

Übungsaufgaben I, SBV1

Lisa Panholzer, Lukas Fiel

October 18, 2018

1 Gauss Filter

1.0.1 Code

```
import ij.*;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;

import ij.gui.GenericDialog;

public class Gauss_ 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) {

        // convert to pixel array
        byte[] pixels = (byte[])ip.getPixels();
        int width = ip.getWidth();
        int height = ip.getHeight();
        int tgtRadius = 4;
        int sigma = 4;

        int[] [] inArr = ImageJUtility.
            ↪ convertFrom1DByteArr(pixels, width,
            ↪ height);
        double[] [] inDataArrDouble = ImageJUtility.
            ↪ convertToDoubleArr2D(inArr, width, height
            ↪ );

        //user input for radius
```

```

        GenericDialog gd = new GenericDialog("user_
        ↪ input:");
        gd.addNumericField("radius", tgtRadius, 0);
        gd.showDialog();
        if(gd.wasCanceled()) {return;}
        tgtRadius = (int)gd.getNextNumber();

        double[] [] filterMask = ConvolutionFilter.
        ↪ GetGaussMask(tgtRadius,sigma);
        ImageJUtility.showNewImage(filterMask,
        ↪ filterMask.length, filterMask.length, "
        ↪ Gauss_Mask");

        //double[] [] resultImage = ConvolutionFilter.
        ↪ ConvolveDouble(inDataArrDouble, width,
        ↪ height, filterMask, tgtRadius);
        double[] [] resultImage = ConvolutionFilter.
        ↪ ConvolveDoubleNorm(inDataArrDouble, width
        ↪ , height, filterMask, tgtRadius);
        ImageJUtility.showNewImage(resultImage, width, height,
        ↪ "mean_with_kernel_r=" + tgtRadius);

    } //run

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

} //class FilterTemplate_

```

1.0.2 Ablauf und Idee

1.0.3 Tests und Sonderfälle

2 MedianFilter

2.0.1 Code

```
import ij.*;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;
import ij.gui.GenericDialog;
import java.awt.Rectangle;
import java.util.Arrays;

import com.sun.net.httpserver.Authenticator.Success;

public class Median_ 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) {

        System.out.println("RUN: □Plugin□Median");
        // convert to pixel array
        byte[] pixels = (byte[]) ip.getPixels();
        int width = ip.getWidth();
        int height = ip.getHeight();

        int[] [] inArr = ImageJUtility.
            ↪ convertFrom1DByteArr(pixels, width,
            ↪ height);
        double[] [] inDataArrDouble = ImageJUtility.
            ↪ convertToDoubleArr2D(inArr, width, height
            ↪ );
    }
}
```

```

int radius = getUserInputRadius(4);
// int radius = 2; // default value for
    ↪ debugging

if (2 * radius > width || 2 * radius > height)
    ↪ {
        System.out.println("Be aware that double
            ↪ the radius has to fit in the
            ↪ image!");
    }

double[][] resultImage = inDataArrDouble.clone
    ↪ ();
int successIndex = 0;
int failureIndex = 0;
// step1: move mask to all possible image
    ↪ pixel positions
for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
        double[][] mask = inDataArrDouble
            ↪ .clone();
        try {

            // roi = new Rectangle(x
            ↪ - radius, y -
            ↪ radius, size -
            ↪ deltaX - 1, size);
            Rectangle roi = getROI(
                ↪ width, height, x, y,
                ↪ radius);
            mask = ImageJUtility.
                ↪ cropImage(mask, roi.
                ↪ width, roi.height,
                ↪ roi);
            double median = getMedian(
                ↪ mask, roi.width, roi.
                ↪ height);

```

```

        resultImage[x][y] = median
        ↪ ;

        successIndex++;
    } catch (java.lang.
        ↪ ArrayIndexOutOfBoundsException
        ↪ exc) {
        // TODO: error handling
        ↪ for edge cases

        resultImage[x][y] =
            ↪ resultImage[x][y];
        failureIndex++;

    }

}

}

System.out.println("inputImg:␣width:␣" + width
    ↪ + "␣height:␣" + height + "␣surface:␣" +
    ↪ width * height);
System.out.println("SUCCESS:␣run␣over␣picture.␣
    ↪ succeed:␣" + successIndex + "␣failed:␣"
    ↪ + failureIndex
        + "␣sum:␣" + (int) (successIndex
            ↪ + failureIndex));
System.out.println("Now␣show␣the␣result␣image!"
    ↪ );
ImageJUtility.showNewImage(resultImage, width,
    ↪ height, "mean␣with␣kernel␣r=" + radius);
System.out.println("SUCCESS:␣MEDIAN␣FILTER␣DONE
    ↪ .");

} // run

void showAbout() {
    IJ.showMessage("About␣Template_...", "this␣is␣a
        ↪ ␣PluginFilter␣template\n");

```

```

} // showAbout

/**
 * get region of interest. defined by a Rectangle with
 *   ↪ x and y coordinates of the
 * upper left corner and width and height as parameters
 *   ↪ .
 *
 * @param width of the image
 * @param height of the image
 * @param x the x coordinate of the center of the mask
 * @param y the y coordinate of the center of the mask
 * @param radius of the mask
 * @return
 */
public static Rectangle getROI(int width, int height,
    ↪ int x, int y, int radius) {
    int xsize = 2 * radius + 1;
    int ysize = 2 * radius + 1;

    // special behaviour
    if (x - radius < 0) {
        xsize = xsize - (radius - x);
        x = radius;
    } // set minimum x
    if (y - radius < 0) {
        ysize = ysize - (radius - y);
        y = radius;
    } // set minimum y

    if (x + radius >= width) {
        int d = (radius - (width - x));
        xsize = xsize - d - 1 ;
    } // set maximum x
    if (y + radius >= height) {
        int d = (radius - (height - y));

```

```

        ysize = ysize - d - 1 ;
    } // set maximum y

    return new Rectangle(x - radius, y - radius,
        ↪ xsize, ysize);
}

public static double getMedian(double[][] inputImg, int
    ↪ width, int height) {
    int size = width * height;

    // fill array
    double[] arr = new double[size];
    int index = 0;
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            arr[index] = inputImg[i][j];
            index++;
        }
    }

    // sort array
    Arrays.sort(arr);
    // System.out.println("SUCCESS: getMedian.
        ↪ size: " + size);
    return arr[(int) (size / 2 + 1)];
}

/**
 * Asks the user to input a radius.
 *
 * @return radius from user input. 0 if failed.
 */
public static int getUserInputRadius(int defaultValue)
    ↪ {
    // user input
    System.out.println("Read user input: radius");

```



```

        GenericDialog gd = new GenericDialog("user_
        ↪ input:");
        gd.addNumericField("radius", defaultValue, 0);
        gd.showDialog();
        if (gd.wasCanceled()) {
            return 0;
        }
        return (int) gd.getNextNumber();
    }
} // class FilterTemplate_

```

2.0.2 Ablaufund Idee

2.0.3 Tests

3 Steuerung des Filtereffekts

3.0.1 Code

```
import ij.*;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;
import ij.gui.GenericDialog;
import java.awt.Rectangle;
import java.util.Arrays;

import com.sun.net.httpserver.Authenticator.Success;

public class Median_ 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) {

        System.out.println("RUN: □Plugin□Median");
        // convert to pixel array
        byte[] pixels = (byte[]) ip.getPixels();
        int width = ip.getWidth();
        int height = ip.getHeight();

        int[] [] inArr = ImageJUtility.
            ↪ convertFrom1DByteArr(pixels, width,
            ↪ height);
        double[] [] inDataArrDouble = ImageJUtility.
            ↪ convertToDoubleArr2D(inArr, width, height
            ↪ );
    }
}
```

```

int radius = getUserInputRadius(4);
// int radius = 2; // default value for
    ↪ debugging

if (2 * radius > width || 2 * radius > height)
    ↪ {
        System.out.println("Be aware that double
            ↪ the radius has to fit in the
            ↪ image!");
    }

double[][] resultImage = inDataArrDouble.clone
    ↪ ();
int successIndex = 0;
int failureIndex = 0;
// step1: move mask to all possible image
    ↪ pixel positions
for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
        double[][] mask = inDataArrDouble
            ↪ .clone();
        try {

            // roi = new Rectangle(x
            ↪ - radius, y -
            ↪ radius, size -
            ↪ deltaX - 1, size);
            Rectangle roi = getROI(
                ↪ width, height, x, y,
                ↪ radius);
            mask = ImageJUtility.
                ↪ cropImage(mask, roi.
                ↪ width, roi.height,
                ↪ roi);
            double median = getMedian(
                ↪ mask, roi.width, roi.
                ↪ height);

```

```

        resultImage[x][y] = median
        ↪ ;

        successIndex++;
    } catch (java.lang.
        ↪ ArrayIndexOutOfBoundsException
        ↪ exc) {
        // TODO: error handling
        ↪ for edge cases

        resultImage[x][y] =
            ↪ resultImage[x][y];
        failureIndex++;

    }

}

}

System.out.println("inputImg:␣width:␣" + width
    ↪ + "␣height:␣" + height + "␣surface:␣" +
    ↪ width * height);
System.out.println("SUCCESS:␣run␣over␣picture.␣
    ↪ succeed:␣" + successIndex + "␣failed:␣"
    ↪ + failureIndex
        + "␣sum:␣" + (int) (successIndex
            ↪ + failureIndex));
System.out.println("Now␣show␣the␣result␣image!"
    ↪ );
ImageJUtility.showNewImage(resultImage, width,
    ↪ height, "mean␣with␣kernel␣r=" + radius);
System.out.println("SUCCESS:␣MEDIAN␣FILTER␣DONE
    ↪ .");

} // run

void showAbout() {
    IJ.showMessage("About␣Template...", "this␣is␣a
        ↪ ␣PluginFilter␣template\n");
}

```

```

} // showAbout

/**
 * get region of interest. defined by a Rectangle with
 *   ↪ x and y coordinates of the
 * upper left corner and width and height as parameters
 *   ↪ .
 *
 * @param width of the image
 * @param height of the image
 * @param x the x coordinate of the center of the mask
 * @param y the y coordinate of the center of the mask
 * @param radius of the mask
 * @return
 */
public static Rectangle getROI(int width, int height,
    ↪ int x, int y, int radius) {
    int xsize = 2 * radius + 1;
    int ysize = 2 * radius + 1;

    // special behaviour
    if (x - radius < 0) {
        xsize = xsize - (radius - x);
        x = radius;
    } // set minimum x
    if (y - radius < 0) {
        ysize = ysize - (radius - y);
        y = radius;
    } // set minimum y

    if (x + radius >= width) {
        int d = (radius - (width - x));
        xsize = xsize - d - 1 ;
    } // set maximum x
    if (y + radius >= height) {
        int d = (radius - (height - y));

```

```

        ysize = ysize - d - 1 ;
    } // set maximum y

    return new Rectangle(x - radius, y - radius,
        ↪ xsize, ysize);
}

public static double getMedian(double[][] inputImg, int
    ↪ width, int height) {
    int size = width * height;

    // fill array
    double[] arr = new double[size];
    int index = 0;
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            arr[index] = inputImg[i][j];
            index++;
        }
    }

    // sort array
    Arrays.sort(arr);
    // System.out.println("SUCCESS: getMedian.
        ↪ size: " + size);
    return arr[(int) (size / 2 + 1)];
}

/**
 * Asks the user to input a radius.
 *
 * @return radius from user input. 0 if failed.
 */
public static int getUserInputRadius(int defaultValue)
    ↪ {
    // user input
    System.out.println("Read user input: radius");

```

```
        GenericDialog gd = new GenericDialog("user_␣  
        ↪ input:");  
        gd.addNumericField("radius", defaultValue, 0);  
        gd.showDialog();  
        if (gd.wasCanceled()) {  
            return 0;  
        }  
        return (int) gd.getNextNumber();  
    }  
} // class FilterTemplate_
```

3.0.2 Ablaufund Idee

3.0.3 Tests

4 Histogrammeinebnung

4.0.1 Code

```
import ij.*;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;
import ij.gui.GenericDialog;
import java.awt.Rectangle;
import java.util.Arrays;

import com.sun.net.httpserver.Authenticator.Success;

public class Median_ 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) {

        System.out.println("RUN: □Plugin□Median");
        // convert to pixel array
        byte[] pixels = (byte[]) ip.getPixels();
        int width = ip.getWidth();
        int height = ip.getHeight();

        int[] [] inArr = ImageJUtility.
            ↪ convertFrom1DByteArr(pixels, width,
            ↪ height);
        double[] [] inDataArrDouble = ImageJUtility.
            ↪ convertToDoubleArr2D(inArr, width, height
            ↪ );
    }
}
```



```

int radius = getUserInputRadius(4);
// int radius = 2; // default value for
    ↪ debugging

if (2 * radius > width || 2 * radius > height)
    ↪ {
        System.out.println("Be aware that double
            ↪ the radius has to fit in the
            ↪ image!");
    }

double[][] resultImage = inDataArrDouble.clone
    ↪ ();
int successIndex = 0;
int failureIndex = 0;
// step1: move mask to all possible image
    ↪ pixel positions
for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
        double[][] mask = inDataArrDouble
            ↪ .clone();
        try {

            // roi = new Rectangle(x
            ↪ - radius, y -
            ↪ radius, size -
            ↪ deltaX - 1, size);
            Rectangle roi = getROI(
                ↪ width, height, x, y,
                ↪ radius);
            mask = ImageJUtility.
                ↪ cropImage(mask, roi.
                ↪ width, roi.height,
                ↪ roi);
            double median = getMedian(
                ↪ mask, roi.width, roi.
                ↪ height);

```

```

        resultImage[x][y] = median
        ↪ ;

        successIndex++;
    } catch (java.lang.
        ↪ ArrayIndexOutOfBoundsException
        ↪ exc) {
        // TODO: error handling
        ↪ for edge cases

        resultImage[x][y] =
            ↪ resultImage[x][y];
        failureIndex++;

    }

}

}

System.out.println("inputImg:␣width:␣" + width
    ↪ + "␣height:␣" + height + "␣surface:␣" +
    ↪ width * height);
System.out.println("SUCCESS:␣run␣over␣picture.␣
    ↪ succeed:␣" + successIndex + "␣failed:␣"
    ↪ + failureIndex
        + "␣sum:␣" + (int) (successIndex
            ↪ + failureIndex));
System.out.println("Now␣show␣the␣result␣image!"
    ↪ );
ImageJUtility.showNewImage(resultImage, width,
    ↪ height, "mean␣with␣kernel␣r=" + radius);
System.out.println("SUCCESS:␣MEDIAN␣FILTER␣DONE
    ↪ .");

} // run

void showAbout() {
    IJ.showMessage("About␣Template...", "this␣is␣a
        ↪ ␣PluginFilter␣template\n");

```

```

} // showAbout

/**
 * get region of interest. defined by a Rectangle with
 *   ↪ x and y coordinates of the
 * upper left corner and width and height as parameters
 *   ↪ .
 *
 * @param width of the image
 * @param height of the image
 * @param x the x coordinate of the center of the mask
 * @param y the y coordinate of the center of the mask
 * @param radius of the mask
 * @return
 */
public static Rectangle getROI(int width, int height,
    ↪ int x, int y, int radius) {
    int xsize = 2 * radius + 1;
    int ysize = 2 * radius + 1;

    // special behaviour
    if (x - radius < 0) {
        xsize = xsize - (radius - x);
        x = radius;
    } // set minimum x
    if (y - radius < 0) {
        ysize = ysize - (radius - y);
        y = radius;
    } // set minimum y

    if (x + radius >= width) {
        int d = (radius - (width - x));
        xsize = xsize - d - 1 ;
    } // set maximum x
    if (y + radius >= height) {
        int d = (radius - (height - y));

```

```

        ysize = ysize - d - 1 ;
    } // set maximum y

    return new Rectangle(x - radius, y - radius,
        ↪ xsize, ysize);
}

public static double getMedian(double[][] inputImg, int
    ↪ width, int height) {
    int size = width * height;

    // fill array
    double[] arr = new double[size];
    int index = 0;
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            arr[index] = inputImg[i][j];
            index++;
        }
    }

    // sort array
    Arrays.sort(arr);
    // System.out.println("SUCCESS: getMedian.
        ↪ size: " + size);
    return arr[(int) (size / 2 + 1)];
}

/**
 * Asks the user to input a radius.
 *
 * @return radius from user input. 0 if failed.
 */
public static int getUserInputRadius(int defaultValue)
    ↪ {
    // user input
    System.out.println("Read user input: radius");

```

```

        GenericDialog gd = new GenericDialog("user_
        ↪ input:");
        gd.addNumericField("radius", defaultValue, 0);
        gd.showDialog();
        if (gd.wasCanceled()) {
            return 0;
        }
        return (int) gd.getNextNumber();
    }
} // class FilterTemplate_

```

4.0.2 Ablaufund Idee

4.0.3 Tests

5 Raster-Entfernung im Frequenzraum

5.1 Workflow

- Starten von *imageJ.exe*
- Öffnen eines Bildes
- $Process \rightarrow FFT \rightarrow FFT$
- Zuschneiden des interessanten Bereichs im FFT Bild
- $Process \rightarrow FFT \rightarrow inverse\ FFT$

5.2 Beispiele

5.2.1 Auge

Es wurde ein Bild gewählt, welches (wie bei einem Plakatdruck) Punkte in regelmässigen Abständen aufweist. Die eigentliche Bildinformation steckt in der Dicke der Punkte. Eine FFT Transformation zeigt deutlich ein periodisches Muster. Will man nur die eigentliche Bildinformation gewinnen, müssen hochfrequente Anteile des Bildes entfernt werden. Tabelle 1 zeigt deutlich dass durch ein Entfernen der Randbereiche (höhere Frequenzen) im FFT Bild und die anschließende Rücktransformation die eigentliche Bildinformation gewonnen werden konnte.

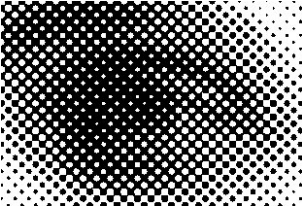
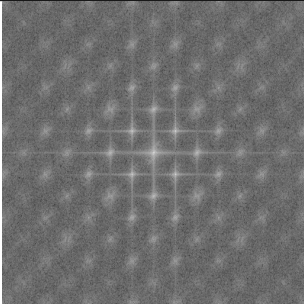

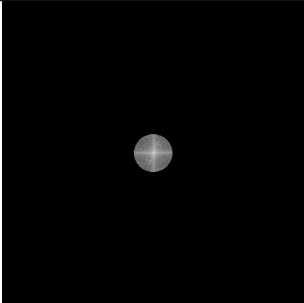
Bild	FFT
	
	

Table 1: Auswertung Auge

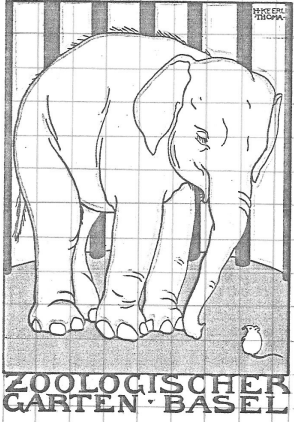
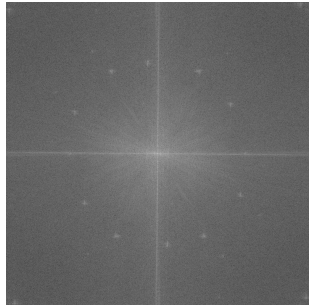
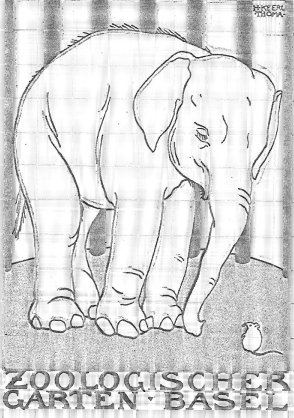
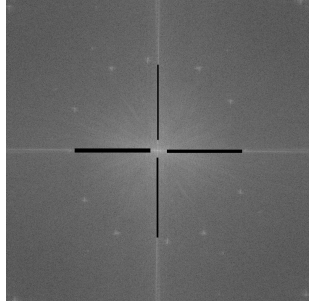
Bild	FFT
	
	

Table 2: Auswertung Elefant

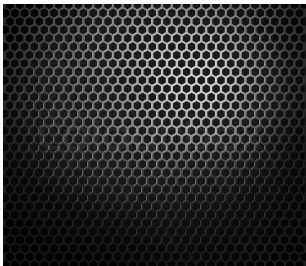
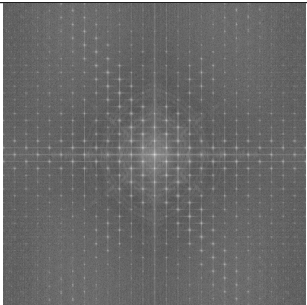

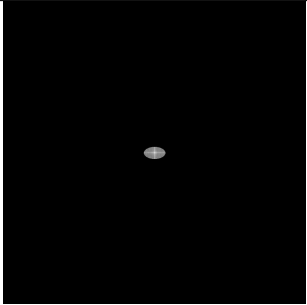
Bild	FFT
	
	

Table 3: Auswertung Lochgitter