

StefannusChristian / Computer-Vision

<> Code Issues Pull requests Actions Projects Wiki Security Insights Set

Computer-Vision / Materi-UTS / README.MD

StefannusChristian

finished vertical bars

1 minute ago

616 lines (512 loc) · 23.9 KB

PreviewCodeBlame

Raw

Image J Plugins / Codes

In ImageJ-plugins it is important that the last character of the filename is an underscore “_”! It is also important that the internal name of the plugin is the same as the filename except of the extension “.java”. This internal name is specified in the source code line, which starts with public class.

Latihan 5_2

Example0_.java

```
// Import necessary statements
import ij.*;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;
import ij.gui.*;

// Define the class
public class Example0_ implements PlugInFilter {

    // The setup method sets up the plugin and defines what it will do
    public int setup(String arg, ImagePlus imp) {
        // The plugin will operate on the entire image
        return DOES_ALL;
    }

    // The run method contains the main functionality of the plugin
    public void run(ImageProcessor ip) {
        // putPixel parameters (x coordinate, y coordinate, intensity)
        // This line writes a white pixel at coordinates (10, 10)
        ip.putPixel(10, 10, 255);
    }
}
```

Short White Line (Exercise0202_.java)

Creates a short white line from pixel (30, 10) to pixel (39, 10).

public void run(ImageProcessor ip) {
 // Starting x-coordinate for the line
 int x = 30;

 // Loop 10 times to draw a white line of length 10 pixels
 for (int i = 0; i < 10; i++) {
 // Set the pixel at coordinates (30 + i, 10) to white (intensity 255)
 ip.putPixel(30 + i, 10, 255);
 }
}

Loop in one direction (Example0_.java)

Draw a white line with a length of 250 pixels at y-coordinate 50 in the image.

// Loop through 250 x-coordinates to create the line
for (int x = 0; x < 250; x++) {
 // Set the pixel at coordinates (x, 50) to white (intensity 255)
 ip.putPixel(x, 50, 255);
}

Vertical Line Of 250 Pixels

Draws a vertical white line in the image.

public void run(ImageProcessor ip) {
 // Loop through 250 y-coordinates to create the vertical line
 for (int x = 0; x < 250; x++) {
 // Set the pixel at coordinates (50, x) to white (intensity 255)
 ip.putPixel(50, x, 255);
 }
}

Latihan 6_1

Using the Image Dimensions

To draw a line across the whole image, you need to know the image's width and height. Use the getWidth() and getHeight() methods, and store the dimensions in local variables before using them.

// Get the width and height of the image from the ImageProcessor
int width = ip.getWidth(); // Width of the image
int height = ip.getHeight(); // Height of the image

Vertical Middle Line

Draws a vertical white line from top to bottom of the image, with its horizontal position in the middle of the image.

// Get the width and height of the image
int width = ip.getWidth(); // Width of the image
int height = ip.getHeight(); // Height of the image

```
// Calculate the horizontal center of the image
int centerHorizontal = width / 2;

// Loop through the height of the image to draw a vertical line
for (int y = 0; y < height; y++) {
    // Set the pixel at the horizontal center (centerHorizontal, y) to white (intensity 255)
    ip.putPixel(centerHorizontal, y, 255);
}
```

Exercise: Black Top

Fills the upper half of the image with zero intensity (black).

```
// Get the width and height of the image
int width = ip.getWidth(); // Width of the image
int height = ip.getHeight(); // Height of the image

// Loop through the top half of the image
for (int y = 0; y < height / 2; y++) {
    // Loop through the entire width of the image
    for (int x = 0; x < width; x++) {
        // Set the pixel at coordinates (x, y) to black (intensity 0)
        ip.putPixel(x, y, 0);
    }
}
```

Exercise: Black Square

Draws a black box in the middle of the current image. The width and height of the box shall be 50% of the width and height of the image.

```
// Get the width and height of the image
int width = ip.getWidth(); // Width of the image
int height = ip.getHeight(); // Height of the image

// Loop through the vertical range of the middle square (from 25% to 75% of height)
for (int y = height / 4; y < (height * 3 / 4); y++) {
    // Loop through the horizontal range of the middle square (from 25% to 75% of width)
    for (int x = width / 4; x < (width * 3 / 4); x++) {
        // Set the pixel at coordinates (x, y) to black (intensity 0)
        ip.putPixel(x, y, 0);
    }
}
```

Reading intensities from the image

So far, we have only written intensities to pixels. However, it is of course also very relevant to read existing intensities from (input) pixels. This is done by the method getPixel: `v=ip.getPixel(x,y);` This command stores the intensity, which is present at the pixel (x,y) in the variable v.

Inverted Image

Inverts the current image: `Iout = 255 – Iin` (White input-pixels will be turned into black, black input pixels will be turned into white, light pixels will become dark and dark pixels will become light.)

```
// Get the width and height of the image
int width = ip.getWidth(); // Width of the image
int height = ip.getHeight(); // Height of the image

int v, Iout, Iin; // Variables to store pixel values and intensity

// Loop through all rows of the image
for (int y = 0; y < height; y++) {
    // Loop through all columns of the image
    for (int x = 0; x < width; x++) {
        // Get the intensity (pixel value) at the current pixel
        v = ip.getPixel(x, y);
        // Calculate the inverted intensity
        Iin = v; // Input intensity
        Iout = 255 - Iin; // Inverted intensity
        // Set the pixel at coordinates (x, y) to the inverted intensity
        ip.putPixel(x, y, Iout);
    }
}
```

Top left quarter copied

Copies the top left quarter of the image to the three other quarters.

```
// Get the width and height of the image
int width = ip.getWidth(); // Width of the image
int height = ip.getHeight(); // Height of the image
int v, Iout, Iin; // Variables to store pixel values and intensity

// Loop through all rows of the image
for (int y = 0; y < height; y++) {
    // Loop through all columns of the image
    for (int x = 0; x < width; x++) {
        // Get the intensity (pixel value) at the current pixel
        v = ip.getPixel(x, y);

        // Copy the intensity value to the other three quarters of the image
        // First quarter: original location (x, y)

        // Second quarter: move right by half the width (x + width/2, y)
        ip.putPixel(x + (width / 2), y, v);

        // Third quarter: move down by half the height (x, y + height/2)
        ip.putPixel(x, y + (height / 2), v);

        // Fourth quarter: move right and down (x + width/2, y + height/2)
        ip.putPixel(x + (width / 2), y + (height / 2), v);
    }
}
```

Duplicated in a mirror [↗](#)

Mirrors the upper half of the current image to the lower half.

```
// Get the width and height of the image
int width = ip.getWidth(); // Width of the image
int height = ip.getHeight(); // Height of the image
int Yout, Yin, v; // Variables to store pixel values and Y-coordinates

// Loop through the top half of the image (up to height/2)
for (int y = 0; y < (height / 2); y++) {
    // Loop through all columns of the image
    for (int x = 0; x < width; x++) {
        // Get the intensity (pixel value) at the current pixel
        v = ip.getPixel(x, y);

        // Calculate the new Y-coordinate (Yout) for the mirrored pixel in the bottom half
        Yin = y;
        Yout = height - Yin;

        // Copy the pixel value to the corresponding position in the bottom half
        ip.putPixel(x, Yout, v);
    }
}
```

Black Pixels To White [↗](#)

Turns the black pixels (intensity=0) into white pixels (intensity=255) and leaves all other pixels unchanged.

```
// Get the width and height of the image
int width = ip.getWidth(); // Width of the image
int height = ip.getHeight(); // Height of the image
int Yout, Yin, v, black, white; // Variables to store pixel values and Y-coordinates

// Loop through all rows of the image
for (int y = 0; y < height; y++) {
    // Loop through all columns of the image
    for (int x = 0; x < width; x++) {
        // Get the intensity (pixel value) at the current pixel
        v = ip.getPixel(x, y);

        // Check if the pixel is black (intensity=0)
        if (v == 0) {
            // If black, turn it into white (intensity=255)
            ip.putPixel(x, y, 255);
        } else if (v == 255) {
            // If white, turn it into black (intensity=0)
            ip.putPixel(x, y, 0);
        }
    }
}
```

Counting Pixels [↗](#)

Outputs the number of pixels with zero intensity and the number of pixels with intensity larger or equal than 200.

```
// Initialize counters for zero intensity and high intensity pixels
int n = 0; // Counter for zero intensity pixels
int m = 0; // Counter for high intensity (greater than or equal to 200) pixels
int v; // Variable to store pixel intensity

// Get the width and height of the image
int width = ip.getWidth(); // Width of the image
int height = ip.getHeight(); // Height of the image

// Loop through all rows of the image
for (int y = 0; y < height; y++) {
    // Loop through all columns of the image
    for (int x = 0; x < width; x++) {
        // Get the intensity (pixel value) at the current pixel
        v = ip.getPixel(x, y);

        // Check if the pixel has zero intensity
        if (v == 0) {
            // Increment the zero intensity counter
            n = n + 1;
        } else if (v >= 200) {
            // Check if the pixel has high intensity (greater than or equal to 200)
            // Increment the high intensity counter
            m = m + 1;
        }
    }
}

// Show a message with the count of zero intensity pixels
IJ.showMessage("n=" + n + " pixels.");

// Show a message with the count of high intensity pixels
IJ.showMessage("m=" + m + " pixels.");
```

Latihan 6_2 [↗](#)

Definition of variables (Data Types) [↗](#)

In the beginning, we will only use integer and floating points variables. Integer variables are defined by int, floating point variables by double, e.g.

```
int start, end;
double average;
```

2.22 Mathematical functions [↗](#)

Mathematical functions are called from the Math-class, e.g.

```
Double u;
```

```
// Square-Root
u=Math.sqrt( 2.4 );

// Sine-function in radians!!
// Analogously: Math.cos, Math.tan
u=Math.sin( 3.14 );

// The circle-number Pi defined as a constant
u=Math.PI;
```

Exercise: Circle

Draws a white circle into the current image. The circle shall have its center at the center of the image, and its radius shall be a quarter of the image diagonal.

```
// Get the width and height of the image
double width = ip.getWidth(); // Width of the image
double height = ip.getHeight(); // Height of the image

// Calculate the coordinates of the center of the image
double xo = width / 2; // x-coordinate of the center
double yo = height / 2; // y-coordinate of the center

// Calculate the radius of the circle (one-quarter of the image diagonal)
double diagonal = Math.sqrt(Math.pow(width, 2) + Math.pow(height, 2)); // Diagonal of the im
double r = diagonal / 4; // Radius of the circle
// By dividing the diagonal by 4, you are effectively choosing a radius that is one-quarter

// Set the intensity (color) of the circle (255 = white)
int intensity = 255; // White color

// Initialize variables for x and y coordinates
double x, y;

// Define the constant pi
double pi = Math.PI;

// Initialize the angle (winkel) to 0
double winkel = 0;

// Loop to draw the circle points
while (winkel <= 2 * pi) {
    // Calculate the x and y coordinates of the circle point
    x = xo + r * Math.cos(winkel);
    y = yo + r * Math.sin(winkel);

    // Convert the double coordinates to integer for pixel placement
    int intx = (int) x;
    int inty = (int) y;

    // Set the pixel at the calculated coordinates to the specified intensity (color)
    ip.putPixel(intx, inty, intensity);

    // Increment the angle to draw the next point (the smaller the increment the faster the
    winkel = winkel + 0.001; // A small step along the circle's circumference
}
```

Maximum Intensity

Searches the maximum intensity of an image.

```
// Initialize the maximum intensity to a value that's guaranteed to be lower than the actual
maxIntensity = 0;
for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
        // Get the intensity value at pixel (x, y)
        v = ip.getPixel(x, y);

        // If the current intensity (v) is greater than the current maximum (maxIntensity),
        if (maxIntensity < v) {
            // update the maximum intensity
            maxIntensity = v;
        }
    }
}

// Display a message box with the calculated maximum intensity value
IJ.showMessage("Maximum intensity=" + maxIntensity);
```

Maximum Search

Find the maximum intensity in an image, locate the coordinates with the maximum intensity, and then draw a white rectangle that surrounds the area containing the maximum intensity.

```
int maxIntensity = -1000; // Initialize the maximum intensity to a value lower than any pote
int width = ip.getWidth(); // Get the width of the image
int height = ip.getHeight(); // Get the height of the image
int v; // Variable to store pixel intensity
int max_x = -1000; // Initialize max_x to a value lower than any x-coordinate
int min_x = width + 1000; // Initialize min_x to a value higher than the image width
int max_y = -1000; // Initialize max_y to a value lower than any y-coordinate
int min_y = height + 1000; // Initialize min_y to a value higher than the image height

// Loop through the image to find the maximum intensity value
for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
        v = ip.getPixel(x, y); // Get the intensity of the current pixel
        if (maxIntensity < v) { // If the current intensity is greater than the current maxi
            maxIntensity = v; // Update the maximum intensity
        }
    }
}

// Loop through the image again to find the coordinates with the maximum intensity
for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
        v = ip.getPixel(x, y); // Get the intensity of the current pixel
        if (v == maxIntensity) { // If the intensity matches the maximum intensity
            if (max_x < x) { // Update max_x if the current x-coordinate is greater
                max_x = x; // Store the maximum x-coordinate
            }
            if (min_x > x) { // Update min_x if the current x-coordinate is smaller
                min_x = x; // Store the minimum x-coordinate
            }
        }
    }
}
```

```
        if (max_y < y) { // Update max_y if the current y-coordinate is greater
            max_y = y; // Store the maximum y-coordinate
        }
        if (min_y > y) { // Update min_y if the current y-coordinate is smaller
            min_y = y; // Store the minimum y-coordinate
        }
    }
}

// Print the Results
IJ.showMessage("The maximum intensity is: " + maxIntensity);
IJ.showMessage("The largest x-coordinate with maximum intensity is: " + max_x);
IJ.showMessage("The smallest x-coordinate with minimum intensity is: " + min_x);
IJ.showMessage("The largest y-coordinate with maximum intensity is: " + max_y);
IJ.showMessage("The smallest y-coordinate with minimum intensity is: " + min_y);

// Draw a white rectangle using the calculated coordinates
for (int i = min_x; i <= max_x; i++) {
    ip.putPixel(i, min_y, 255); // Set pixels in the top edge to white
    ip.putPixel(i, max_y, 255); // Set pixels in the bottom edge to white
}
for (int j = min_y; j <= max_y; j++) {
    ip.putPixel(min_x, j, 255); // Set pixels in the left edge to white
    ip.putPixel(max_x, j, 255); // Set pixels in the right edge to white
}
```

Diagonal

Draws a white diagonal line starting from the top left corner, moving one row below and one column right at each step (works for quadratic images as well as for rectangular images, where one image dimension is smaller than the other).

```
int min_x = 0; // Initialize the x-coordinate of the pixel
int min_y = 0; // Initialize the y-coordinate of the pixel
int height = ip.getHeight(); // Get the height of the image

for (int y = 0; y <= height; y++) {
    ip.putPixel(min_x, min_y, 255); // Set the pixel at (min_x, min_y) to white (intensity 2)
    min_x++; // Increment the x-coordinate
    min_y++; // Increment the y-coordinate
}
```

Vertical Bars

Fills the current image with vertical bars, which have a width of 50 pixels. The first bar shall have intensity 0, the second bar shall have intensity 10, the next one intensity 10, and so on. The last bar will probably not have the full width of 50 pixels, because the image width is not always a multiple of 50.

```
int width = ip.getWidth(); // Get the width of the image.
int height = ip.getHeight(); // Get the height of the image.
int width_bar = 50; // Define the width of each vertical bar.

int i = 0, j = 0, v = 0; // Initialize counters and intensity value.
```

```
for (int x = 0; x < height; x++) {
    for (int y = 0; y < width; y++) {
        ip.putPixel(x, y, v); // Set the pixel at (x, y) to the determined intensity (v).
    }

    if (j == 50) {
        i++; // Increment the row counter.
        j = 0; // Reset the column counter.
        if (i % 2 == 0) {
            v = 0; // If the row is even, set intensity to 0 (black).
        }
        else {
            v = 255; // If the row is odd, set intensity to 255 (white).
        }
    }
    j += 1; // Increment the column counter.
}
```

Vertical Bars of Growing Width

Fills the first column of the current image with bars of increasing intensity.

```
int height = ip.getHeight(); // Get the height of the image.
int width = ip.getWidth(); // Get the width of the image.

int intensityImprovement = 5; // Define the intensity improvement for each column.

int xStart, xEnd, intensity; // Declare variables for the starting x-coordinate, ending x-c

int j = 0; // Initialize a variable j for controlling intensity.
int columnCount = 1; // Initialize a variable to count columns.

int i = 0; // Initialize a variable i to control the loop.

while (i < width) { // Start a loop that continues until i is less than the image w

    columnCount++; // Increment the column count.

    xStart = i; // Set the starting x-coordinate.
    xEnd = i + columnCount - 1; // Calculate the ending x-coordinate based on the column co

    intensity = j * intensityImprovement; // Calculate the intensity for this column.


    for (int x = xStart; x < xEnd; x++) { // Loop through the x-coordinates of the current
        for (int y = 0; y < height; y++) { // Loop through the y-coordinates of the image.
            ip.putPixel(x, y, intensity); // Set the pixel at (x, y) to the calculated int
        }
    }

    j++; // Increment the intensity control variable.
    i = xEnd; // Update i to the new ending x-coordinate for the next column.

}
```

Chessboard 3x3

Fills the current image with a 3x3 chessboard pattern of alternating black and white rectangles.



```

int height = ip.getHeight(); // Get the height of the image.
int width = ip.getWidth(); // Get the width of the image.

int rectangleHeight = height / 3; // Calculate the height of each rectangular region (1/3 of
int rectangleWidth = width / 3; // Calculate the width of each rectangular region (1/3 of

int intensityBlack = 0; // Define the intensity value for black.
int intensityWhite = 255; // Define the intensity value for white.
int intensity = intensityWhite; // Initialize intensity with white.
int sum; // Variable to keep track of the sum of i and j, used for alter

int xEnd; // Variable to store the end x-coordinate of the current rectangular region.
int yEnd; // Variable to store the end y-coordinate of the current rectangular region.

for (int j = 0; j < 3; j++) { // Iterate over rows of rectangular regions.
    int xStart = 0;
    int yStart = 0 + j * rectangleHeight; // Calculate the starting y-coordinate of the cur

    for (int i = 0; i < 3; i++) { // Iterate over columns of rectangular regions.
        sum = i + j;

        // Check if the sum of i and j is even or odd to alternate between black and white.
        intensity = intensityWhite;
        if (sum % 2 != 0) {
            intensity = intensityBlack;
        }

        xEnd = xStart + rectangleWidth; // Calculate the end x-coordinate of the current re
        yEnd = yStart + rectangleHeight; // Calculate the end y-coordinate of the current r


        // Nested loops to fill the rectangular region with the determined intensity.
        for (int x = xStart; x < xEnd; x++) {
            for (int y = yStart; y < yEnd; y++) {
                ip.putPixel(x, y, intensity); // Set the pixel at (x, y) to the determined
            }
        }

        xStart = xEnd; // Update the starting x-coordinate for the next rectangular region.
    }
}

```

Chessboard Inversion

3x3 chessboard pattern where one group of rectangles shows the inverse image, and the other group shows the original image.



```

int height = ip.getHeight(); // Get the height of the image.
int width = ip.getWidth(); // Get the width of the image.
int v, Iout, Iin; // Variables for pixel intensity calculations.

int intensityBlack = 0; // Define the intensity value for black.
int intensityWhite = 255; // Define the intensity value for white.
int intensity = intensityWhite; // Initialize intensity with white.

int rectangleHeight = height / 3; // Calculate the height of each rectangular region (1/3 of
int rectangleWidth = width / 3; // Calculate the width of each rectangular region (1/3 of

int sum; // Variable to keep track of the sum of i and j, used for alternatin

```

```

int xEnd; // Variable to store the end x-coordinate of the current rectangular region.
int yEnd; // Variable to store the end y-coordinate of the current rectangular region.

for (int j = 0; j < 3; j++) { // Iterate over rows of rectangular regions.
    int xStart = 0;
    int yStart = 0 + j * rectangleHeight; // Calculate the starting y-coordinate of the cur

    for (int i = 0; i < 3; i++) { // Iterate over columns of rectangular regions.
        sum = i + j;

        // Check if the sum of i and j is even or odd to alternate between black and white.
        intensity = intensityWhite;
        if (sum % 2 != 0) {
            // If sum is odd, invert the pixel intensities (black to white and vice versa).
            for (int y = 0; y < height; y++) {
                for (int x = 0; x < width; x++) {
                    v = ip.getPixel(x, y);
                    Iin = v;
                    Iout = 255 - Iin;
                    intensity = Iout;
                    // ip.putPixel(x, y, Iout); // This line is commented out, so it doesn't

                }
            }

            xEnd = xStart + rectangleWidth; // Calculate the end x-coordinate of the current re
            yEnd = yStart + rectangleHeight; // Calculate the end y-coordinate of the current r

            // Nested loops to fill the rectangular region with the determined intensity.
            for (int x = xStart; x < xEnd; x++) {
                for (int y = yStart; y < yEnd; y++) {
                    ip.putPixel(x, y, intensity); // Set the pixel at (x, y) to the determined
                }
            }

            xStart = xEnd; // Update the starting x-coordinate for the next rectangular region.
        }
    }
}

```