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 lediglisch 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 unersuchenden 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.

```
      ; columns
1
2  import java.awt.Point;
3  import java.awt.Rectangle;
4  import java.util.Stack;
5
6  import ij.*;
7  import ij.gui.GenericDialog;
8  import ij.gui.PointRoi;
9  import ij.plugin.filter.PlugInFilter;
10 import ij.process.*;
11
12 public class RegionGrowing_ implements PlugInFilter {
13
14         ImagePlus imp1;
15
16         public int setup(String arg, ImagePlus imp) {
17                 if (arg.equals("about")) {
18                         showAbout();
19                         return DONE;
20                 }
21
22                 imp1 = imp;
23                 return DOES_8G + DOES_STACKS + SUPPORTS_MASKING + ROI_REQUIRED;
24         } // setup
25
26         public static int[][] performRegionGrowing(int[][] inImgArr, int width, int height, int lowerThresh, int upperThresh
               ↪ , int seedX, int seedY, String region) {
27                 // constants
28                 int BG_VAL = 0;
29                 int FG_VAL = 255;
30                 int UNPROCESSED_VAL = -1;
31
32                 int[][] returnArr = new int[width][height];
33
34                 for( int x= 0; x < width; x++) {
35                         for (int y = 0; y < height; y++) {
36                                 returnArr[x][y] = UNPROCESSED_VAL;
37                         }
38                 }
39
40                 Stack<Point> processingStack = new Stack<Point>();
41
42                 //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 vlaue 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;
                                                                                ↪          //set current pixel to foreground
77 |                                                                         processingStack.push(new Point(nbX,nbY));        //
                                                                                ↪ push current pixel to the stack
78 |                                                                 }
79 |                                                                 else {
80 |                                                                         returnArr[nbX][nbY] = BG_VAL;
81 |                                                                 }
82 |                                                         }
83 |                                                 }
```

```
                                        } // if N4 region
                                }//for yOffset
                        }// for xOffset

                } //while

                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.convertFrom1DByteArr(pixels, width, height);

                //request seed point
                PointRoi pr = (PointRoi)imp1.getRoi();
                Rectangle rect = pr.getBounds();
                int xStart = pr.getXCoordinates()[0] + rect.x;
                int yStart = pr.getXCoordinates()[0] + rect.y;

                System.out.println("xStart : " + xStart +  ", yStart : " + yStart);

                // user input - default
                int lowerThresh = 100;
                int upperThresh = 255;
                // user dialog
                GenericDialog gd = new GenericDialog("thresh_params");
                gd.addSlider("lower_thresh", 0, 255, lowerThresh);
                gd.addSlider("upper_thresh", 0, 255, upperThresh);
                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,
    |                     ↪ yStart,"N4");
137 |
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 Implmentierung 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.

```
      ; columns
1
2  import java.awt.Point;
3  import java.awt.Rectangle;
4  import java.util.Stack;
5
6  import ij.*;
7  import ij.gui.GenericDialog;
8  import ij.gui.PointRoi;
9  import ij.plugin.filter.PlugInFilter;
10 import ij.process.*;
11
12 public class AutoRegionGrowing_ implements PlugInFilter {
13
14    public int setup(String arg, ImagePlus imp) {
15                  if (arg.equals("about"))
16                          {showAbout(); return DONE;}
17                  return DOES_8G+DOES_STACKS+SUPPORTS_MASKING;
18          } //setup
19
20
8 21         public static int[][] performRegionGrowing(int[][] inImgArr, int width, int height, int lowerThresh, int upperThresh
                ↪ , String region) {
22                  // constants
23                  int BG_VAL = 0;
24                  int FG_VAL = 255;
25                  int UNPROCESSED_VAL = -1;
26
27                  int[][] returnArr = new int[width][height];
28
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++) {
42
```

```
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
                                                ↪ != 0)) isRegion = true;
65                                          if(region.equals("N8") && (xOffset != 0 || yOffset != 0)) isRegion =
                                                ↪  true;
66
67                                          if(isRegion) {
68
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
                                                                ↪ pixel is unprocessed and if vlaue in
                                                                ↪ threshold range)
75                                                          if(returnArr[nbX][nbY] == UNPROCESSED_VAL) {
76
77                                                                  //if range valid
78                                                                  if(nbVal >= lowerThresh && nbVal <=
                                                                        ↪ upperThresh) {
79                                                                          returnArr[nbX][nbY] = FG_VAL;
                                                                                ↪                       //set
                                                                                ↪  current pixel to foreground
80                                                                          processingStack.push(new Point(nbX,
                                                                                ↪ nbY));        // push current
```

```
                                                                                        ↪ 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 hight -> y
        } //ffor 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.convertFrom1DByteArr(pixels, width, height);

        // user input - default
        int lowerThresh = 100;
```

```
                int upperThresh = 255;
                // user dialog
                GenericDialog gd = new GenericDialog("thresh_params");
                gd.addSlider("lower_thresh", 0, 255, lowerThresh);
                gd.addSlider("upper_thresh", 0, 255, upperThresh);
                gd.showDialog();

                if (!gd.wasCanceled()) {
                        lowerThresh = (int) gd.getNextNumber();
                        upperThresh = (int) gd.getNextNumber();
                }

                //finally calling function
                int[][] resultImg = performRegionGrowing(inDataArrInt, width, height, lowerThresh, upperThresh,"N8");


                ImageJUtility.showNewImage(resultImg, width, height, "region_coin_result");

        } // run

        void showAbout() {
                IJ.showMessage("About_Template_...", "this_is_a_PluginFilter_template\n");
        } // showAbout

} // class FilterTemplate_
```

4.2 Optimaler Threshold

Listing 3: Optimal Threshold Algorythmus.

```
; columns
1
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
51                         // calculate mean
52                         meanThresh01 = (sumThresh01 / countThresh01);
53                         meanThresh02 = (sumThresh02 / countThresh02);
54
55                         //calculate intermediate threshold
56                         intermediateThresh = (meanThresh01 + meanThresh02) / 2;
57                         loopCount++;
```

```
58                      System.out.println("intermediate thresh=" +intermediateThresh+"; 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.convertFrom1DByteArr(pixels, width, height);
111             int[][] resultImg = performOptimalThresh(inDataArrInt, width, height, BG_VAL, FG_VAL, initialThresh, DELTA_VAL);
112
113             // inDataArrInt = ImageJUtility.convertFrom1DByteArr(pixels, width, height);
114
115             ImageJUtility.showNewImage(resultImg, width, height, "threshold image");
116
117     } // run
```

```
118 |                                                                                                                  |
119 |         void showAbout() {                                                                                       |
120 |                 IJ.showMessage("About Template ...", "this is a PluginFilter template\n"                         |
    |                     ↪ );                                                                                         |
121 |         } // showAbout                                                                                           |
122 |                                                                                                                  |
123 | } // class FilterTemplate_                                                                                       |
```

a) Adaptiver optimar Threshold

Listing 4: Adaptive Threshold Algorythmus.

```
; 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_8G + 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,                        |
   |                     ↪ height);                                                                                  |
44 |                 double[][] inDataArrDouble = ImageJUtility.convertToDoubleArr2D(                                |
   |                     ↪ inDataArrInt, width, height);                                                             |
45 |                 double[][] resultImg = new double[width][height];                                               |
```

```
		int xCount = (width / maskSize);
		int yCount = (height / maskSize);

		int xPos = 0;
		for (int x = 0; x < xCount; x++) {
			int yPos = 0;
			for (int y = 0; y < yCount; y++) {
				int rWidth = maskSize;
				int rHeight = maskSize;
				if(xPos+rWidth > width) {
					rWidth-=width % maskSize;
				}
				if(yPos+rHeight > height) {
					rHeight-=height % maskSize;
				}
				Rectangle r = new Rectangle(xPos, yPos, maskSize,
				     ↪ maskSize);

				double[][] mask = ImageJUtility.cropImage(
				     ↪ inDataArrDouble, rWidth, rHeight, r);
				double[][] tempResult = performOptimalThresh(mask,
				     ↪ rWidth, rHeight, BG_VAL, FG_VAL, initialThresh,
				     ↪ DELTA_VAL);


				for (int u = 0; u < rWidth; u++) {
					for (int v = 0; v < rHeight; v++) {
						int uPos = xPos + u;
						int vPos = yPos + v;
						resultImg[uPos][vPos] = tempResult[u][v
						     ↪ ];
					}
				}
				yPos += maskSize;
			}
			xPos += maskSize;
		}

		//double[][] resultImg = performOptimalThresh(inDataArrDouble, width,
		     ↪ height, BG_VAL, FG_VAL, initialThresh, DELTA_VAL);

		// inDataArrInt = ImageJUtility.convertFrom1DByteArr(pixels, width,
		     ↪ height);

		ImageJUtility.showNewImage(resultImg, width, height, "threshold_image");

	} // run




	public double[][] performOptimalThresh(double[][] inImg, int width, int height,
	     ↪ int BG_VAL, int FG_VAL,
			double initialThresh, double DELTA_VAL) {
		double[][] resultImg = new double[width][height];
		double sumThresh01 = 0;
		double sumThresh02 = 0;
		int countThresh01 = 0;
		int countThresh02 = 0;
		double meanThresh01 = 0;
		double meanThresh02 = 0;
		double intermediateThresh = 0;

		int loopCount = 0;
```

```
103 |                                                                                                                 |
104 |                 // calculate intermediate threshold value and check                                            |
105 |                                                                                                                 |
106 |                 while (true) {                                                                                  |
107 |                         for (int x = 0; x < width; x++) {                                                       |
108 |                                 for (int y = 0; y < height; y++) {                                              |
109 |                                         double currVal = inImg[x][y];                                           |
110 |                                                                                                                 |
111 |                                         if (currVal < initialThresh) {                                          |
112 |                                                 sumThresh01 += currVal;                                         |
113 |                                                 countThresh01++;                                                |
114 |                                                                                                                 |
115 |                                         } else {                                                                |
116 |                                                                                                                 |
117 |                                                 sumThresh02 += currVal;                                         |
118 |                                                 countThresh02++;                                                |
119 |                                         }                                                                       |
120 |                                                                                                                 |
121 |                                 }                                                                               |
122 |                                                                                                                 |
123 |                         }                                                                                       |
124 |                                                                                                                 |
125 |                         // calculate mean                                                                       |
126 |                         meanThresh01 = (sumThresh01 / countThresh01);                                           |
127 |                         meanThresh02 = (sumThresh02 / countThresh02);                                           |
128 |                                                                                                                 |
129 |                         // calculate intermediate threshold                                                     |
130 |                         intermediateThresh = (meanThresh01 + meanThresh02) / 2;                                 |
131 |                         loopCount++;                                                                            |
132 |                         //System.out.println("intermediate thresh= " +                                         |
    |                             ↪ intermediateThresh + "; Iteration= " + loopCount);                                |
133 |                                                                                                                 |
134 |                         if (Math.abs((initialThresh - intermediateThresh)) > DELTA_VAL)                         |
    |                             ↪ {                                                                                 |
135 |                                 initialThresh = intermediateThresh;                                             |
136 |                         } else {                                                                                |
137 |                                 break;                                                                          |
138 |                         }                                                                                       |
139 |                                                                                                                 |
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 |                                                                                                                 |
158 |                 return resultImg;                                                                               |
159 |         }                                                                                                       |
160 |                                                                                                                 |
161 |                                                                                                                 |
162 |                                                                                                                 |
163 |         void showAbout() {                                                                                      |
164 |                 IJ.showMessage("About_Template_...", "this_is_a_PluginFilter_template\n"                        |
```

```
                    ↪ );
        } // showAbout
165
166
167 } // class FilterTemplate_
```