jDIP

A Java framework for digital image processing

M. Thaler, ©Zürcher Hochschule Winterthur, 1. September 2010

1 Introduction

jDIPis a simple framework written in Java to read, display and process digital images. In addition, it provides access to a frame grabber for reading images from a webCam based on JMF[1]. Within the framework, images are represented as two-dimensional pixel arrays.

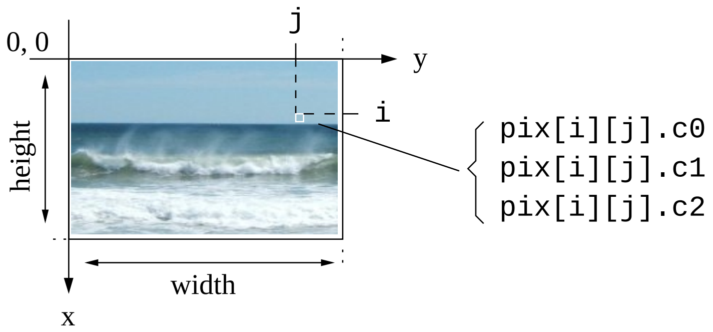
The main purpose of the framework is to enable users to implement their own image processing algorithms without having to care about reading and displaying images. In contrast to MATLAB, there is no library available with a comprehensive set of image processing functions. The framework only supports basic image processing functionality, mainly to convert between different image types: `RGB`, `GRAY` and `BINARY`.

2 Basics

With **jDIP**there is one important class: `Img`. `Img` handles everything concerning the image. Its main data structure is the two-dimensional pixel array `pix[][]`, with the parameters `width` and `hight` describing the geometry of the image. A third parameter, `colorMap`, represents the color type of the image. `colorMap` is used by the `display()`-function to correctly display the image.

2.1 Pixel arrays

Image data is represented as a two-dimensional pixel array named `pix[][]`, with `width` and `height` being the number of pixels in the x- respectively y-direction:



A pixel itself consists of three data values or so called channels named `c0`, `c1` and `c2`. The channels represent the components red, green and blue for color images, channel `c0` represents gray-levels for gray-scale images or black&white information for binary images. **jDIP**assumes data for each channel to be in the range $0 \leq x \leq 255$. Be aware, the data type of the channels is `short` which is a 16-Bit signed value[2]. **jDIP**assumes that the user takes care to keep pixel data within the

[1]Java Media Framework, Sun

[2]Java doesn't support unsigned data types

correct data range. The following table shows how the three image types and the corresponding pixel data are handled by **jDIP**:

channel	RGB	GRAY	BINARY
`c0`	red	gray (intensity)	binary (black & white)
`c1`	green	...	...
`c2`	blue	...	...
data range	$0 \leq x \leq 255$	$0 \leq x \leq 255$	$x = 0, x \neq 0$

To use logical operators with binary images, the data values of channel `c0` should be set to `0x0000` for black and to `0xFFFF` for white[3]. Both colors are defined as constants: `Img.bB` for black and `Img.bW` for white.

The display function selects the image type to be displayed according to the parameter `colorMap`. In the case of gray-scale and binary images, the `c1`- and `c2`-components are set accordingly (the image is actually displayed as a color image). For binary images **jDIP**interprets `c0 = 0` as black and `c0` $\neq$ `0` as white.

2.2 Images

When creating an `Img`-object, an image is either read from file or grabbed from a camera. In this case the size and color map of the image is automatically retrieved and set up accordingly. Alternatively the user can create an image and provide the size and color map by himself. The size of an image can be changed at any time (a new pixel array is allocated).

3 The class `Img`

The class `Img` handles everything concerning images. Access to pixel data is provided by the `public` array `pix[][]` (`pix[][].c0`, `pix[][].c1`, `pix[][].c2`) and the two `public` parameters `width` and `height`.

3.1 Constructors

Three constructors are available to create an `Img`-object:

`Img()`

Creates a new `Img`-object without allocating a pixel array.

`Img(int width, int height, int colorMap)`

Creates a new `Img`-object and allocates a pixel array of size `width` $\times$ `height`. Pixels are set to black. Sets color map to `colorMap`. Supported color maps are `Img.RGB`, `Img.GRAY` and `Img.BINARY`.

`Img(ImgGrabber fG)`

Creates a new `Img`-object and connects it to the frame grabber `fG`. A frame is read to determine the parameters `width` and `height`. See section ”The class ImgGrabber” for instantiating a frame grabber.

[3]With this definition, the negation of white is black and vice versa

3.2 Image I/O

Images can be read from and written to files. Reading overwrites the current image and sets the parameters `width`, `height` and `colorMap` according to the values obtained from the new image.

`void read()`

Displays an interactive dialog to read the input image file. The file type must be *jpeg*.

`void read(String fileName)`

Reads the input image file named `fileName`. The file type must be *jpeg*.

`void write(String fileName, String type)`

Writes image data to file with name `fileName`, the parameter `type` must be set to *jpeg*. Existing files are overwritten.

`void grabFrame()`

Grabs one frame from the camera. A frame grabber must be associated with the current image.

3.3 Image display

Window positions in Java are defined by the pair of parameters `{h,v}` relative to the upper left corner (position `{0,0}`): `h` corresponds to the horizontal displacement, whereas `v` corresponds to the vertical displacement.

`void display()`

Displays the image in a new window, unless the image is set to *single window display* (see below). In this case subsequent calls to `display()` show the image in the same window.

`void closeWindow()`

Closes the display window.

`void setWindowPosition(int h, int v)`

Set the position of the window to `h, v`.

`void setSingleWindowDisplay()`

Subsequent calls to `display()` will display the image in the same window.

`setSingleWindowDisplay(int h, int v)`

Subsequent calls to `display()` will display the image in the same window, the window is placed at position `h,v`.

3.4 Basic image processing functions

`void rgb2gray()`

Converts a color image to a gray-scale image, gray levels are stored in color channel `r`.

`gray2bin(int threshold)`

Converts a gray-scale image to a binary image based in the parameter `threshold` to determine white (> threshold) or black (≤ threshold. Black and white information is stored in color channel `c0` and is 0 for black and 1 for white.

`rgb2bin(int threshold)`
: Converts a color image directly to a black and white image based in the parameter `threshold`.

`void copy(Img fromImg)`
: Makes a copy of image `fromImage`, including the parameters `width`, `height`, `colorMap` and the frame grabber (if defined). A call to `display()` will show the copied image in a new window.

`void set(int width, int height, int colorMap)`
: Sets the parameters `width`, `height` and `colorMap` according to the given values and allocates an *empty* image (all color channels set to zero, independent of the color map).

`int getColorMap()`
: Returns the color map of the current image. The color maps are numerical constants and are defined as follows: `Img.RGB`: color image, `Img.GRAY`: gray-scale image, `Img.BINARY`: black and white image. In addition the color map parameter may be set to undefined `Img.UNDEF` (color map not yet defined) or `Img.ZEROED` (all color channels set to 0).

`setColorMap(int colorMap)`
: Sets the color map of the current image to `colorMap`. colorMap must be a valid color map.

3.5 Utilities

`static void sleep(int sleepTime)`
: `Img.sleep()` lets `Img` (current Thread) sleep for `sleepTime` milliseconds.

`static void waitAndExit()`
: `Img.waitAndExit()` waits for `CR` (new line) and then terminates **jDIP**.

`static void waitAndContinue()`
: `Img.waitAndContinue()` waits for `CR` (new line) and then contionues processing **jDIP**(e.g. for user controlled frame grabbing, etc.).

4 The class ImgGrabber

The class ImgGrabber provides a frame grabber to access frames from a web camera. Only one image grabber can be instantiated.

4.1 Constructors

Two constructors are available to create an `ImgGrabber`-object:

`ImgGrabber()`
: Creates an `ImgGrabber`-object and initializes the frame grabber.

`ImgGrabber(String preview)`
: Creates an `ImgGrabber`-object and initializes the frame grabber. If the parameter `preview` is set to the string ”`preview`”, a preview window is displayed int the upper left of the screen showing the camera input.

5 The class ImgTextDisplay

The class `ImgTextDisplay` displays a window where a line of text can be displayed.

5.1 Constructor

`ImgTextDisplay(int width, int height, int h, int v)`
A window at location {`h,v`} with size `width` and `height` is created.

5.2 Display text

`void displayText(String text)`
Displays the string passed in `text` within the text window.

6 Examples

The following examples demonstrate the usage of **jDIP**, they do not (necessarily) implement useful applicactions.

6.1 Example 1

Creates an `Img`-object, asks for the name of an image file and displays the image.

```
public static void main(String[] args) {
    Img img = new Img();
    img.setSingleWindowDisplay(100,100);
    while (true) {
        img.read();
        img.display();
    }
}
```

6.2 Example 2

The file ”wave.jpg” is read into image `imgA` and an empty image named `imgB` is created. The display windows are set up and positioned. Image `imgA` is then copied into `imgB` and tranformed into a gray-scale image. Every pixel in image `imgA` is set to {0, 255, 0} (green) if the corresponding pixel in image `imgB` has a intensity-value greater than 164. Finally both images are displayed.

```
public static void main(String[] args) {
    Img imgA = new Img("../i/wave.jpg");
    Img imgB = new Img();
    imgA.setSingleWindowDisplay(20,200);
    imgB.setSingleWindowDisplay(512,200);
    imgB.copy(imgA);
    imgB.rgb2gray();
    for (int i = 0; i < imgB.height; i++) {
        for (int j = 0; j < imgB.width; j++) {
            if (imgB.pix[i][j].c0 > 164) {
                imgA.pix[i][j].c0 = 0;
                imgA.pix[i][j].c1 = (short)255;
                imgA.pix[i][j].c2 = 0;
            }
        }
    }
    imgA.display();
    imgB.display();
}
```

6.3 Example 3

This example demonstrates how images are grabbed from a camera into an image, transformed into an gray-scale image and then displayed in a separate window. Frames are taken twice a second.

```
public static void main(String[] args) {
    ImgTextDisplay iW = new ImgTextDisplay(400, 60, 512, 20);
    iW.displayText("Testing the frame grabber");
    ImgGrabber iG = new ImgGrabber("preview");
    Img img = new Img(iG);
    img.setSingleWindowDisplay(20,200);
    while (true) {
        img.rgb2gray();
        img.display();
        img.grabFrame();
        img.sleep(500);
     }
}
```