novo valor em escala de cinza do pixel que está sendo analisado. O valor da escala de cinza é calculado de acordo com a média de valores de intensidade de RGB, ou seja, as intensidades de cada uma das 3 cores são somadas e o seu resultado é dividido por 3, gerando assim, o tom de cinza daquele pixel. O processo de alteração de escala de cinza é feito para todos os pixels da imagem de maneira individual, ou seja, cada pixel terá sua média extraída e definida como nova intensidade em tons de cinza.
* negativo():
  + A função atribuída a este método é transformar os pixels da imagem em seus valores negativos da seguinte forma:
    - os valores RGB são extraídos individualmente, e novos valores são calculados. O cálculo consiste em subtrair de 255(intensidade máxima) o valor de cada elemento RGB. Ou seja, o valor de R passará a ser 255 - R, G será 255 - G e B será 255 -B. Ao final do cálculo do valor de cada elemento RGB, uma nova cor é gerada com a junção dos resultados e atribuída ao pixel da imagem. Este processo de alteração de valores de pixel é realizado para todos os pixels pertencentes a imagem.
* sliderColorVermelho(int valor):
  + Na interface do PSE, temos alguns sliders que permitem ao usuário fazer alterações com a movimentação do slider. Este método é responsável pela ação decorrente da movimentação do slider. De acordo com o valor atual presente no slider, todos os pixels da imagem tem seu valor de vermelho transformados no valor passado para a função, dessa forma manipulando apenas os valores de vermelho. O vermelho é representado pelo R em RGB. O processo de alteração de valores do vermelho é realizado em todos os pixels da imagem.
* sliderColorVerde(int valor):
  + Assim como a função sliderColorVermelho, esta função é responsável pela alteração dos níveis de verde de todos os pixels da imagem, com base no valor presente no slider que representa a cor verde. Ou seja, este método irá alterar o valor da intensidade de verde de cada pixel da imagem para o valor passado por parâmetro. O verde é representado pelo G em RGB. O processo de alteração do valor de verde é realizado em todos os pixels da imagem.
* sliderColorAzul(int valor):
  + Para completar a manipulação individual dos elementos de RGB, temos a utilização do slider que contém a intensidade do valor do azul presente em cada pixel da imagem. Assim como os outros métodos, este fará a alteração dos níveis de azul da imagem para o valor informado através do valor do slider. O azul é representado pelo B no RGB. O processo de alteração dos valores de azul é realizado em todos os pixels da imagem.
* binarizacao(int valor):
  + O método de binarização, realiza a segmentação da imagem para apenas 2 cores (preto ou branco) a partir de um valor passado por parâmetro. O valor passado como parâmetro, funciona como um limiar para a atribuição das cores, ou seja, é o valor que será utilizado para decidir se determinado pixel é branco ou preto. Cada pixel possui seu tom de cinza calculado da mesma forma como explicado no método escalaCinza: faz-se a média de valores do RGB. Com o tom de cinza do pixel calculado, é verificado se o pixel possui valor menor que o valor de limiar, caso possua seu valor se tornará 0 (se tornará preto), caso contrário seu valor será 255 (se tornará branco). Este processo é realizado em todos os pixels da imagem
* brilho(int valor):
  + Além dos sliders de cores presentes na interface, também consta o slider de brilho da imagem, onde é feita a alteração dos valores de brilho da imagem de acordo com o valor presente no slider. Os valores RGB são extraídos individualmente, e aos seus valores é acrescentado o valor de brilho passado pelo slider. Este cálculo é feito para cada pixel da imagem.
* histograma():
  + A partir dos valores de RGB da imagem, este método calcula os valores dos pixels em tons de cinza e retorna um array com a quantidade de pixels que cada tom de cinza possui. Este array retornado será utilizado posteriormente para a criação de histograma com base na quantidade de pixels de cada tom de cinza.
* calculoHist(double[] vet):
  + Este método realiza o cálculo de probabilidade do valor de cada pixel da imagem com base no valor do pixel em relação a quantidade total de pixels existentes na imagem.
* calculoHistAcumulado(double[] vet):
  + Assim como o método anterior, este método realiza o cálculo de probabilidade dos pixels com base nos pixels totais da imagem, com a diferença de que a probabilidade é calculada de maneira acumulada, levando em conta o pixel anterior a ele.
* calculoAcumulado(double[] vet):
  + Este método realiza o somatório dos valores dos vetores de cor da imagem, tendo o valor total dos valores RGB. O cálculo é feito de forma separada, para se obter o valor dos 3 são necessárias 3 chamadas, onde o vetor a ser somado é passado por parâmetro.
* verificaPosicao(int valorAntigo, int valorNovo, double[][] pixels):
  + Este método realiza a alteração de valores de pixel com base nos cálculos da equalização. É verificado se o pixel possui o valor antigo que está sendo buscado, caso possua o novo valor de pixel é atribuído para o mesmo.
* equalizar():
  + Este método é responsável pela equalização da imagem, bem como do seu histograma. O cálculo da equalização é feito utilizando os métodos de somatório anteriores, bem como os cálculos de histograma. A partir dos valores gerados pelos métodos anteriores, é verificada a proporção de cada um dos valores RGB para que a equalização possa ser feita. Com a proporção de cada valor da imagem, os novos valores de pixel da imagem são calculados para que a imagem e seu histograma sejam equalizados. Após o cálculo, os valores de pixel são alterados com a utilização do método verificaPosicao. Ao fim das atribuições de valores, o vetor contendo os valores a serem utilizados para realizar a representação da imagem por histograma é calculado retornado.
* filtroMedia(int tamanho):
  + O filtro de média é implementado nessa função, onde a partir do tamanho da máscara passada para o método a média da região dentro da máscara é calculada e o valor do pixel que está sendo analisado é alterado para o valor da média de sua região. O cálculo dos novos valores dos pixels é feito com base no tamanho da máscara utilizada podendo ser chamado os métodos para tamanho par ou ímpar. Após os cálculos dos novos valores com os métodos corretos (par ou ímpar), os valores de dos pixels são atualizados.
* colorePixelParMedia(int i, j, tamanho, int[][] matriz):
  + Este método realiza o cálculo da média bem como o novo valor dos pixels da imagem, a partir da análise da média presente na máscara utilizada.
* colorePixelImparMedia(int i, j, tamanho, int[][] matriz):
  + Assim como o método anterior, este é responsável por fazer os cálculos dos valores dos pixels da imagem com base na média da máscara utilizada, porém este método é utilizado quando a máscara de filtro possui tamanho ímpar.
* filtroMediana(int tamanho):
  + O filtro de mediana é aplicado com a utilização deste método que recebe como parâmetro o tamanho da máscara a ser utilizada na filtragem. Dado o tamanho da máscara será definido se o método a se utilizar para o cálculo e atualização dos valores será com método par ou ímpar. Após os cálculos de novos valores do pixel da imagem, os valores na imagem original são alterados com base no que foi gerado.
* colorePixelParMediana(int i, j, tamanho, int[][] matriz):
  + Possui a mesma função do método colorePixelParMedia, porém utilizando como métrica para novos valores a função de mediana.
* colorePixelImparMediana(int i, j, tamanho, int[][] matriz(:
  + Possui a mesma função do método colorePixelImparMedia, porém utilizando como métrica para novos valores a função de mediana.
* filtroSobel():
  + O filtro sobel é aplicado utilizando esta função, onde todos os pixels tem seus novos valores calculados de acordo com as métricas de sobel e após este cálculo os pixels originais possuem seus valores atualizados.
* colorePixelSobel(int i, j, int[][] matriz, matrizHorizontal, matrizVertical):
  + Este método realiza a filtragem utilizando Sobel, onde são analizadas as matrizes horizontais e verticais da imagem para que o realce da borda ocorra nas duas direções. Os valores RGB de cada pixel são verificados horizontalmente e verticalmente de forma independente. Com base nos valores encontrados é verificado se o pixel será atualizado com valor 0 ou 1 da seguinte forma: menor que 0 será representado por valor 0 e maior que 255 será representado com valor 255. Esta verificação é realizada para as 3 cores RGB de modo independente, e após a definição (0 ou 255) a nova cor do pixel é atribuída com os novos valores de RGB em conjunto.
* filtroLaplaciano():
  + Assim como os outros métodos de filtragem, este realiza as chamadas para os cálculos dos novos valores de pixel com base na lógica de filtro laplaciano, e depois dos novos valores calculados os pixels originais da imagem tem seus valores atualizados.
* colorePixelLaplaciano(int i, j, int[][] matriz, int[][] matrizCalculo):
  + Com os valores de cada pixel da imagem, são extraídos os valores RGB e verificado o entorno com base na máscara do filtro laplaciano. A partir do valor gerado com a utilização da máscara, é feita a verificação de qual o valor (0 ou 255) de cada elemento RGB será definido. Após atribuição dos valores para RGB separadamente é feita a união dos valores para definição do novo valor RGB do pixel analisado.
* filtroKNN(int tamanho, int k):
  + Assim como os outros métodos de filtragem, este tem o intuito de fazer o agrupamento das funções relacionadas ao filtro KNN. Primeiramente são calculados os valores dos pixels com base no tamanho da máscara que será utilizada, caso a máscara possua tamanho par executará o método para cálculo do valor de tamanho par, caso seja ímpar o método para cálculo do valor de tamanho ímpar. Finalizados os cálculos para os novos valores de todos os pixels, os pixels da imagem original são atualizados para os novos valores calculados.
* colorePixelParKNN(int i, j, tamanho, int[][] matriz, int vizinhos):
  + Este método faz os cálculos referentes aos novos valores dos pixels com a realização da filtragem KNN. A partir dos valores dos pixels originais, do tamanho da máscara e dos vizinhos, são extraídos os valores RGB dos pixels de modo a realizar o cálculo das novas intensidades de cada um com base nos vizinhos.
* colorePixelImparKNN(int i, j, tamanho, int[][] matriz, int vizinhos):
  + Este método possui funcionamento similar ao método anterior, porém é utilizado quando o tamanho da máscara é ímpar.
* verifica(int [] vetor, int valor):
  + Realiza os cálculos para os valores de intensidade dos pixels a partir dos vizinhos no método de filtragem KNN.
* histogarmaOtsu():
  + Realiza o cálculo para transformação da imagem em RGB para tons de cinza a partir da média dos valores de RGB. Retorna um vetor contendo a quantidade de pixels de cada tom de cinza calculado.
* otsu():
  + Realiza a aplicação da filtragem da segmentação de Otsu. Utiliza-se o méetodo histogramaOtsu() para os cálculos das quantidades de pixels presentes em cada tom de cinza presente na imagem, com estes valores o algoritmo de Otsu é aplicado para a segmentação da imagem.
* globalSimples():
  + Realiza a limiarização global simples da imagem, utilizando 127 como limiar inicial da imagem, separando a imagem em 2 grupos distintos de pixel. Após a primeira separação, um novo limiar é calculado com base nas médias dos dois grupos, este processo se repete até que o módulo da diferença entre o limiar novo com o antigo seja maior que zero. Finalizada a definição do limiar, os pixels com valores menores que o limiar calculado são definidos com valor em tom de cinza = 0 (preto), enquanto os pixels com valores maiores ou iguais com valor = 255 (branco).

7 - ProcImagem:

A classe ProcImagem é responsável pela lógica agregada a interação do usuário com a interface do PSE, onde todos os clicks de botões e ações referentes a utilização da interface são tratados. Devido a quantidade de métodos ser muito grande e de lógica padronizada, teremos abaixo um exemplo de como funciona o clique no botão ou elemento da imagem e a execução de seu método equivalente.

1. O usuário clica no botão
2. São feitas verificações para cada tipo de botão clicado pelo usuário
   1. Ex: Ao fazer a modificação dos valores dos sliders é verificado se de fato a imagem se encontra no PSE.
3. Realizada a chamada da função referente a ação solicitada pelo usuário, passando os parâmetros solicitados
   1. Ex: Ao mover o slider de brilho, é chamada a função de brilho descrita no capítulo 6 passando o valor do slider de brilho.

.