# Übungsaufgaben III, SBV1

Lukas Fiel, Lisa Panholzer January 6, 2019

### 3 Übungsaufgaben III

### 3.1 Resampling und Bildüberlagerung

#### a) Zerteilen eines Bildes

Zur vertikalen Teilung eines Bildes wurde ein simpler Filter ChopImgInHalfin ImageJ implementiert. Dieser definiert zuerst eine ROI (region of interest) welche die erste Hälfte des Bildes beinhaltet. Mittels ImageJUtility.chopImage kann dieser Bereich aus dem Ursprungsbild herausgeschnitten und angezeigt werden. Die Berechnung der zweiten Hälfte des Bildes unterscheidet sich lediglich durch die linke obere Koordinate des interessanten Bereichs (ROI).



Table 1: Zerteilung eines Bildes anhand selbst definiertem Filter

```
; columns

1 | import ij.*;
2 | import ij.plugin.filter.PlugInFilter;
3 | import ij.process.*;
4 | import java.awt.Rectangle;
5 | import java.awt.*;
6 | import ij.gui.GenericDialog;
```

```
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
      {\bf public\ class\ ChopImgInHalf\_\ implements\ PlugInFilter\ \{}
           public int setup(String arg, ImagePlus imp) {
    if (arg.equals("about"))
                                          {showAbout(); return DONE;}
                              return DOES_8G+DOES_STACKS+SUPPORTS_MASKING;
                  } //setup
                  public void run(ImageProcessor ip) {
                              byte[] pixels = (byte[]) ip.getPixels();
                              int width = ip.getWidth();
                              int height = ip.getHeight();
                  int [][] inDataArrInt = ImageJUtility.convertFrom1DByteArr(pixels, width, height)
\begin{array}{c} 24 \\ 25 \\ 26 \end{array}
                              double widthHalf = width / 2.0;
                              \begin{aligned} \textbf{double} \, [\,] \, [\,] \, & \, \text{tmpImage} \, = \, ImageJUtility.convertToDoubleArr2D (\,inDataArrInt \,, \\ & \hookrightarrow \, width \,, \, \, height ) \,; \end{aligned}
\frac{27}{28}
                              Rectangle roi = new Rectangle(0, 0, (int)widthHalf, height);
                              \begin{aligned} \textbf{double}\,[\,]\,[\,] & \text{ Img1} = \text{ImageJUtility.cropImage(tmpImage, roi.width, roi.} \\ &\hookrightarrow \text{height, roi)}; \end{aligned}
29
                               \begin{split} & ImageJUtility.showNewImage(Img1\,,~(int)\,widthHalf\,,~height\,,~"first\_half\_\\ &\hookrightarrow~image")\,; \end{split}
30
                              roi = new Rectangle ((int) width Half, 0, (int) width Half, height);
31
                              \mathbf{double}\,[\,][\,] \quad Img2 \ = \ ImageJUtility\,.\,cropImage\,(\,tmpImage\,,\ roi\,.\,width\,,\ roi\,.

    height, roi);
32

→ image");

\begin{array}{c} 33 \\ 34 \\ 35 \\ 36 \\ 37 \\ 38 \\ 39 \end{array}
                  } //run
                  void showAbout() {
                              IJ.showMessage("About_Template_...",
                                          "this_is_a_PluginFilter_template\n");
                  } //showAbout
40
         //class FilterTemplate_
```

## b) Transformation mittels Nearest Neighbor und Bilinearer Interpolation

```
16
17
18
19
20
21
22
23
24
                                  \} //setup
                                  public void run(ImageProcessor ip) {
                                                       byte[] pixels = (byte[])ip.getPixels();
                                                        int width = ip.getWidth();
                                                        int height = ip.getHeight();
                                  \textbf{int} \ [\ ] \ [\ ] \ \ in Data Arr Int \ = \ Image JUtility.convert From 1DByte Arr (\ pixels \ , \ width \ , \ height)
\frac{25}{26} \\ 27
                                  int widthHalf = (int) (width / 2.0);
                                                        \mathbf{double}\,[\,]\,[\,] \quad \mathrm{img1} \,=\, \mathrm{chopImgInHalf}(\,\mathrm{inDataArrInt}\,\,,\,\,\,\mathrm{width}\,\,,\,\,\,\mathrm{height}\,\,,\,\,\,\mathrm{widthHalf}\,\,,
                                                                     \hookrightarrow true);
28
                                                        \begin{aligned} \textbf{double} \ [] \ [] \ \ img2 \ = \ chopImgInHalf(inDataArrInt \,, \ width \,, \ height \,, \ widthHalf \,, \\ \hookrightarrow \ \textbf{false}) \,; \end{aligned}
29
30
                                                        \begin{array}{ll} int \ [] \ [] & int Img 2 \ = \ Image JU tility.convert To Int Arr 2D (img 2 \,, \ width Half \,, \\ & \hookrightarrow \ height) \,; \end{array}
\begin{array}{c} 31 \\ 32 \\ 33 \\ 34 \\ 35 \\ 36 \\ 37 \end{array}
                                   // define transform
                                  \label{eq:double_double} \textbf{double} \ \operatorname{trans} X \ = \ \operatorname{get} \operatorname{UserInput} \left( \, 0 \, \, , \text{``deltaX''} \, \right);
                                  double transY = getUserInput(0,"deltaY");
                                  double rotAngle = getUserInput(0,"rotation");
                                  //int\ [\ ][\ ]\ transformedImg\ =\ transformImage\ (inDataArrInt\ ,\ width\ ,\ height\ ,\ transX\ ,
                                                \hookrightarrow transY, rotAngle);
38
                                  \begin{array}{ll} \textbf{int} \; [\;] \; [\;] \; \; transformed Img \; = \; transform Image (int Img 2 \; , \; width Half \; , \; height \; , \; trans X \; , \\ \hookrightarrow \; trans Y \; , \; rot Angle ) \; ; \end{array}
39
40
                                  ImageJUtility.showNewImage(transformedImg, widthHalf, height, "transformed\_image") and the property of the p
\begin{array}{c} 41 \\ 42 \\ 43 \\ 44 \\ 45 \end{array}
                                  } //run
                                  void showAbout() {
                                                        {\tt IJ.showMessage("About\_Template\_..."}\;,
\begin{array}{c} 46 \\ 47 \\ 48 \\ 49 \\ 50 \\ 51 \\ 52 \\ 53 \end{array}
                                                                             "this_is_a_PluginFilter_template\n");
                                  } //showAbout
                                  public static int getUserInput(int defaultValue, String nameOfValue) {
                                                        // user input
                                                        System.out.print("Read_user_input:_" + nameOfValue);
                                                        GenericDialog gd = new GenericDialog(nameOfValue);
                                                        gd.addNumericField("please_input_" + nameOfValue + ":_", defaultValue,
54 \\ 55 \\ 56 \\ 57 \\ 58 \\ 59
                                                        gd.showDialog();
                                                        if (gd.wasCanceled()) {
                                                                             return 0:
                                                        int radius = (int) gd.getNextNumber();
                                                        System.out.println(radius);
60
                                                        return radius;
61
                                  }
6\overline{2}
63
                                  public double GetBilinearinterpolatedValue(int[][] inImg, double x, double y,
                                                → int width, int height) {
64
                                                        // calculate the delta for x and y
65
                                                        double deltaX = x - Math.floor(x);
\begin{array}{c} 66 \\ 67 \end{array}
                                                        68
69
                                                        // set calculation fragment
                                                        int xPlus1 = (int) x + 1;
                                                        int yPlus1 = (int) y + 1;
```

```
72
73
74
                                //handling\ translation\ and\ rotation\ for\ x\ and\ y
                                75
76
77
78
79
80
81
82
83
84
85
                                          return 0:
                                // get 4 neighboring pixels
                                \mathbf{int} \ \mathtt{neighbor1} = \mathtt{inImg} \, [\, \mathtt{xPlus1} \, ] \, [\, (\, \mathbf{int} \, ) \ (\, \mathtt{y} \, ) \, ] \, ;
                                int neighbor2 = inImg[(int) (x)][yPlus1];
                                int neighbor3 = inImg[xPlus1][yPlus1];
                                int neighbor4 = inImg[(int) (x)][(int) (y)];
                                // calculate weighted mean out of neighbors
                                double weightedMean = ((1 - \text{deltaX}) * (1 - \text{deltaY}) * \text{neighbor4}) + (
\hookrightarrow \text{deltaX} * (1 - \text{deltaY}) * \text{neighbor1})
                                                        \begin{array}{lll} + & ((1 - \text{deltaX}) * \text{deltaY} * \text{neighbor2}) + (\text{deltaX} * \text{deltaY} \\ & \hookrightarrow & * \text{neighbor3}); \end{array} 
 86
 87
 88
89
                                {\bf return}\ \ {\bf weighted Mean}\ ;
                    }
  90
 91
                    \begin{array}{ll} \textbf{public int} \; [] \; [] \; \; transformImage (int \; [] \; [] \; \; inImg \, , int \; \; width \, , \; int \; \; height \, , \; \; double \; \; transX \\ \hookrightarrow \; , \; \; double \; \; transY \, , \; \; double \; \; rotAngle) \; \; \{ \end{array}
 92
 9\overline{3}
                                // allocate result image
 94
95
                                int[][] resultImg = new int[width][height];
 96
97
                                // prepare cos theta, sin theta
                                double cosTheta = Math.cos(Math.toRadians(-rotAngle));
 98
                                double sinTheta = Math.sin(Math.toRadians(-rotAngle)); // - weil
                                      \hookrightarrow backgroundmapping
 99
100
                                double widthHalf = width / 2.0;
101
                                double heightHalf = height / 2.0;
102
103
104
                                //1) interate over all pixels and calc value utilizing backward-mapping
105
                                for ( int x = 0; x < width; x++) {
                                           for (int y = 0; y < height; y++) {
106
107
108
                                                       \label{eq:double_double} \textbf{double} \ \operatorname{tmppos} X \ = \ x \ - \ \operatorname{widthHalf} \, ;
109
                                                       double tmpposY = y - heightHalf;
110
111
                                                        //3) rotate
112
                                                       double posX = tmpposX * cosTheta + tmpposY * sinTheta;
113
                                                       double posY = - tmpposX * sinTheta + tmpposY * cosTheta;
114
115
                                                       //4) translate
116
117
                                                       posX \ -\!\!=\ transX \ ;
                                                       posY -= transY;
\begin{array}{c} 118 \\ 119 \end{array}
120
                                                       // move origin back from center to top corner
\frac{121}{122}
                                                       posX = posX + widthHalf;
                                                       posY = posY + heightHalf;
123
124
125
                                                        //6) get interpolated value if flag is true
                                                       if (nnFlag) {
126
127
128
129
                                                                   int nnX = (int) (posX + 0.5);
                                                                   int nnY = (int) (posY + 0.5);
                                                                   //6) \ assigne \ value \ from \ original \ img \ inImg \ if \\ \hookrightarrow \ inside \ the \ image \ boundaries
```

```
130
                                                                if(nnX >= 0 \&\& nnX < width \&\& nnY >= 0 \&\& nnY <
                                                                       ⇔ height) {
131 \\ 132 \\ 133
                                                                           resultImg[x][y] = inImg[nnX][nnY];
\frac{134}{135}
                                                     else {
                                                                // if nearest neighbor flag is false, do
                                                                       \hookrightarrow bilinear interpolation
136
                                                                \begin{array}{ll} \textbf{double} \ \ resultVal = \ \ GetBilinear interpolatedValue (\\ \hookrightarrow \ \ inImg \,, \ posX \,, \ posY \,, \ width \,, \ height); \end{array}
137
\frac{138}{139}
                                                                //set new rounded value for current location
                                                                resultImg[x][y] = (int) (resultVal + 0.5);
140
141
142
143
144
                                         }
\overline{145}
146
                              return resultImg;
147
                   }
148
149
150
                    \begin{array}{ll} \textbf{public static double} \, [\,] \, [\,] & \texttt{chopImgInHalf} \, (\textbf{int} \, [\,] \, [\,] & \texttt{inDataArrInt} \, , \, \, \textbf{int} \, \, \textbf{width} \, , \, \, \textbf{int} \\ & \hookrightarrow \, \, \textbf{height} \, , \, \, \textbf{int} \, \, \textbf{widthHalf} \, , \, \, \textbf{boolean} \, \, \, \textbf{flag} \, ) \, \, \, \{ \end{array} 
151
                              //\ store\ half\ of\ width\ in\ int\ var
\begin{array}{c} 152 \\ 153 \end{array}
                              // create temporary image
154
                              155
156
157
                              if (flag == true) {
                                          // create region of interest
158
                                         Rectangle roi = new Rectangle(0, 0, widthHalf, height);
159
160
                                          // crop image and store first half in var
                                         double[][] Img1 = ImageJUtility.cropImage(tmpImage, roi.width, 

→ roi.height, roi);
161
162
                                         ImageJUtility.showNewImage(Img1, widthHalf, height, "first_half_

→ image");
163
164
                                         return Img1;
165
                              } else {
166
167
                                          // create region of interest
168
                                         Rectangle roi = new Rectangle(0, 0, widthHalf, height);
169
170
                                          // overwrite roi with values for second half, crop image and
                                               \hookrightarrow store second half
171 \\ 172 \\ 173
                                          // in var
                                          roi = new Rectangle(widthHalf, 0, widthHalf, height);
                                         double[][] Img2 = ImageJUtility.cropImage(tmpImage, roi.width,

    roi.height, roi);
174
                                          ImageJUtility.showNewImage(Img2\,,\ widthHalf\,,\ height\,,\ "second\_half
                                                175
                                         return Img2;
\frac{176}{177}
                              }
                   }
178
179
180
       } //class FilterTemplate_
```

#### c) automatische Registrierung

Es wurde ein Filter in ImgaeJ implementiert, der zur automatischen Registrierung von Bildinhalten herangezogen werden soll. Dabei wurde von den gegebenen Testbildern ausgegangen.

Da diese mit einer Bildtiefe von 8bit nur Werte von 0 (schwarz) bis 255 (weiß) aufweisen, kann mittels SSE einfach ein Algorithmus geschrieben werden, der die Bilder voneinander subtrahiert und die Pixelwerte des Resultatbildes als Fitness heranzieht und aufsummiert. Der Hintergrund der gegebenen Bilder ist dabei meist weiß (255). Bei einer Verschiebung und anschließender Subtraktion entstehen aus diesem Grund aber schwarze Fragmente am Rand. Dieser Umstand kann leicht eliminiert werden, indem das Ursprungsbild zu Beginn invertiert wird. So ist der Hintergrund schwarz (0). Kanten werden dementsprechend weiß (255) dargestellt.

Das invertierte Bild wird anschließend, wie in Punkt a ) beschrieben, zerteilt und die Einzelbilder dargestellt.

Die eigentliche Registrierung verschiebt nun Bild1 in x und y Richtiung und rotiert dieses auch um jeweils ein Inkrement. Jedes dieser transformierten Bilder wird nun von Bild2 abgezogen und erneut ein Fitneswert berechnet. Es ist davon auszugehen, dass ein schwarzer Hintergrund (0) abgezogen von einem schwarzen Hintergrund (0) wiederum 0 ergibt. Werden allerdings weiße Pixel von schwarzem Hintergrund abgezogen, oder schwarzer Hintergrund von weißen Linien abgezogen, so erhält man Werte abweichend von 0. Auch Negativwerte sind so denkbar, wesshalb diese Differenzwerte zum Quadrat genommen werden. Hierdurch sind Differenzwerte immer positiv.

Wird Bild1 irgendwann genau auf die Position geschoben an der sich Bild2 befindet so subtrahieren sich die weißen Linien im Idealfall zu 0. So kann ein eindeutiger Fitnesswert errechnet werden, der sein Optimum bei 0 findet.

Aus Ressourcengründen werden all die beschriebenen Berechnungen/Verschiebungen mit dem NearesNeighbor Algorithmus berechnet. Ist das Optimum gefunden wird anschließend nocheinmal die Transformation mit Bilinearer Interpolation berechnet und von Bild2 subtrahiert. Das Resultatbild wird zum Schluss für den User sichtbar dargestellt um den Erfolg des Filters zu veranschlaulichen.

gegebenes Testbild		invertierter Bildausschnitt1	invertierter Bildausschnitt2	resultierendes Differenzbild
•				
•				

Table 2: Testfälle: automatische Registrierung

```
; columns
\begin{array}{c} 1\\2\\3\\4\\5\\6\\7\\8\\9\\10\\11\\12\\13\\14\\15\\16\\17\\18\\19\\20\\21\\22\end{array}
     | import ij.*;
     import ij.plugin.filter.PlugInFilter;
     import ij.process.*;
     import java.awt.*;
      public class AutoRegisterFinal_ implements PlugInFilter {
                public int setup(String arg, ImagePlus imp) {
    if (arg.equals("about")) {
                                      showAbout();
                                      return DONE;
                           return DOES_8G + DOES_STACKS + SUPPORTS_MASKING;
                public void run(ImageProcessor ip) {
                           // read image
                           \mathbf{byte}\,[\,] \quad \mathtt{pixels} \;=\; (\,\mathbf{byte}\,[\,]\,) \quad \mathtt{ip}\,.\,\mathtt{getPixels}\,(\,)\;;
                           int width = ip.getWidth();
                           int height = ip.getHeight();
                           \begin{array}{c} 23 \\ 24 \\ 25 \\ 26 \end{array}
                           // invert to set background to black
                           \mathbf{int} \; [ \; ] \; \; \mathbf{invertTF} \; = \; \mathbf{ImageTransformationFilter} \, . \, \mathbf{GetInversionTF} \left( 255 \right) \, ;
```

```
\frac{27}{28} \\ 29
                                                     int widthHalf = (int) (width / 2.0);
                                                     double[][] img1 = chopImgInHalf(inDataArrInt, width, height, widthHalf,
                                                                30
                                                     \mathbf{double}\,[\,]\,[\,] \quad \mathrm{img2} \,=\, \mathrm{chopImgInHalf}\big(\,\mathrm{inDataArrInt}\;,\;\; \mathrm{width}\;,\;\; \mathrm{height}\;,\;\; \mathrm{widthHalf}\;,
                                                                 \hookrightarrow false);
\begin{array}{c} 31 \\ 32 \\ 33 \\ 34 \\ 35 \\ 36 \\ 37 \\ 38 \end{array}
                                                     // initialize ranges
                                                     int xRadius = 20;
                                                     int yRadius = 20;
                                                     int rotRadius = 20:
                                                      // initialize arrays
                                                      int[][] intImg1 = ImageJUtility.convertToIntArr2D(img1, widthHalf,
                                                                    height);
39
                                                      int[][] intImg2 = ImageJUtility.convertToIntArr2D(img2, widthHalf,
                                                                     height);
40
                                                      int[][] transformedImg;
                                                     int[][] diffImg;
41
                                                     int [][] [] ssE = new int [2 * xRadius + 1][2 * yRadius + 1][2 * rotRadius \leftrightarrow + 1];
42
43
44
                                                      // initial fitness
45
                                                      \label{eq:diffImg} diffImg \ = \ ImageJUtility.calculateImgDifference (intImg1 \,, \ intImg2 \,,

    widthHalf, height);

46
                                                     int initialFitness = calculateSSE(diffImg, widthHalf, height);
47
                                                     System.out.println ("initiale\_Fitness:\_" + initialFitness);\\
48
49
                                                     // fill ssE matrix and find minimum
50
51
52
53
                                                     int minimum = initialFitness;
                                                     int tmpSSE = 0:
                                                     int \min Xind = 0;
                                                     int \min Yind = 0;
 54
                                                     int \min AngleInd = 0;
55
                                                     \label{eq:formula} \textbf{for (int } \textbf{x} = -\textbf{x} \\ \textbf{Radius; } \textbf{x} < \textbf{x} \\ \textbf{Radius; } \textbf{x} + +) \text{ } \{
56
                                                                         \label{eq:formula} \textbf{for (int } y = -y Radius; \ y < \ y Radius; \ y++) \ \{
57
                                                                                              for (int angle = -rotRadius; angle < rotRadius; angle++)
58
                                                                                                                   \begin{array}{l} \text{transformedImg} = \text{transformImage(intImg1,} \\ \hookrightarrow \text{widthHalf, height, x, y, angle, false);} \end{array} 
59
                                                                                                                    diffImg = ImageJUtility.calculateImgDifference(
                                                                                                                              \hookrightarrow \ transformedImg \ , \ intImg2 \ , \ widthHalf \ ,

    height);
60
                                                                                                                   tmpSSE = calculateSSE(diffImg, widthHalf, height
                                                                                                                              \hookrightarrow );
61
                                                                                                                   ssE[x + xRadius][y + yRadius][angle + rotRadius]
                                                                                                                               \hookrightarrow = tmpSSE;
62
63
                                                                                                                   // find minimum and save indices for later
64
                                                                                                                   if (tmpSSE < minimum) {
65
                                                                                                                                      \label{eq:minimum} \operatorname{minimum} \ = \ \operatorname{tmpSSE}\,;
66
                                                                                                                                      //System.out.println("current minimal 

→ fitness: " + minimum);
67
                                                                                                                                       \min X \operatorname{ind} = x;
68
                                                                                                                                       minYind = y;
69
70
71
72
73
74
75
                                                                                                                                       minAngleInd = angle;
                                                                                                                  }
                                                     System.out.println("final_Fitness:_" + minimum);
                                                     System.out.println("minXind:"+minXind+"minYind:"+minYind+"minAngleInd:"+minYind+"minAngleInd:"+minYind+"minAngleInd:"+minYind+"minAngleInd:"+minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minYind+"minY

→ minAngleInd);
76
77
78
                                                     minXind = 22;
                                                     minYind = -6;
```

```
79
                                                         minAngleInd = -4:
   80
   \frac{81}{82}
                                                         // plot difference image to proof the transformation
                                                         transformedImg = transformImage(intImg1, widthHalf, height, minXind,\\
                                                                     83
                                                         diffImg \ = \ ImageJUtility.calculateImgDifference (transformedImg, intImg2, intImg2)

→ widthHalf, height);
   84
                                                         \label{eq:continuous_loss} \begin{split} & Image J U tility .show New Image ( diff Img , width Half , height , "fittest\_diff \_ & \mapsto image" ) \, ; \end{split}
   85
   86
                                   } // run
   87
88
                                    void showAbout() {
                                                         IJ.show Message ("About\_Template\_...", "this\_is\_a\_PluginFilter\_template \backslash n") is a plugin for the plate of the plate of the plugin for the plate of the plugin for the pl
   89
                                                                    \hookrightarrow );
   90
                                   } // showAbout
   91
   92
                                   \begin{array}{lll} \textbf{public int} \; [\,] \; [\,] \; \; transformImage (\, \textbf{int} \; [\,] \; [\,] \; \; inImg \,, \; \; \textbf{int} \; \; width \,, \; \; \textbf{int} \; \; height \,, \; \; \textbf{double} \\ \hookrightarrow \; transX \,, \; \; \textbf{double} \; \; transY \,, \; \; \textbf{double} \; \; rotAngle \,, \end{array}
   93
                                                                             boolean interpolation) {
   94
   95
                                                         // allocate result image
   96
                                                         int[][] resultImg = new int[width][height];
   97
   98
                                                         // prepare cos theta, sin theta
   99
                                                         double cosTheta = Math.cos(Math.toRadians(-rotAngle));
100
                                                         double sinTheta = Math.sin(Math.toRadians(-rotAngle)); // - weil
                                                                     \hookrightarrow backgroundmapping
101
102
                                                         double widthHalf = width / 2.0;
103
                                                         double heightHalf = height / 2.0;
104
105
                                                         // 1) interate over all pixels and calc value utilizing backward-mapping
106
                                                         for (int x = 0; x < width; x++) {
107
                                                                              \label{eq:formula} \textbf{for (int } y = 0; \ y < \ \text{height}; \ y\text{++}) \ \{
\overline{108}
109
                                                                                                   double tmpposX = x - widthHalf;
                                                                                                   110
111
\overline{112}
                                                                                                   // 3) rotate
\begin{array}{c} 113\\114 \end{array}
                                                                                                   \label{eq:double_posX} \textbf{double} \hspace{0.1cm} posX \hspace{0.1cm} = \hspace{0.1cm} tmpposX \hspace{0.1cm} * \hspace{0.1cm} cosTheta \hspace{0.1cm} + \hspace{0.1cm} tmpposY \hspace{0.1cm} * \hspace{0.1cm} sinTheta;
                                                                                                   double posY = -tmpposX * sinTheta + tmpposY * cosTheta;
115
116
117
                                                                                                   // 4) translate
                                                                                                   posX -= transX;
118
                                                                                                   posY -= transY;
119
120
                                                                                                   // move origin back from center to top corner
121
122
123
124
                                                                                                   posX = posX + widthHalf;
                                                                                                   posY = posY + heightHalf;
                                                                                                   // 6) assigne value from original imag inImg if inside

    → the image boundaries

\frac{125}{126}
                                                                                                   // get interpolated value if flag is true
                                                                                                   if (interpolation == false) {
127
128
129
130
                                                                                                                       \mathbf{int} \hspace{0.2cm} \mathrm{nnX} \hspace{0.1cm} = \hspace{0.1cm} (\hspace{0.1cm} \mathbf{int}\hspace{0.1cm}) \hspace{0.2cm} (\hspace{0.1cm} \mathrm{posX} \hspace{0.1cm} + \hspace{0.1cm} 0.5\hspace{0.1cm}) \hspace{0.1cm} ;
                                                                                                                        int nnY = (int) (posY + 0.5);
                                                                                                                        // \ 6) \ assign \ value \ from \ original \ img \ inImg \ if \\ \hookrightarrow \ inside \ the \ image \ boundaries 
131
                                                                                                                        if (nnX >= 0 && nnX < width && nnY >= 0 && nnY <

    height) {
132
                                                                                                                                            resultImg[x][y] = inImg[nnX][nnY];
133
134
                                                                                                   } else {
```

```
135
                                                       // if not nearest neighbor, do bilinear
                                                             \hookrightarrow interpolation
136
                                                       double resultVal = GetBilinearinterpolatedValue(
                                                             → inImg, posX, posY, width, height);
137
138
139
                                                       //\ set\ new\ rounded\ value\ for\ current\ location
                                                       {\tt resultImg}\,[\,x\,]\,[\,y\,] \;=\; (\,\mathbf{int}\,) \ (\,{\tt resultVal}\,+\,0.5\,)\;;
140
141
                                   }
142
                          }
143
                          return resultImg:
\overline{144}
                }
145
146
                public int calculateSSE(int[][] diffImg, int width, int height) {
147
                         int sse = 0;
148
149
                          for (int x = 0; x < width; x++) {
150
                                   for (int y = 0; y < height; y++) {
151
                                            sse = sse + diffImg[x][y];
152
153 \\ 154
                          }
155
                         return sse;
156
                }
\bar{157}
                158
159
160
161
                          // create temporary image
162
                          double[][] tmpImage = ImageJUtility.convertToDoubleArr2D(inDataArrInt,

    width, height);
163
164
                          if (flag == true) {
165
166
                                    // create region of interest
                                    Rectangle roi = new Rectangle(0, 0, widthHalf, height);
167
168
                                    // crop image and store first half in var
169
                                   double[][] Img1 = ImageJUtility.cropImage(tmpImage, roi.width,

    roi.height, roi);

170
                                   \begin{split} & ImageJUtility.showNewImage(Img1\,,\ widthHalf\,,\ height\,,\ "first\_half\_ \\ &\hookrightarrow image")\,; \end{split}
171
172
173
                                   return Img1;
                          } else {
174
175
176
177
178
                                    // create region of interest
                                   Rectangle roi = new Rectangle(0, 0, widthHalf, height);
                                    // overwrite roi with values for second half, crop image and
                                         \hookrightarrow \ store \ second \ half
179
                                    // in var
180
                                   roi = new Rectangle(widthHalf, 0, widthHalf, height);
                                   \begin{aligned} \textbf{double}\,[\,]\,[\,] & \text{Img2} = \text{ImageJUtility.cropImage(tmpImage, roi.width,} \\ &\hookrightarrow \text{roi.height, roi)}; \end{aligned}
181
182
                                   ImageJUtility.showNewImage(Img2, widthHalf, height, "second_half
                                         183
                                   return Img2;
184
                         }
185
                }
186
187
                \textbf{public double} \ \ \mathrm{GetBilinear interpolatedValue}(\textbf{int}\ [\ ]\ [\ ]\ \ inImg\ ,\ \ \textbf{double}\ \ x\ ,\ \ \textbf{double}\ \ y,
                      \hookrightarrow int width, int height) {
188
                          // calculate the delta for x and y
189
                          double deltaX = x - Math.floor(x);
```

```
190
                            double deltaY = y - Math.floor(y);
191
192
193
                            // set calculation fragment
                            int xPlus1 = (int) x + 1;
194
195
                            int yPlus1 = (int) y + 1;
196
197
                            //handling translation and rotation for x and y
                            if (x < 0 || x >= width || y < 0 || y >= height || xPlus1 < 0 || xPlus1 

\leftrightarrow >= width || yPlus1 < 0 || yPlus1 >= height) {
198
199
200
201
                                     return 0;
201
202
203
204
205
206
                            // get 4 neighboring pixels
                            int neighbor1 = inImg[xPlus1][(int) (y)];
                            int neighbor2 = inImg[(int) (x)][yPlus1];
                            int neighbor3 = inImg[xPlus1][yPlus1];
                            \mathbf{int} \ \mathtt{neighbor4} = \mathtt{inImg} \, [\, (\, \mathbf{int}\,) \ (x) \, ] \, [\, (\, \mathbf{int}\,) \ (y) \, ] \, ;
207
208
209
                            // calculate weighted mean out of neighbors
                            210
211 | return weig
212 | return weig
213 | }
214 | 215 |} // class FilterTemplate.
                            return weightedMean;
```

Figure 1: Resampling anhand bilinearer Interpolation und Skalierung um Faktor  $2.0\,$