

Übungsaufgaben IV, SBV1

Lukas Fiel, Lisa Panholzer

January 28, 2019

4 Übungsaufgaben IV

4.1 Region Growing

a) Manuelles Image Growing

Der Algorithmus zu dieser Übung wurde aus der Vorlesung übernommen. Es waren lediglich N4 und N8 Nachbarpixelregionen zu unterscheiden. Diese wurden einfach durch Variable der Funktion mitgegeben und in einer *if* Abfrage abgefragt.

Figure 1 und Figure 2 vergleichen die zu untersuchenden Nachbarschaftspixel.

Regionsvergleich

x/y	-1	0	1
-1	0	x	0
0	x	0	x
1	0	x	0

Table 1: N4 Region

x/y	-1	0	1
-1	x	x	x
0	x	0	x
1	x	x	x

Table 2: N8 Region

Listing 1: RegionGrowing-Algorithmus.

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
; columns

import java.awt.Point;
import java.awt.Rectangle;
import java.util.Stack;

import ij.*;
import ij.gui.GenericDialog;
import ij.gui.PointRoi;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;

public class RegionGrowing_ implements PlugInFilter {

    ImagePlus impl;

    public int setup(String arg, ImagePlus imp) {
        if (arg.equals("about")) {
            showAbout();
            return DONE;
        }

        impl = imp;
        return DOES_SG + DOES_STACKS + SUPPORTS_MASKING + ROI_REQUIRED;
    } // setup

    public static int[][] performRegionGrowing(int[][] inImgArr, int width, int height, int lowerThresh, int upperThresh
        ↪ , int seedX, int seedY, String region) {
        // constants
        int BG_VAL = 0;
        int FG_VAL = 255;
        int UNPROCESSED_VAL = -1;

        int[][] returnArr = new int[width][height];

        for ( int x= 0; x < width; x++) {
            for (int y = 0; y < height; y++) {
                returnArr[x][y] = UNPROCESSED_VAL;
            }
        }

        Stack<Point> processingStack = new Stack<Point>();

        //first check if seed point is valid

```

```

43 | int seedVal = inImgArr[seedX][seedY];
44 | if (seedVal >= lowerThresh && seedVal <= upperThresh) {
45 |     processingStack.push(new Point(seedX, seedY));
46 |     returnArr[seedX][seedY] = FG_VAL;
47 | }
48 |
49 | while (!processingStack.empty()) {
50 |     Point nextPos = processingStack.pop();
51 |
52 |     //check all children in N4
53 |     for (int xOffset = -1; xOffset <= 1; xOffset++) {
54 |         for (int yOffset = -1; yOffset <= 1; yOffset++) {
55 |             int nbX = nextPos.x + xOffset;
56 |             int nbY = nextPos.y + yOffset;
57 |
58 |             // check if N4 region
59 |             boolean isRegion = false;
60 |             if (region.equals("N4") && (xOffset*yOffset == 0 && xOffset+yOffset != 0)) isRegion =
                ↳ true;
61 |             if (region.equals("N8") && (xOffset != 0 || yOffset != 0)) isRegion = true;
62 |
63 |
64 |             if (isRegion) {
65 |
66 |                 // check if valid range ==> position within image boundaries
67 |                 if (nbX >= 0 && nbY >= 0 && nbX < width && nbY < height) {
68 |
69 |                     int nbVal = inImgArr[nbX][nbY];
70 |
71 |                     //if current pixel was not processed yet (check if pixel is
                ↳ unprocessed and if value in threshold range)
72 |                     if (returnArr[nbX][nbY] == UNPROCESSED_VAL) {
73 |
74 |                         //if range valid
75 |                         if (nbVal >= lowerThresh && nbVal <= upperThresh) {
76 |                             returnArr[nbX][nbY] = FG_VAL;
77 |                             ↳ //set current pixel to foreground
78 |                             processingStack.push(new Point(nbX, nbY));
79 |                             ↳ //push current pixel to the stack
79 |                         }
80 |                         else {
81 |                             returnArr[nbX][nbY] = BG_VAL;
82 |                         }
83 |                     }

```

```

84         } // if N4 region
85     } // for yOffset
86 } // for xOffset
87
88 } // while
89
90 System.out.println("processingStack.size()");
91
92 //cleanup - all values still unprocessed - get assigned the background value BG_VAL
93 for ( int x = 0; x < width; x++) {
94     for (int y = 0; y < height; y++) {
95         if (returnArr[x][y] == UNPROCESSED_VAL) {
96             returnArr[x][y] = BG_VAL;
97         }
98     }
99 }
100
101
102     return returnArr;
103 } //performRegionGrowing
104
105
106 public void run(ImageProcessor ip) {
107     byte[] pixels = (byte[]) ip.getPixels();
108     int width = ip.getWidth();
109     int height = ip.getHeight();
110
111     int [][] inDataArrInt = ImageJUtility.convertFromIDByteArr(pixels, width, height);
112
113     //request seed point
114     PointRoi pr = (PointRoi)impl.getRoi();
115     Rectangle rect = pr.getBounds();
116     int xStart = pr.getXCoordinates()[0] + rect.x;
117     int yStart = pr.getYCoordinates()[0] + rect.y;
118
119     System.out.println("xStart:_" + xStart + " ,_yStart:_" + yStart);
120
121     // user input - default
122     int lowerThresh = 100;
123     int upperThresh = 255;
124     // user dialog
125     GenericDialog gd = new GenericDialog("thresh_params");
126     gd.addSlider("lower_thresh", 0, 255, lowerThresh);
127     gd.addSlider("upper_thresh", 0, 255, upperThresh);
128     gd.showDialog();

```

```

129 |
130 |         if (!gd.wasCanceled()) {
131 |             lowerThresh = (int) gd.getNextNumber();
132 |             upperThresh = (int) gd.getNextNumber();
133 |         }
134 |
135 |         //finally calling function
136 |         int [][] resultImg = performRegionGrowing(inDataArrInt, width, height, lowerThresh, upperThresh, xStart,
137 |             ↪ yStart,"N4");
138 |
139 |         ImageJUtility.showNewImage(resultImg, width, height, "region_coin_result");
140 |
141 |     } // run
142 |
143 |     void showAbout() {
144 |         IJ.showMessage("About_Template...", "this_is_a_PluginFilter_template\n");
145 |     } // showAbout
146 |
147 | } // class FilterTemplate_

```

b) Image Growing mit Labeling

Für die Implementierung wurde der Code aus Aufgabe a) kopiert und erweitert. Muss das gesamte Bild untersucht werden um alle Objekte zu finden. Wird ein passendes Pixel gefunden, wird der Region-Growing Algorithmus herangezogen. Mittels der Fordergrundfarbe werden die Objekte eingeteilt und unterschieden.

Listing 2: RegionGrowing-Algorithmus

```

1      , columns
2
3      import java.awt.Point;
4      import java.awt.Rectangle;
5      import java.util.Stack;
6
7      import ij.*;
8      import ij.gui.GenericDialog;
9      import ij.gui.PointRoi;
10     import ij.plugin.filter.PlugInFilter;
11     import ij.process.*;
12
13     public class AutoRegionGrowing_ implements PlugInFilter {
14
15         public int setup(String arg, ImagePlus imp) {
16             if (arg.equals("about"))
17                 {showAbout(); return DONE;}
18             return DOES_8G+DOES_STACKS+SUPPORTS_MASKING;
19         } //setup
20
21     public static int[][] performRegionGrowing(int[][] inImgArr, int width, int height, int lowerThresh, int upperThresh)
22         ↪ , String region) {
23         // constants
24         int BG_VAL = 0;
25         int FG_VAL = 255;
26         int UNPROCESSED_VAL = -1;
27
28         int[][] returnArr = new int[width][height];
29
30         // prepare → set every pixel to unprocessed state
31         for ( int x= 0; x < width; x++) {
32             for (int y = 0; y < height; y++) {
33                 returnArr[x][y] = UNPROCESSED_VAL;
34             }
35         }
36
37         Stack<Point> processingStack = new Stack<Point>();
38
39         // for whole image
40         for ( int x = 0; x < width; x++) {
41             for ( int y = 0; y < height; y++) {

```



```

43 | //first check if seed point is valid
44 | int seedVal = inImgArr[x][y];
45 | if (seedVal >= lowerThresh && seedVal <= upperThresh && returnArr[x][y] == UNPROCESSED_VAL) {
46 |     processingStack.push(new Point(x, y));
47 |     FG_VAL = FG_VAL - 20;
48 |     System.out.println("next-foreground_will_be:_" + FG_VAL);
49 |     //returnArr[x][y] = FG_VAL;
50 | }
51 |
52 |
53 | while (!processingStack.empty()) {
54 |     Point nextPos = processingStack.pop();
55 |
56 |     //check all children in N4
57 |     for (int xOffset = -1; xOffset <= 1; xOffset++) {
58 |         for (int yOffset = -1; yOffset <= 1; yOffset++) {
59 |             int nbX = nextPos.x + xOffset;
60 |             int nbY = nextPos.y + yOffset;
61 |
62 |             // check if N4 region
63 |             boolean isRegion = false;
64 |             if (region.equals("N4") && (xOffset*yOffset == 0 && xOffset+yOffset
65 |                 ↳ != 0)) isRegion = true;
66 |             if (region.equals("N8") && (xOffset != 0 || yOffset != 0) isRegion =
67 |                 ↳ true;
68 |             if (isRegion) {
69 |                 // check if valid range ==> position within image boundaries
70 |                 if (nbX >= 0 && nbY >= 0 && nbX < width && nbY < height) {
71 |
72 |                     int nbVal = inImgArr[nbX][nbY];
73 |
74 |                     //if current pixel was not processed yet (check if
75 |                     ↳ pixel is unprocessed and if value in
76 |                     ↳ threshold range)
77 |                     if (returnArr[nbX][nbY] == UNPROCESSED_VAL) {
78 |
79 |                         //if range valid
80 |                         if (nbVal >= lowerThresh && nbVal <=
81 |                             ↳ upperThresh) {
82 |                             returnArr[nbX][nbY] = FG_VAL;
83 |                             ↳ current pixel to foreground
84 |                             processingStack.push(new Point(nbX,
85 |                             ↳ nbY)); // push current

```

```

81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
↪ pixel to the stack
}
else {
    returnArr[nbX][nbY] = BG_VAL;
}
}
} // if N4 region
} // for yOffset
} // for xOffset
} // while processed all pixels of growing region
} // for height -> y
} // for width -> x
}
System.out.println("processingStack.size()");
//cleanup - all values still unprocessed - get assigned the background value BG_VAL
for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
        if (returnArr[x][y] == UNPROCESSED_VAL) {
            returnArr[x][y] = BG_VAL;
        }
    }
}
return returnArr;
} //performRegionGrowing
public void run(ImageProcessor ip) {
    byte[] pixels = (byte[]) ip.getPixels();
    int width = ip.getWidth();
    int height = ip.getHeight();
    int[][] inDataArrInt = ImageJUtility.convertFrom1DByteArray(pixels, width, height);
    // user input - default
    int lowerThresh = 100;

```

```

125 | int upperThresh = 255;
126 | // user dialog
127 | GenericDialog gd = new GenericDialog("thresh-params");
128 | gd.addSlider("lower_thresh", 0, 255, lowerThresh);
129 | gd.addSlider("upper_thresh", 0, 255, upperThresh);
130 | gd.showDialog();
131 |
132 | if (!gd.wasCanceled()) {
133 |     lowerThresh = (int) gd.getNextNumber();
134 |     upperThresh = (int) gd.getNextNumber();
135 | }
136 |
137 | //finally calling function
138 | int[][] resultImg = performRegionGrowing(inDataArrInt, width, height, lowerThresh, upperThresh, "N8");
139 |
140 |
141 | ImageJUtility.showNewImage(resultImg, width, height, "region_coin_result");
142 |
143 | } // run
144 |
145 | void showAbout() {
146 |     IJ.showMessage("About_Template...", "this_is_a_PluginFilter_template\n");
147 |     // showAbout
148 |
149 | } // class FilterTemplate_

```

4.2 Optimaler Threshold

Der zum Start benötigte initiale Threshold wird anhand eines User Inputs definiert. Wird kein Input angegeben wird als Default-Wert 127 gesetzt ($255/2$), somit startet die Berechnung mittig vom Histogramm.

Um mit der Berechnung des optimalen Threshold starten zu können wurden zwei Konstanten FG_VAL und BG_VAL definiert, die die jeweils kleinste und höchste Intensität enthalten. Bei jeder Iteration wird geprüft, ob der aktuelle Wert des Eingangsbildes über oder unter dem Threshold Wert. Liegt der Wert darunter, wird dieser in der sumThresh01 Variable aufsummiert und der Counter erhöht. Ist der Wert darüber, passiert dasselbe nur mit anderen Variablen.

Die aufsummierten Werte der zwei Bereiche werden nun benötigt um nach durlaufen des Eingangsbildes den Durchschnitt zu berechnen. Der vorübergehende Threshold wird aus den beiden meanThresh01, meanThresh02 berechnet. Nun wird geprüft, ob das Delta das sich zwischen dem vorübergehenden und initialen Threshold ergibt, über der definierten Konvergenz (DELTA_VAL), in diesem Fall 0.01, liegt. Wenn ja, wird ein erneuter Schleifendurchlauf gestartet. Wenn die Werte gleich sind, wird die Schleife abgebrochen und das berechnete Ausgangsbild ausgegeben.


initialer Threshold Wert	Eingangsbild	Optimal Threshold	Threshold Bild
60		117.81496778947292;	
127		120.52898065546574	

Table 3: Optimal Threshold Testbilder

Listing 3: Optimal Threshold Algorithmus

```

1      ; columns
2  import ij.*;
3  import ij.gui.GenericDialog;
4  import ij.plugin.filter.PlugInFilter;
5  import ij.process.*;
6
7  public class OptimalThreshold_ implements PlugInFilter {
8
9      public int setup(String arg, ImagePlus imp) {
10         if (arg.equals("about")) {
11             showAbout();
12             return DONE;
13         }
14         return DOES_8G + DOES_STACKS + SUPPORTS_MASKING;
15     } // setup
16
17     public int[][] performOptimalThresh(int[][] inImg, int width, int height, int
        ↪ BG_VAL, int FG_VAL,

```

```

18         double initialThresh, double DELTA_VAL) {
19     int [][] resultImg = new int [width][height];
20     double sumThresh01 = 0;
21     double sumThresh02 = 0;
22     int countThresh01 = 0;
23     int countThresh02 = 0;
24     double meanThresh01 = 0;
25     double meanThresh02 = 0;
26     double intermediateThresh = 0;
27
28     int loopCount = 0;
29
30     // calculate intermediate threshold value and check
31
32     while (true) {
33         for (int x = 0; x < width; x++) {
34             for (int y = 0; y < height; y++) {
35                 int currVal = inImg[x][y];
36
37                 if (currVal < initialThresh) {
38                     sumThresh01 += currVal;
39                     countThresh01++;
40
41                 } else {
42
43                     sumThresh02 += currVal;
44                     countThresh02++;
45                 }
46             }
47         }
48     }
49
50     // calculate mean
51     meanThresh01 = (sumThresh01 / countThresh01);
52     meanThresh02 = (sumThresh02 / countThresh02);
53
54     // calculate intermediate threshold
55     intermediateThresh = (meanThresh01 + meanThresh02) / 2;
56     loopCount++;
57     System.out.println("intermediate_thresh="+intermediateThresh+"
58         ↳ _Iteration="+loopCount);
59
60     if (Math.abs((initialThresh-intermediateThresh))>DELTA_VAL) {
61         initialThresh = intermediateThresh;
62     } else {
63         break;
64     }
65
66 }
67
68 // calculate result image
69 for (int x = 0; x < width; x++) {
70     for (int y = 0; y < height; y++) {
71         int currVal = inImg[x][y];
72
73         if (currVal < initialThresh) {
74             resultImg[x][y] = BG_VAL;
75
76         } else {
77             resultImg[x][y] = FG_VAL;
78         }
79     }

```

```

80         }
81     }
82 }
83
84     return resultImg;
85 }
86
87 public void run(ImageProcessor ip) {
88     byte[] pixels = (byte[]) ip.getPixels();
89     int width = ip.getWidth();
90     int height = ip.getHeight();
91
92     // user input
93     double initialThresh = 255 / 2;
94
95     // constants
96     int BG_VAL = 0;
97     int FG_VAL = 255;
98     double DELTA_VAL = 0.01;
99
100     GenericDialog gd = new GenericDialog("thresh_params");
101     gd.addNumericField("Initial_Threshold_Value=", initialThresh, 0);
102     gd.showDialog();
103
104     if (!gd.wasCanceled()) {
105         initialThresh = (int) gd.getNextNumber();
106     }
107
108     System.out.println("initial_Threshold_Value=" + initialThresh);
109
110     int[][] inDataArrInt = ImageJUtility.convertFrom1DByteArray(pixels, width,
111         ↪ height);
112     int[][] resultImg = performOptimalThresh(inDataArrInt, width, height,
113         ↪ BG_VAL, FG_VAL, initialThresh, DELTA_VAL);
114
115     // inDataArrInt = ImageJUtility.convertFrom1DByteArray(pixels, width,
116         ↪ height);
117
118     ImageJUtility.showNewImage(resultImg, width, height, "threshold_image");
119 } // run
120
121 void showAbout() {
122     IJ.showMessage("About_Template...", "this_is_a_PluginFilter_template\n"
123         ↪ );
124 } // showAbout
125
126 } // class FilterTemplate_

```

a) Adaptiver optimaler Threshold

Um den Filter des Optimal Threshold auch bei Bildern gut einsetzen zu können, die unterschiedliche Intensitätsverteilung pro Sektor haben, wurde der Adaptive Threshold-Filter implementiert.

Dieser verwendet die Funktionalität des Optimal Threshold wendet diese jedoch auf Sektoren mit einer Größe von 100x100 Pixel an. Die Sektoren werden aus dem Bild herausgeschnitten, der Optimal Threshold berechnet

und zwischengespeichert. Abschließend werden Sie zu einem Bild zusammen gesetzt und ausgegeben.





initialer Threshold Wert	Eingangsbild	Threshold Bild
60		
127		

Table 4: Adaptive Threshold Testbilder

Listing 4: Adaptive Threshold Algorithmus

```

; columns
1
2 import java.awt.Rectangle;
3
4 import ij.*;
5 import ij.gui.GenericDialog;
6 import ij.plugin.filter.PlugInFilter;
7 import ij.process.*;
8
9 public class AdaptiveThreshold_ implements PlugInFilter {
10
11     public int setup(String arg, ImagePlus imp) {
12         if (arg.equals("about")) {
13             showAbout();

```



```

14         return DONE;
15     }
16     return DOES_SG + DOES_STACKS + SUPPORTS_MASKING;
17 } // setup
18
19 public void run(ImageProcessor ip) {
20     byte[] pixels = (byte[]) ip.getPixels();
21     int width = ip.getWidth();
22     int height = ip.getHeight();
23
24     // user input
25     double initialThresh = 255 / 2;
26
27     // constants
28     int BG_VAL = 0;
29     int FG_VAL = 255;
30     double DELTA_VAL = 0.01;
31     int maskSize = 100;
32
33     GenericDialog gd = new GenericDialog("thresh_params");
34     gd.addNumericField("Initial_Threshold_Value:", initialThresh, 0);
35     gd.showDialog();
36
37     if (!gd.wasCanceled()) {
38         initialThresh = (int) gd.getNextNumber();
39     }
40
41     //System.out.println("initial Threshold Value = " + initialThresh);
42
43     int[][] inDataArrInt = ImageJUtility.convertFrom1DByteArr(pixels, width,
44         ↪ height);
45     double[][] inDataArrDouble = ImageJUtility.convertToDoubleArr2D(
46         ↪ inDataArrInt, width, height);
47     double[][] resultImg = new double[width][height];
48
49     int xCount = (width / maskSize);
50     int yCount = (height / maskSize);
51
52     int xPos = 0;
53     for (int x = 0; x < xCount; x++) {
54         int yPos = 0;
55         for (int y = 0; y < yCount; y++) {
56             int rWidth = maskSize;
57             int rHeight = maskSize;
58             if (xPos+rWidth > width) {
59                 rWidth-=width % maskSize;
60             }
61             if (yPos+rHeight > height) {
62                 rHeight-=height % maskSize;
63             }
64             Rectangle r = new Rectangle(xPos, yPos, maskSize,
65                 ↪ maskSize);
66
67             double[][] mask = ImageJUtility.cropImage(
68                 ↪ inDataArrDouble, rWidth, rHeight, r);
69             double[][] tempResult = performOptimalThresh(mask,
70                 ↪ rWidth, rHeight, BG_VAL, FG_VAL, initialThresh,
71                 ↪ DELTA_VAL);

```

```

72         resultImg[uPos][vPos] = tempResult[u][v
73             ↪ ];
74     }
75     yPos += maskSize;
76 }
77 xPos += maskSize;
78 }
79
80 //double [][] resultImg = performOptimalThresh(inDataArrDouble, width,
81     ↪ height, BG_VAL, FG_VAL, initialThresh, DELTA_VAL);
82
83 // inDataArrInt = ImageJUtility.convertFrom1DByteArr(pixels, width,
84     ↪ height);
85
86 ImageJUtility.showNewImage(resultImg, width, height, "threshold_image");
87
88 } // run
89
90
91 public double [][] performOptimalThresh(double [][] inImg, int width, int height,
92     ↪ int BG_VAL, int FG_VAL,
93     double initialThresh, double DELTA_VAL) {
94     double [][] resultImg = new double[width][height];
95     double sumThresh01 = 0;
96     double sumThresh02 = 0;
97     int countThresh01 = 0;
98     int countThresh02 = 0;
99     double meanThresh01 = 0;
100    double meanThresh02 = 0;
101    double intermediateThresh = 0;
102
103    int loopCount = 0;
104
105    // calculate intermediate threshold value and check
106
107    while (true) {
108        for (int x = 0; x < width; x++) {
109            for (int y = 0; y < height; y++) {
110                double currVal = inImg[x][y];
111
112                if (currVal < initialThresh) {
113                    sumThresh01 += currVal;
114                    countThresh01++;
115
116                } else {
117
118                    sumThresh02 += currVal;
119                    countThresh02++;
120                }
121            }
122        }
123
124    }
125
126    // calculate mean
127    meanThresh01 = (sumThresh01 / countThresh01);
128    meanThresh02 = (sumThresh02 / countThresh02);
129
130    // calculate intermediate threshold
131    intermediateThresh = (meanThresh01 + meanThresh02) / 2;
    loopCount++;

```

```

132         //System.out.println("intermediate thresh= " +
133             ↪ intermediateThresh + "; Iteration= " + loopCount);
134
135         if (Math.abs((initialThresh - intermediateThresh)) > DELTA_VAL)
136             ↪ {
137             initialThresh = intermediateThresh;
138         } else {
139             break;
140         }
141     }
142     // calculate result image
143     for (int x = 0; x < width; x++) {
144         for (int y = 0; y < height; y++) {
145             double currVal = inImg[x][y];
146
147             if (currVal < initialThresh) {
148                 resultImg[x][y] = BG_VAL;
149
150             } else {
151                 resultImg[x][y] = FG_VAL;
152             }
153
154         }
155     }
156
157     return resultImg;
158 }
159
160
161
162 void showAbout() {
163     IJ.showMessage("About_Template...", "this_is_a_PluginFilter_template\n"
164         ↪ );
165 } // showAbout
166
167 } // class FilterTemplate_

```