CS161: Introduction to Computer Science
Lab Assignment 10

Image Processor

In this lab, we'll be working with images! Images can naturally be represented using a 2-dimensional array of Pixel (short for *picture element*) values. This lab asks you to implement various image-transformation algorithms such as inverting the pixels in the image or rotating the image clockwise.

We'll be using an image format known as PPM (portable pixel map). PPM files are not as well-known as other file formats such as jpeg, gif, etc. However, PPM is an excellent educational format because PPM files are human-readable. The tradeoff is that even small images can be very large in size. To view a PPM file, you will need a special program:

- If you're working on a lab computer in Thompson 409, then the tools are already installed for you to open a `.pgm` file. If you're on Windows, double-clicking on any `.pgm` file will open up the file in Libre OfficeDraw.
- If you're working on your own computer, please download GIMP *before coming to lab*. You can download it for free at `http://www.gimp.org/downloads`. It is compatible with Mac, Windows, and Linux. After you install GIMP, double-clicking on any `.pgm` file will open up the file in GIMP.

For this lab, you'll be opening and reading in an image, transforming the image somehow (e.g. rotating it), and then saving the transformed image to a new file. To view the result of your transformation, you will need to double-click on the new file you created – i.e., you will not be able to view the transformed file in BlueJ but instead will manually open the file by double-clicking.

Getting Started

Download the starter code for this lab. After you unzip it, you should see the following files:

- `test.pgm, test2.pgm, test3.pgm`: Three small images for debugging.
- `grumpycat.pgm`: An image of Grumpy Cat.
- `nasa.pgm`: A washed-out aerial image from NASA.
- `Image.java`: This is the Image class, which represents a greyscale image and has methods to manipulate the image. There are several methods left for you to implement.
- `Controller.java`: Put code here to test the methods in the `Image` class.

Double click on `grumpycat.pgm` and `nasa.pgm` to make sure you can open `.pgm` images.

The `Pixel` Class

We'll be working with black and white images where each pixel is a single integer in the range $[0, 255]$. A pixel value of 0 is black and a pixel value of 255 is white. As the pixel value ranges from 0 to 255 the color changes from black, to gray, to white.

Create a new Java class called `Pixel`. The `Pixel` class should contain the following methods:

- A constructor that takes in an integer pixel value.
- A `getValue()` and `setValue()` method for the pixel value. (These are accessor and mutator methods).
- The following methods transform a pixel value in some way:
  - A method to invert the intensity of a pixel. To invert a pixel value, you subtract the pixel value from 255. For example, if the pixel value is 0 (i.e. black) then the inverted pixel value would be $255 - 0 = 255$ (i.e. white).
  - A method to compute the log intensity of a pixel value using the following equation:

    $$\text{pixel} = \text{coeff} * log(\text{pixel} + 1)$$

    Note that `coeff` is a double that should be passed into the method. You can use the `log()` method in the `java.lang.Math` class.
  - A method to compute the power law intensity of the pixel value using the following equation:

    $$\text{pixel} = \text{coeff} * \text{pixel}^{\text{gamma}}$$

    Note that both `coeff` and `gamma` should be doubles passed into the method. Again, you can use the `pow()` method in the `java.lang.Math` class to compute the pixel value raised to the power of gamma.

For each of these methods, you should make sure that the pixel value is always an integer value between $[0, 255]$. If the pixel value is ever less than 0, set the value to 0. If the pixel value is ever larger than 255, set the pixel value to 255.

Image Class

Open the `Image` class that has been provided. Notice that the instance variables and the constructor have already been written for you. The instance variables are the 2-dimensional array of Pixels (representing the image) along with the height of the image (i.e. the number of rows) and the width of the image (i.e. the number of columns). In addition, at the bottom you will find the following methods already implemented:

- `public void open(String filename)`: Opens the given filename and reads it into the 2D array.
- `public void save(String filename)`: Saves the current Image to the given filename.
- `public String toString()`: Returns the image in String format.

You should briefly read through these methods to familiarize yourself but do not modify them!

Add the following methods to the image class:

- A method `inverseTransform()` that inverts all the pixels in the image. This method should take no inputs.

- A method `logTransform()` that takes as input a double named `coeff` and converts every pixel to the appropriate log intensity value.
- A method `powerLawTransform()` that takes as input two doubles `coeff` and `gamma` and converts every pixel to the appropriate power law intensity value.

To test your methods, you can add code to the `main()` method in the `Controller` class. For example,

```
public static void main(String[] args){
    Image img = new Image("grumpycat.pgm");
    img.powerLawTransform(.75, 1.5);
    img.save("grumpycat_transformed.pgm");
}
```

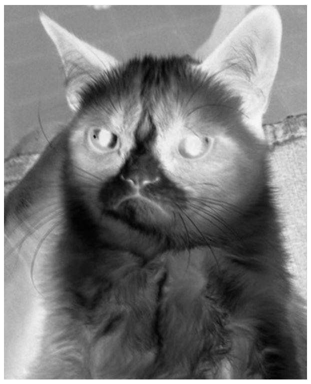
After running the `main()` method, double click on the `grumpycat_transformed.pgm` file and it should automatically open.

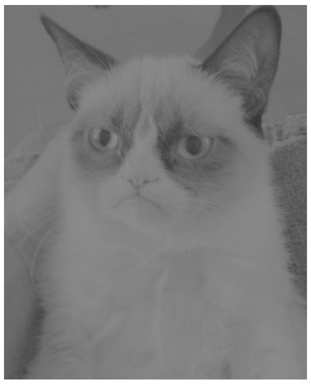
More Transformations

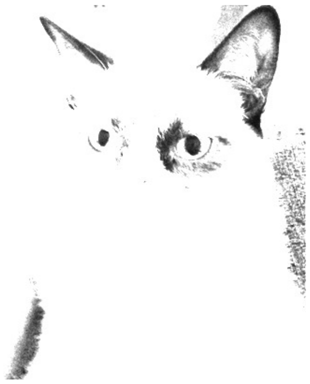
Now let's implement transformations that flip or rotate the image. Add the following methods to the `Image` class:

- A method `flipVertical()` that flips the image across the vertical (y) axis. In other words, the pixels on the left-hand side of the image get swapped to the right-hand side and vice versa.
- A method `flipHorizontal()` that flips the image across the horizontal (x) axis. In other words, the pixels on the top of the image get swapped with the pixels on the bottom and vice versa.
- A method `rotateClockwise()` that rotates the image clockwise. Note that after rotating the image, the dimensions of the image should be swapped. For example, if the original image has 10 rows and 3 columns then the rotated image will have 3 rows and 10 columns.
- A method `rotateCounterClockwise()` that rotates the image counter-clockwise. Again, after rotating, the dimensions of the image should be swapped.

Below is what my program outputs after I perform each transformation on the grumpy cat.


(a) inverse


(b) log transform (25)


(c) power law (.75, 1.5)

(d) vertical

(e) horizontal

(f) clockwise

(g) counter

Submitting Your Lab

Rename your folder with both of your names and then upload it to Moodle.