Handbuch zur ImageToolBox^2

14. Januar 2018

Inhaltsverzeichnis

1 Systemvoraussetzungen

Zum Ausführen der ImageToolBox^2 wird das JRE 8 oder höher benötigt. Damit können kompilierte Filter im `.class` Format geöffnet werden. Um Filter zu schreiben und zu kompilieren wird das Development Kit JDK 8 oder höher benötigt. Ist das JDK im Pfad enthalten, können Filter auch direkt in der ImageToolBox^2 kompiliert werden.

2 Installation

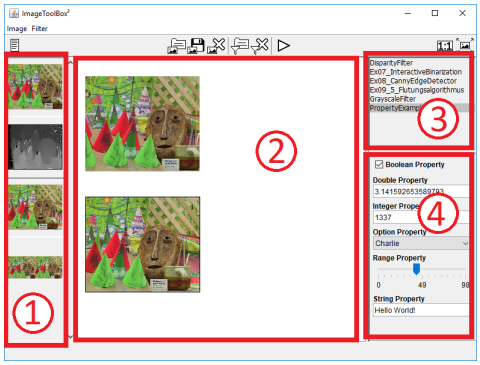
Zum Ausführen der ImageToolBox^2 diese entweder durch einen Doppelklick oder über die Konsole via `java -jar ITB2.jar` starten. Um Filter für die ImageToolBox^2 zu entwickeln sollte die `ITB2.jar` als externe Bibliothek hinzugefügt werden.
In Eclipse Rechtsklick auf das Projekt → Build Path → Add External Archives und dort die `.jar` Datei auswählen.
Um die ImageToolBox^2 innerhalb der IDE zu starten, reicht es für gewöhnlich aus auf Run zu klicken; alternativ kann eine `main` Methode erstellt werden, die die Funktion `Controller.startApplication()` ausführt:

```
import itb2.engine.Controller;

public class ItbLauncher {
  public static void main(String[] args) {
    Controller.startApplication();
  }
}
```

3 Bedienung der ImageToolBox^2

3.1 Hauptfenster



1. **Bilderübersicht**
   Hier werden die geöffneten sowie gefilterten Bilder dargestellt. Per Drag'n'Drop können Bilder verschoben, sowie neue Bilder geöffnet werden; dafür die Bilder aus dem Windows Explorer in

die Übersicht ziehen. Mit einem Doppelklick kann ein Fenster mit dem jeweiligen Bild geöffnet werden. Zur Auswahl das Bild anklicken, für eine Mehrfachauswahl die Bilder mit gedrückter Strg- oder Shift-Taste auswählen. Bei einer Mehrfachauswahl wird die Auswahl-Reihenfolge durch die Reihenfolge der Eingabe bestimmt.

2. **Arbeitsfläche**
   Hier werden die in der Bilderübersicht ausgewählten Bilder angezeigt. Bei einer Einzelauswahl kann mit dem Mausrad in das Bild hineingezoomt und mit der Maustaste das Bild verschoben werden.

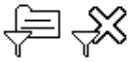
3. **Liste der Filter**
   Hier werden die geladenen Filter aufgelistet und ausgewählt. Zum Öffnen können neue Filter per Drag'n'Drop vom Windows Explorer aus in die Liste gezogen werden.

4. **Filtereigenschaften**
   Hier werden die Einstellungen des ausgewählten Filters angezeigt.

Menüleiste


Öffnet das Log-Fenster


Buttons zum Öffnen, Speichern und Schließen von Bildern

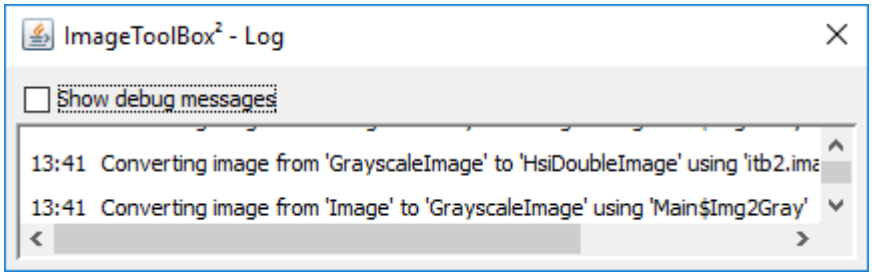

Buttons zum Öffnen und Schließen von Filtern


Startet den Filter


Zoomt das Bild auf der Arbeitsfläche zur tatsächlichen Größe


Passt das Bild auf der Arbeitsfläche dem Fenster an

3.2 Log



Hier werden alle ausgegebenen Nachrichten aufgelistet. Filter können Debug-Nachrichten ausgeben, die mit der Checkbox ein- und ausgeschaltet werden.

3.3 Pixel Auswahl

Der Filter hat die Möglichkeit ein Fester zur Auswahl bestimmter Pixel zu öffnen, zum Beispiel für die Auswahl von Objekt-Pixeln. Dabei kann der Filter eine bestimmte Anzahl oder beliebig viele Pixel anfragen. Sobald das Fenster geschlossen wird, wird der Filter weiter ausgeführt.

4 Filter und AbstractFilter

Jeder Filter muss das `Filter` Interface implementieren. Dieses Interface ermöglicht es ihm, mehrere Bilder als Eingabe zu erhalten, diese zu bearbeiten und daraufhin mehrere Bilder als Ausgabe wieder zurück zu geben, sowie eine `Collection` von `FilterProperty` Objekten zurück zu geben, welche alle Filtereigenschaften beinhaltet.

```
public Image[] filter(Image[] input)
public Collection<FilterProperty> getProperties()
```

Der `AbstractFilter` ist eine abstrakte Klasse, die die Grundfunktionen implementiert hat, sowie einige hilfreiche Zusatzfunktionen bietet:

```
public Image filter(Image input)
protected static Image callFilter(String name, Image image)
protected static Image[] callFilter(String name, Image[] image)
protected static int bound(int min, int val, int max)
protected static double bound(double min, double val, double max)
protected static void assertTrue(String message, boolean condition)
protected static void assertNotNull(String message, Object object)
```

In den meisten Fällen reicht die `filter(Image):Image` Methode aus; diese bekommt ein einzelnes Bild übergeben und kann ein Bild (oder `null`) zurückgeben. Sollte der Filter mehrere Eingabebilder benötigen oder mehrere Bilder zurückgeben, muss die `filter(Image[]):Image[]` Methode überschrieben werden. Mit den `callFilter` Methoden, kann ein Filter einen anderen Filter anhand seines Klassennamens aufrufen, sofern dieser geladen ist. Die `bound` Befehle, geben `val` zurück, sofern der Wert zwischen `min` und `max` liegt. Ansonsten wird der `min` bzw. `max` Wert zurück gegeben. Über die `assert` Methoden kann sichergestellt werden, dass der zweite Parameter wahr bzw. nicht `null` ist. Trifft dies nicht zu, wird der Filter mit der übergebenen Nachricht abgebrochen.

5 Ein- und Ausgabe von Bildern

Jedes Bild implementiert das Interface `Image` mit seinen Grundfunktionen. Es gibt sieben Grundarten von Bildern, darunter RGB-Bild, HSI-Bild, HSV-Bild, Grauwertbild und Binärbild. Dazu kommen noch das Gruppenbild, indem jedes Pixel einer bestimmten Gruppe zugeordnet wird und das Zeichnungsfähige-Bild, welches ein `Graphics` Objekt zurück geben kann, auf das Formen und Texte gezeichnet werden können. Beim Erstellen von Bildern kann zwischen zwei Implementationen gewählt werden: Einer mit hoher Genauigkeit (`double`) und einer mit geringer Genauigkeit (`byte`), welche im Vergleich zur Ersteren bedeutend weniger Speicherplatz benötigt. Alle Bilder haben ihren Ursprung oben links. Die Spalten zählen von links nach rechts, und die Zeilen zählen von oben nach unten, beide 0-basiert.

5.1 Bilder erstellen

Bilder können mit Hilfe der `ImageFactory` erstellt werden. Dafür muss zunächst die Genauigkeit festgelegt werden:

```
ImageFactory.bytePrecision()
ImageFactory.doublePrecision()
ImageFactory.getPrecision(Image image)
```

Mit der letzten Funktion wird die Genauigkeit des übergebenen Bildes gewählt.
Anschließend kann mit Angabe der Bildgröße das neue Bild erstellt werden:

```
.rgb(...)    // Erstellt ein RGB-Bild
.gray(...)   // Erstellt ein Grauwertbild
.binary(...)  // Erstellt ein Binärbild
.hsi(...)    // Erstellt ein HSI-Bild
.hsv(...)    // Erstellt ein HSV-Bild
.group(...) // Erstellt ein Gruppen-Bild
.drawable(...) // Erstellt ein Zeichnungsfähiges-Bild
```

Beispiel

```
public Image filter(Image input) {
  Image output = ImageFactory.getPrecision(input).gray(input.getSize());
  ...
  return output;
}
```

Da sowohl das Zeichnungsfähige-Bild (`DrawableImage`) als auch das Binärbild nur eine Genauigkeit kennen (`byte`), können diese beiden Bild-Sorten nur über die `.bytePrecision()` Factory erstellt werden.

5.2 Grundfunktionen von Bildern

Mit den folgenden Funktionen kann die Größe des Bildes und die Anzahl der Kanäle bestimmt werden:

```
public int getWidth()
public int getHeight()
public Dimension getSize()
public int getChannelcount()
```

Die erste Funktion `getValue` gibt ein Array mit den Farbwerten der Kanäle zurück, während die zweite Funktion den Wert für den angegebenen Kanal zurück gibt:

```
public double[] getValue(int column, int row)
public double getValue(int column, int row, int channel)
```

Über die folgenden Funktionen lässt sich der Wert eines Pixels setzen. Bei der ersten Funktion muss die Anzahl der Werte mit der Anzahl der Kanäle übereinstimmen. Zu beachten ist, dass das Bild den Wert nicht notwendiger Weise als `double` speichert; das ist abhängig von der gewählten Genauigkeit.

```
public void setValue(int column, int row, double... values)
public void setValue(int column, int row, int channel, double value)
```

Bilder können auch einen Namen speichern, bei geöffneten Bildern entspricht der Name dem Dateipfad. Der Name wird im Titel der GUI angezeigt.

```
public Object getName()
public void setName(Serializable name)
```

Außerdem gibt es eine Funktion, die das Bild als eine `BufferedImage` ausgibt. Diese Funktion wird zur Anzeige und zum Speichern des Bildes benötigt.

```
public BufferedImage asBufferedImage()
```

HSI- und HSV-Bilder haben vier zusätzliche Funktionen um den maximalen Wert der Kanäle festzulegen und abzufragen:

```
public void setMaxValue(int hue, int saturation, int intensity)
public int maxHue()
public int maxSaturation()
public int maxIntensity() // Bzw. maxValue()
```

Gruppenbilder haben drei zusätzliche Funktionen um die Gruppe eines Pixels festzulegen und auszulesen:

```
public int getGroupCount()
public void setGroup(int column, int row, int groupID)
public int getGroup(int column, int row)
```

Anstatt auf die Pixel direkt zuzugreifen, kann auch über das Bild iteriert werden:

```
Image image = ...;
for(Channel channel : image) {
  for(Row row : channel.rows())
    for(Cell cell : row)
      cell.setValue(42);
  for(Column column : channel.columns())
    for(Cell cell : column)
      cell.setValue(3.14);
}
```

5.3 Umwandlung von Bildern

Um die Handhabung der verschiedenen Bildtypen zu vereinfachen, gibt es den `ImageConverter`. Mit diesem können Bilder automatisch in einen gewünschten Typ konvertiert werden, sofern ein Filter existiert, der diese Konvertierung vornimmt. Soll ein Filter z.B. nur Grauwertbilder erhalten, kann der Filterklasse die `RequireImageType` Annotation hinzugefügt werden:

```
@RequireImageType(GrayscaleImage.class)
public class ExampleFilter extends AbstractFilter {
```

Ist es nicht möglich, die Eingabebilder in das gewünschte Format zu konvertieren, wird eine `ConversionException` geworfen. Filter können auch manuell ein Bild in ein anderes Format konvertieren:

```
HsiImage hsi = ImageConverter.convert(image, HsiImage.class);
```

Die ImageToolBox[2] bietet Standard-Konverter für die Bildsorten der `ImageFactory` an, es können aber auch eigene Filter als Konverter registriert werden. Zum Beispiel kann ein GrauwertFilter sich als Filter für `RgbImage` → `GrayscaleImage` registrieren:

```
public GrayscaleFilter() {
  ImageConverter.register(RgbImage.class,
          ImageFactory.bytePrecision().gray(), this);
}
```

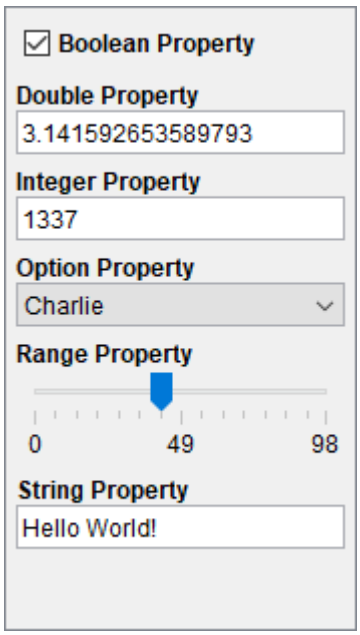
Dabei sollte das Eingabeformat so unspezifisch wie möglich und das Ausgabeformat so spezifisch wie möglich sein:

	Unspezifisch	Spezifisch
Eingabeformat	`RgbImage.class` ✓	`ImageFactory.bytePrecision().rgb()`
Ausgabeformat	`Grayscale.class`	`ImageFactory.bytePrecision().gray()` ✓

Wichtig
Wird keine direkte Konvertierung, aber eine über Umwege gefunden, wird diese ausgeführt. Das kann allerdings Verluste zur Folge haben. Zum Beispiel: RGB → Grauwert → HSI. Dadurch verliert das HSI-Bild die Farbinformationen. Die genutzten Konvertierungen werden im Log mit angegeben.

6 Kommunikation mit dem Nutzer

6.1 Filtereigenschaften



Es stehen sechs Sorten von Filtereigenschaften zur Verfügung: Boolean, Double, Integer, Option, Range und String.
Boolean, Double, Integer und String sind Eigenschaften für den jeweiligen Datentyp. Eine Range-Eigenschaft ist eine Ganzzahl (`int`) mit Grenzen und Schrittweite und wird in der GUI mit einem Slider dargestellt. Die Option-Eigenschaft lässt den Nutzer aus einem Array von Optionen wählen. Die Optionen können von irgendeiner Klasse sein; angezeigt wird der von der `toString()` Methode zurückgegebene Text. Jede Eigenschaft muss einen eindeutigen Namen besitzen, dieser kann auch nicht zwischen verschiedenen Eigenschaftstypen doppelt existieren. Der Name wird in der GUI über der jeweiligen Eigenschaft angezeigt.
Mit `properties.add...` können Eigenschaften einer bestimmten Sorte mit einem Standardwert hinzugefügt werden. Über `properties.get...` wird der Wert einer Eigenschaft abgefragt.

Beispiel

```
properties.addDoubleProperty("Winkel", Math.PI);
double angle = properties.getDoubleProperty("Winkel");
```

6.2 Fortschritt und Nachrichten

Über den `CommunicationManager` können Filter-Nachrichten an den Nutzer ausgegeben werden. Es gibt vier verschiedene Nachrichten-Sorten:

```
Controller.getCommunicationManager().info( ... )
Controller.getCommunicationManager().warning( ... )
Controller.getCommunicationManager().error( ... )
Controller.getCommunicationManager().debug( ... )
```

Die Parameter der vier Funktionen entsprechen dem Aufruf der `String.format(...)` Funktion. So muss mindestens ein String angegeben werden; es können aber noch weitere Parameter übergeben werden, die in die Nachricht mit eingefügt werden. Debug-Nachrichten werden nur angezeigt, wenn sie im Log-Fenster aktiviert wurden. Über den `CommunicationManager` können Filter ebenfalls eine Rückmeldung ihres Fortschritts geben. Werte unter 0 zeigen einen unbekannten Fortschritt an, während Werte zwischen 0 und 1 den Fortschritt in Prozent (0% - 100%) anzeigen.

```
Controller.getCommunicationManager().inProgress(double percent)
```

6.3 Pixel Auswahl

Benötigt ein Filter eine Liste von Pixeln, zum Beispiel für die Koordinaten eines Objekts oder die Farbe des Hintergrunds, kann der Filter über den `CommunicationManager` den Nutzer auffordern eine bestimmte oder unbestimmte Anzahl Pixel auszuwählen. (siehe 3.3 Pixel Auswahl)

```
List<Point> selections = Controller.getCommunicationManager()
        .getSelections(String message, int maxSelections, Image image);
```

Wenn `maxSelections` auf einen Wert größer 0 gesetzt wird, wird das Auswahlfenster automatisch geschlossen, sobald diese Anzahl von Pixeln ausgewählt wurde. Anschließend wird eine Liste mit den ausgewählten Pixel-Koordinaten zurück gegeben. Dabei entspricht die x-Koordinate der Spalte und die y-Koordinate der Zeile. Wird das Auswahlfenster vorzeitig geschlossen, werden die bis dahin ausgewählten Koordinaten zurück gegeben.

7 Beispiele für die Erstellung von Filtern

7.1 Farbfilter

Dieser Filter lässt nur den blauen Kanal durch und setzt die anderen Kanäle auf 0.

```
import itb2.filter.AbstractFilter;
import itb2.filter.RequireImageType;
import itb2.image.Image;
import itb2.image.ImageFactory;
import itb2.image.RgbImage;

// Eingabebild soll ein RGB-Bild sein
@RequireImageType(RgbImage.class)
public class ColorFilter extends AbstractFilter {

  @Override
  public Image filter(Image input) {
    // Ausgabebild hat Genauigkeit und Größe des Eingabebildes
    Image output = ImageFactory.getPrecision(input)
        .rgb(input.getSize());

    for(int col = 0; col < input.getWidth(); col++) {
      for(int row = 0; row < input.getHeight(); row++) {
        // Blauer Wert des Eingabebildes
        double blue = input.getValue(col, row, RgbImage.BLUE);
        // Rot und Grün auf 0, Blau auf den Wert des Eingabebildes
        output.setValue(col, row, 0, 0, blue);
      }
    }
```

```
    // Generiertes Bild ausgeben
    return output;
  }
}
```

Eine kürzere Version, die aus dem blauen Kanal ein Grauwertbild erstellt:

```
import itb2.filter.AbstractFilter;
import itb2.filter.RequireImageType;
import itb2.image.Channel;
import itb2.image.Image;
import itb2.image.ImageFactory;
import itb2.image.RgbImage;

// Eingabebild soll ein RGB-Bild sein
@RequireImageType(RgbImage.class)
public class ColorExtractor extends AbstractFilter {

  @Override
  public Image filter(Image input) {
    // Blauer Kanal des Eingabebildes
    Channel blue = input.getChannel(RgbImage.BLUE);
    // Grauwertbild aus dem Kanal erstellen
    return ImageFactory.getPrecision(input).gray(blue);
  }
}
```

7.2 Filtereigenschaften

Der folgende `PrintFilter` druckt auf das Eingabebild unten links eine Nachricht, welche über die GUI festgelegt werden kann.

```
import java.awt.Graphics;
import itb2.filter.AbstractFilter;
import itb2.image.DrawableImage;
import itb2.image.Image;

// Dieser Filter kann jedes Eingabebild akzeptieren
public class PrintFilter extends AbstractFilter {
  // Name der Eigenschaft
  private static final String MESSAGE = "Message";

  // Konstruktor
  public PrintFilter() {
```

```
    // Eigenschaft registrieren
    properties.addStringProperty(MESSAGE, "Hallo Welt!");
  }

  @Override
  public Image filter(Image input) {
    // Zeichnungsfähiges-Bild erstellen
    DrawableImage output = ImageFactory.bytePrecision()
        .drawable(input.asBufferedImage());
    Graphics graphics = output.getGraphics();

    // Eigenschaft abrufen
    String message = properties.getStringProperty(MESSAGE);
    // Text im Bild unten links einfügen
    graphics.drawString(message, 0, input.getHeight());

    // Bild ausgeben
    return output;
  }
}
```

7.3 Bilder konvertieren

Dieser Filter kann HSI-Bilder zu Grauwertbildern konvertieren und registriert sich dafür beim ImageConverter.

```
import itb2.filter.AbstractFilter;
import itb2.filter.RequireImageType;
import itb2.image.Channel;
import itb2.image.GrayscaleImage;
import itb2.image.HsiImage;
import itb2.image.Image;
import itb2.image.ImageConverter;
import itb2.image.ImageFactory;

// Eingabebild soll ein HSI-Bild sein
@RequireImageType(HsiImage.class)
public class Hsi2Gray extends AbstractFilter {

  // Konstruktor
  public Hsi2Gray() {
    // Den Filter zum Konvertieren von Bildern registrieren
    ImageConverter.register(HsiImage.class, GrayscaleImage.class, this);
  }
```

```
  @Override
  public Image filter(Image input) {
    // Der Intensitäts-Kanal von HSI-Bildern
    // entspricht dem Grauwertbild
    Channel intensity = input.getChannel(HsiImage.INTENSITY);
    // Grauwertbild erstellen und ausgeben
    return ImageFactory.getPrecision(input).gray(intensity);
  }
}
```