

Signal und Bildverarbeitung

Übung 3
Gürsoy & Miesenböck

Aufgabe 3.1 Registrierung und Bildüberlagerung [32] Implementieren Sie einen Registrierungsalgorithmus mit ImageJ. Die beiden zu registrierenden Bildteile sind dabei horizontal versetzt im aktiven Eingangsbild enthalten. Zerteilen Sie daher das Eingangsbild mittig in eine linke und rechte Bildhälfte, um die Bilder A und B verfügbar zu haben. In Bezug auf die Rotation ist i.d.R. Padding notwendig, damit das Bild während der Transformation nicht temporär außerhalb des darstellbaren Bereiches liegt (weißer oder schwarzer „Hintergrund“ um die eigentlichen Bildmotive herum). Gehen Sie, wie in unterer Abbildung angedeutet, davon aus, dass bei den Eingangsbildern ausreichend Rand eingepflegt wurde. Sie müssen folglich die Thematik Padding nicht abhandeln.

a) [3] Zerteilen Sie das Eingangsbild mittig in die beiden Hälften (entlang der angedeuteten roten Linie) und zeigen Sie beide mittels ImageJ an.

Lösungsansatz:

Um das Eingangsbild in zwei Bilder zu teilen, muss zuerst einmal die Hälfte der Breite ermittelt werden. Die Höhe bleibt dabei unangetastet, da diese in dem Fall nicht geteilt werden muss. Zudem muss ermittelt werden, ob es sich um eine gerade oder ungerade Anzahl der Pixel der Breite handelt. Tritt der Fall ein, dass es sich um eine ungerade Anzahl handelt, muss der Start des zweiten Bildes um 1 erhöht werden, damit alle Pixel mit durchlaufen werden. Im Anschluss wird das Bild mit einer Doppelten-For-Schleife durchlaufen. Hier wurde automatisch auch das zweite Bild ermittelt, da zur Variable x direkt der Start des zweiten Bildes dazu addiert wird. Um beide Bilder wieder zurück zu geben, werden diese in ein array (imagePair) gespeichert und als RückgabevARIABLE deklariert.

```

public double[][][] splitImage(double[][][] image, int width, int height) {
    int halfWidth = (int) width/2;

    //second image
    //look if width is odd or even
    int startSecondImage = halfWidth + 1;
    if(width%2==0)
        startSecondImage = halfWidth - 1;

    double[][] retArrA = new double[halfWidth][height];
    double[][] retArrB = new double[halfWidth][height];

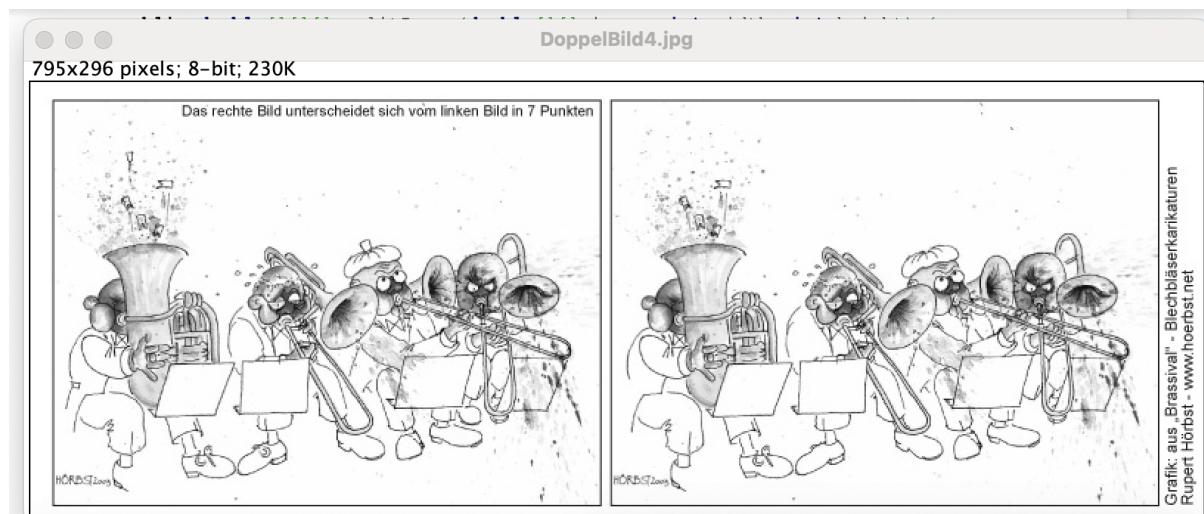
    for (int x = 0; x < halfWidth; x++) {
        for (int y = 0; y < height; y++) {
            retArrA[x][y] = image[x][y];
            retArrB[x][y] = image[startSecondImage+x][y];
        }
    }

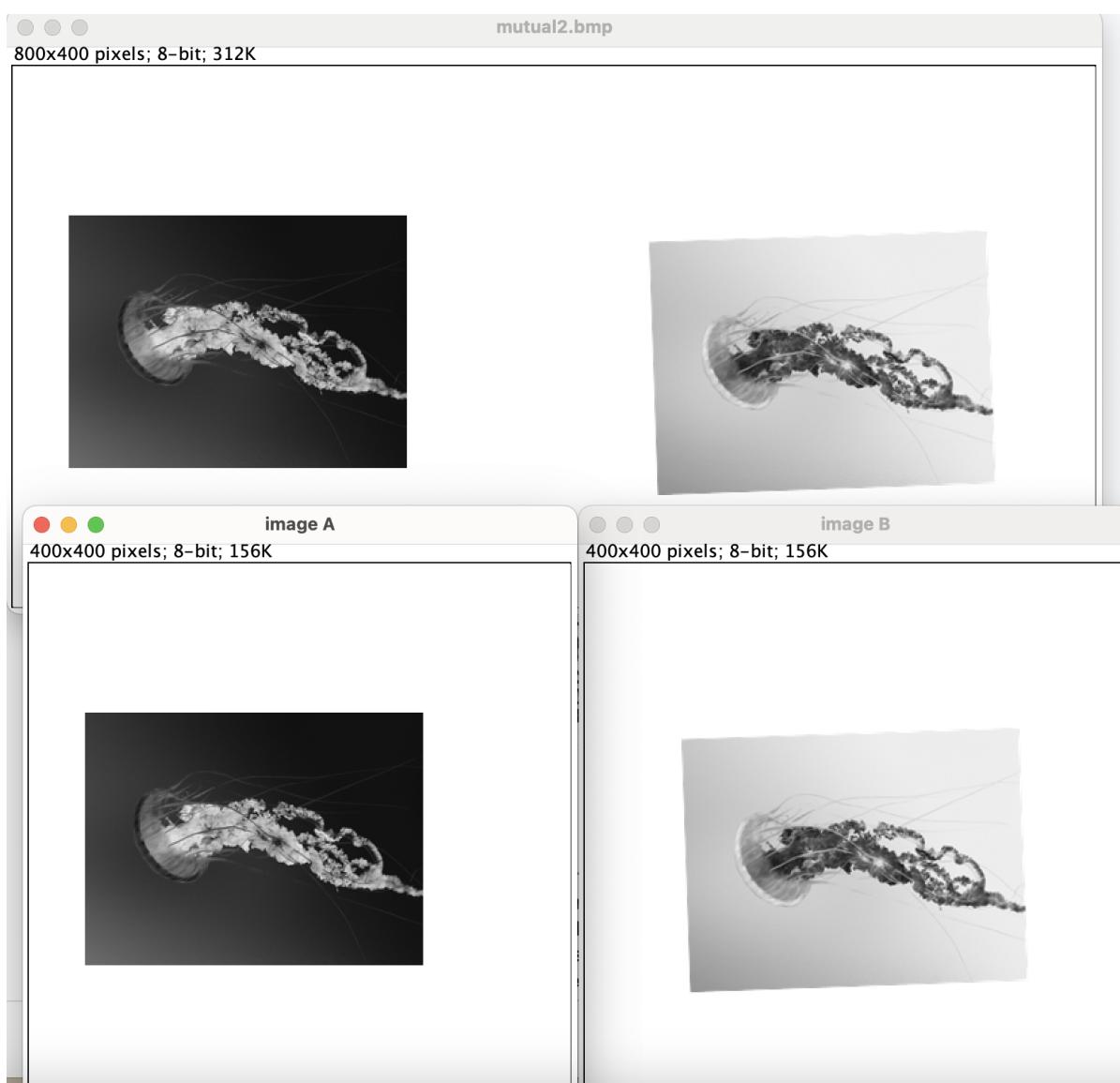
    double[][][] imagePair = new double[2][width][height];
    imagePair[0] = retArrA;
    imagePair[1] = retArrB;

    return imagePair;
}

```

Testfälle:





b) [6] Implementieren Sie Funktionalität, die die Transformation eines Eingangsbildes (Rotation, Translation) erlaubt. Die Transformation wird dabei durch deltaX, deltaY und einen Rotationswinkel bestimmt. Testen Sie Ihre Ergebnisse, indem Sie das Bild nur rotieren, nur verschieben bzw. Rotation und Translation nacheinander anwenden. Die Transformation soll für Zwischenergebnisse mittels NN-Interpolation der Koordinaten erfolgen – für das finale Ergebnis höchster Qualität ist Bilineare Interpolation anzuwenden.

Lösungsidee:

Da bei der Transformation die Möglichkeit bestehen sollte, zwischen NearestNeighbour und der BilineareTransformation auswählen zu können. Wird dieser als zusätzlicher Parameter in der Funktion übergeben. Je nachdem wird die jeweilige Interpolation aufgerufen.

Die Funktion moveImage werden dabei die jeweiligen Parameter übergeben, welche für das Rotieren und Bewegen des Bildes zuständig sind.

Funktionen:

```
public double[][] moveImage(double[][] inImg, int width, int height, double transX, double transY, double rotAngle, String methode) {

    double[][] retArr = new double[width][height];

    double radAngle = -rotAngle * Math.PI / 180.0; //Attention "--" for
    changing direction due tp backward mapping
    double cosTheta = Math.cos(radAngle);
    double sinTheta = Math.sin(radAngle);

    double widthHalf = width / 2.0;
    double heightHalf = height / 2.0;

    //backproject all pixels of result image B
    //iterate all pixels and calculate the new value
    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {

            //1) move coordinate to center
            double posX = x - widthHalf;
            double posY = y - heightHalf;

            //2) rotate
            double rotatedX = posX * cosTheta + posY * sinTheta;
            double rotatedY = -posX * sinTheta + posY * cosTheta;

            //3) move coordinate back from center
            rotatedX = rotatedX + widthHalf;
            rotatedY = rotatedY + heightHalf;

            //4) translate
            posX = rotatedX - transX;
```

```

        posY = rotatedY - transY;

        //finally get the interpolated pixel value
        double scalarVal = 0.0;
        if(methode.equals("N")){
            scalarVal = getNNInterpolatedValue(inImg, width, height,
                posX, posY);
        }
        else {
            scalarVal = getBilinearInterpolatedValue(inImg, width,
                height, posX, posY);
        }

        retArr[x][y] = scalarVal;
    }
}

```

Interpolation-Funktionen:

```

public int getBilinearInterpolatedValue(double[][] imgData, int width, int
height, double idxX, double idxY) {
    double saveX = idxX - (int) (idxX);
    double saveY = idxY - (int) (idxY);

    int x1 = (int) (idxX);
    int x2 = (int) (idxX + 0.5);
    int y1 = (int) (idxY);
    int y2 = (int) (idxY + 0.5);

    if (x1 >= 0 && x1 < width && x2 >= 0 && x2 < width && y1 >= 0 && y1 <
height && y2 >= 0 && y2 < height) {

        double interimResult1 =
calculatePointForBilineareInterpolation(imgData[x1][y1], imgData[x2][y1],
saveX);

        double interimResult2 =
calculatePointForBilineareInterpolation(imgData[x1][y2], imgData[x2][y2],
saveX);

        return (int) (calculatePointForBilineareInterpolation(interimResult1,
interimResult2, saveY));
    }

    return 0;
} //getBilinearInterpolatedValue

public double calculatePointForBilineareInterpolation(double point1,
double point2, double storagedPoint) {
    double pointDifference = Math.abs(point1-point2);
    double minPoint = Math.min(point1, point2);

```

```

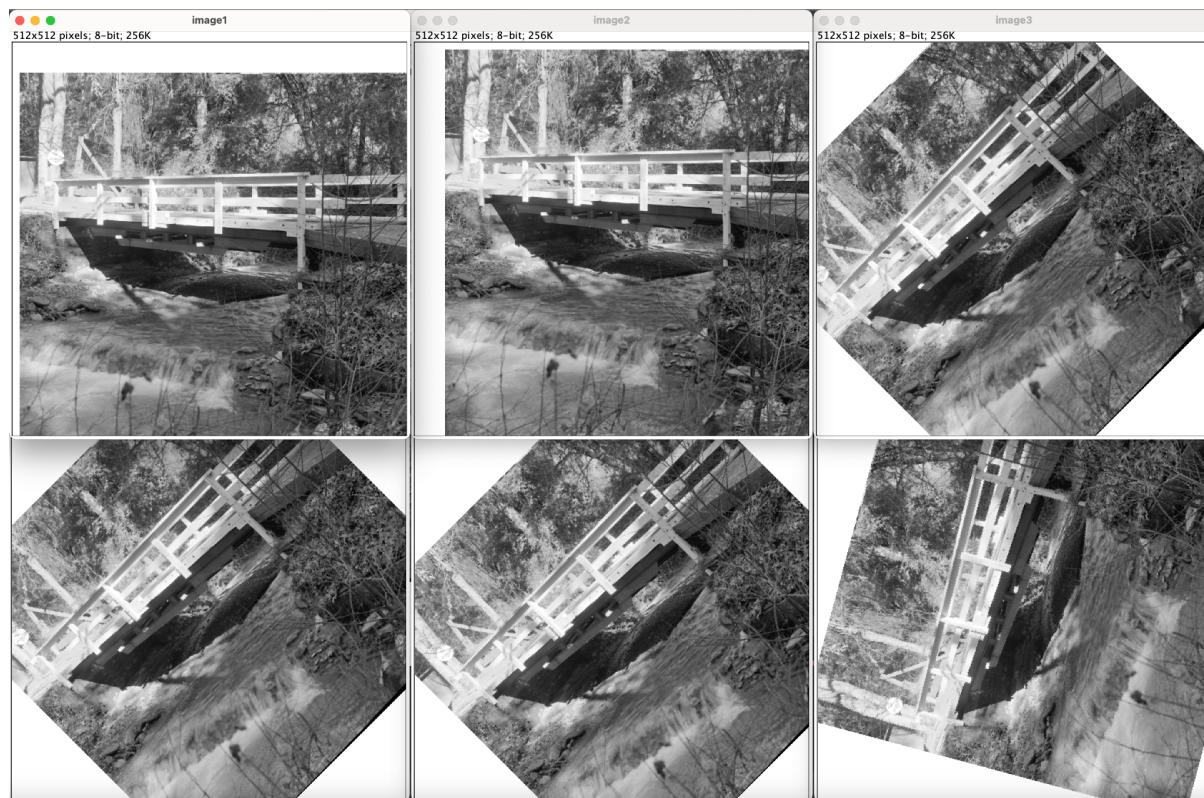
        double result = minPoint + storagedPoint * pointDifference;
        return result;
    }

    public double getNNInterpolatedValue(double[][] imgData, int width, int
height, double idxX, double idxY) {
        int x1 = (int) (idxX + 0.5);
        int y1 = (int) (idxY + 0.5);

        if ((x1 >= 0) && (y1 >= 0) && (x1 < width) && (y1 < height)) {
            double actVal = imgData[x1][y1];
            return actVal;
        } //if
        else {
            return 255.0;
        } //else ==> outside mask
    } //getNN
}

```

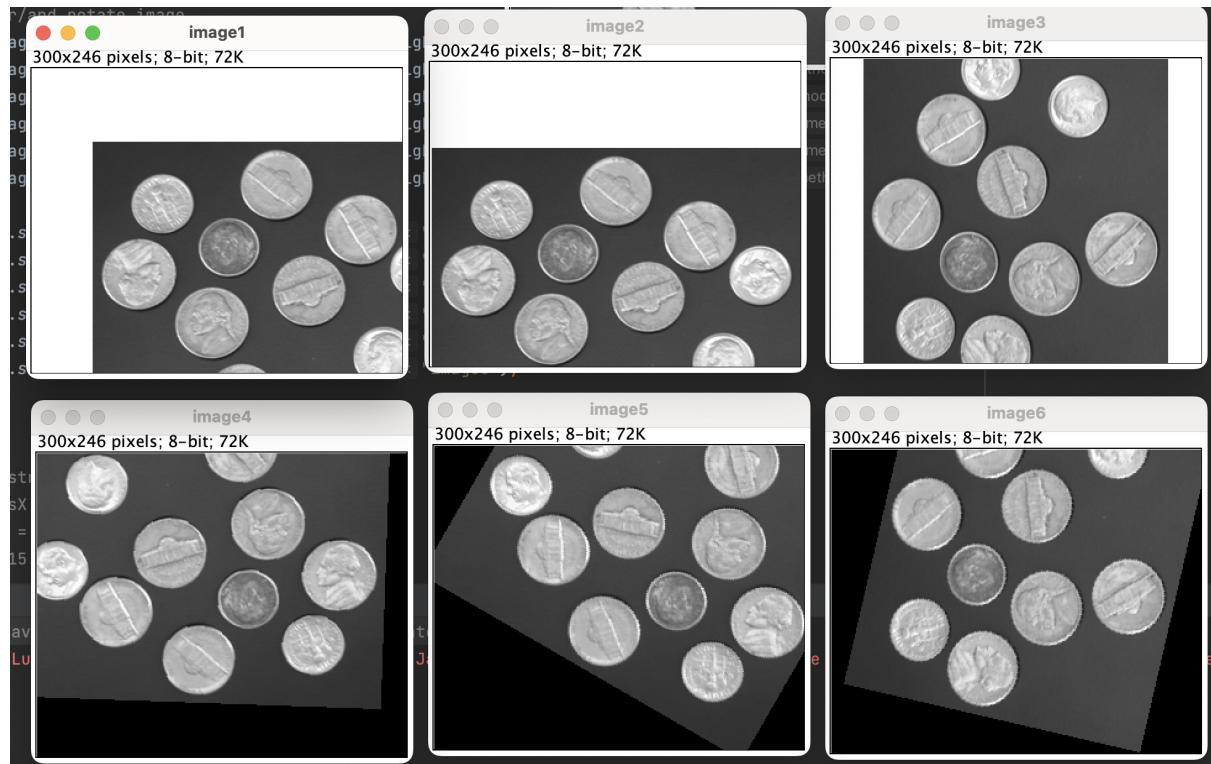
Testfälle:



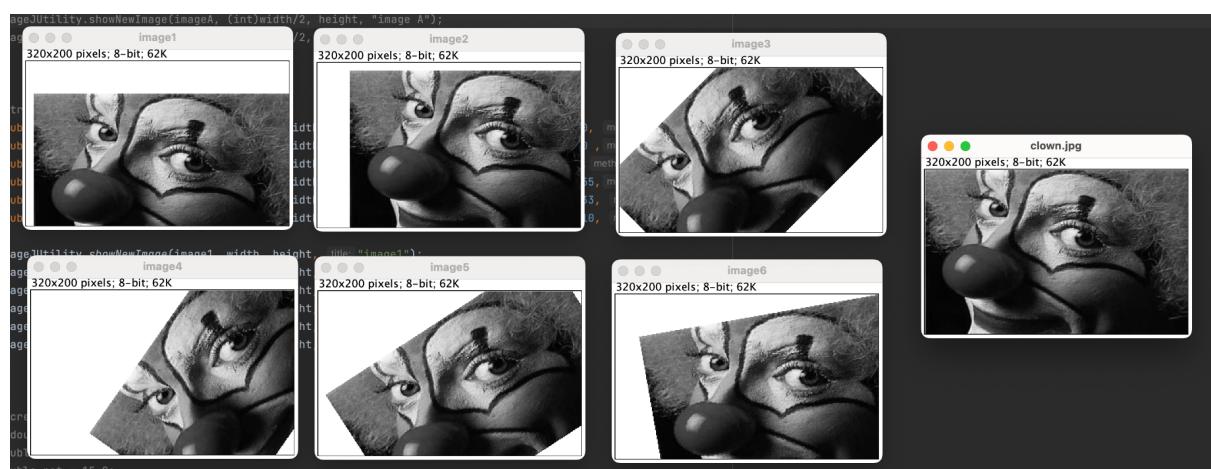
```

double[][] image1 = moveImage(inDataArrDbl, width, height, 10, 40, 0, "N");
double[][] image2 = moveImage(inDataArrDbl, width, height, 40, 10, 0, "N");
double[][] image3 = moveImage(inDataArrDbl, width, height, 0, 0, 45, "N");
double[][] image4 = moveImage(inDataArrDbl, width, height, 5, 5, 45, "N");
double[][] image5 = moveImage(inDataArrDbl, width, height, 10, 40, 45, "N");
double[][] image6 = moveImage(inDataArrDbl, width, height, 40, 30, 75, "N");

```



```
double[][] image1 = moveImage(inDataArrDbl, width, height, 50, 60, 0, "N");
double[][] image2 = moveImage(inDataArrDbl, width, height, 2, 70, 0, "N");
double[][] image3 = moveImage(inDataArrDbl, width, height, 0, 0, 90, "N");
double[][] image4 = moveImage(inDataArrDbl, width, height, 20, 45, 178, "B");
double[][] image5 = moveImage(inDataArrDbl, width, height, 0, 60, 150, "B");
double[][] image6 = moveImage(inDataArrDbl, width, height, 55, 2, 77, "B");
```

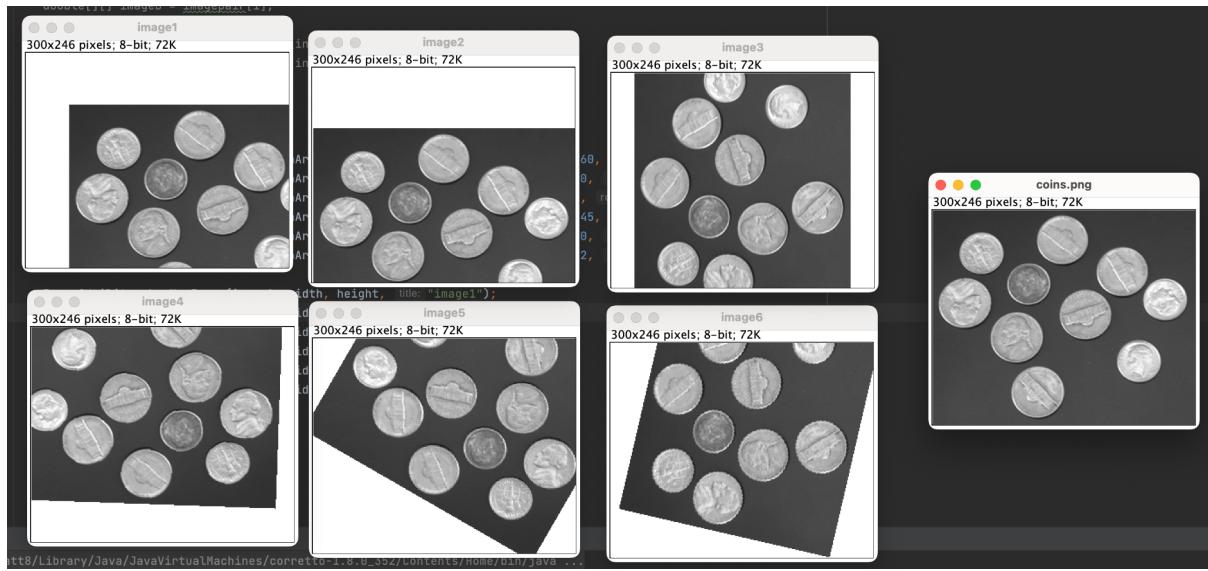


```
double[][] image1 = moveImage(inDataArrDbl, width, height, 10, 40, 0, "N");
```

```

double[][] image2 = moveImage(inDataArrDbl, width, height, 40, 10, 0, "N");
double[][] image3 = moveImage(inDataArrDbl, width, height, 0, 0, 45, "N");
double[][] image4 = moveImage(inDataArrDbl, width, height, 50, 70, 55, "N");
double[][] image5 = moveImage(inDataArrDbl, width, height, 20, 40, 33, "N");
double[][] image6 = moveImage(inDataArrDbl, width, height, 40, 30, 10, "N");

```



```

double[][] image1 = moveImage(inDataArrDbl, width, height, 50, 60, 0, "N");
double[][] image2 = moveImage(inDataArrDbl, width, height, 2, 70, 0, "N");
double[][] image3 = moveImage(inDataArrDbl, width, height, 0, 0, 90, "N");
double[][] image4 = moveImage(inDataArrDbl, width, height, 20, 45, 178, "N");
double[][] image5 = moveImage(inDataArrDbl, width, height, 0, 60, 150, "N");
double[][] image6 = moveImage(inDataArrDbl, width, height, 55, 2, 77, "N");

```

c) [8] Implementieren Sie eine automatische Registrierung. Zur Fehlerbestimmung berechnen Sie die SSE auf Basis der Skalarwerte des Eingangsbildes (8bit grau). Definieren Sie dabei eine Enumeration über eine diskrete Anzahl an Parametrisierungen für deltaX, deltaY und den Rotationswinkel. BSP: 21 Schritte für deltaX und deltaY von [-10.0;10.0] für Translation bzw. 11 Schritte für Rotation im Bereich [-5°;5°]; D.h. es sind $21 * 21 * 11 == 4,851$ Lösungen zu evaluieren, was mit enden-wollender Laufzeit möglich sein sollte. Die Schrittweite (z.B. +1Pixel Translation, +1° Rotation, etc.) ergibt sich aus den jeweiligen Vorgaben. Weisen Sie die initiale und finale Fitness numerisch aus und zeigen Sie ein Differenzbild aus A' und B an, das die Güte Ihrer Registrierung wiedergibt. Geben Sie ferner die gewählten Transformations-Parameter aus.

Lösungsansatz:

Die automatische Registrierung erfolgt durch die Funktion "applyAutomaticRegistration()" welcher die entsprechenden Parameter mitgegeben werden. Mit Hilfe von Variablen numOfStepsTx, numOfSteosTy, stepWidthTx, wird der abzusuchende Bereich definiert. Dabei soll das bestmögliche Zwischenergebnis der einzelnen Variablen gemerkt und später am Bild angewendet werden.

Funktion:

```

public double [][] applyAutomaticRegistration( double[][] imageA, double[][] unregisteredImg, double initError, int width, int height){
    double[][] registeredImg = new double [width][height];
    double[][] diffImg = getDiffImg (imageA, unregisteredImg,
    (int)width/2, height);

    //cf. radius ==> 11 solutions = 2*5+1 (for 0)// -5;5 (-5, -4, -3, ..., 0, 1, 3, ... 5);
    int numOfStepsTx = 10;
    int numOfStepsTy = 10;
    int numOfStepsRot = 10;

    //multiply steps ==> (-10, -8, -6, ..., 0, 2, 6, ... 10);
    double stepWidthTx = 2.0;
    double stepWidthTy = 2.0;
    double stepWidthRot = 2.0;

    //size of search space is: 11 x 11 x 11 ==> 1,331 solutions
    //one possible search candidate is (-4, 2.0, -8.0) for (Tx, Ty, Rot);

    double minERR = initError;
    double bestTx = 0.0;
    double bestTy = 0.0;
    double bestRot = 0.0;

    //permute all possible solutions
    for (int txStep = -numOfStepsTx; txStep <= numOfStepsTx; txStep++) {
        double currTx = txStep * stepWidthTx; //e.g. -4.0 for example
above

        for (int tyStep = -numOfStepsTy; tyStep <= numOfStepsTy;
        tyStep++) {
            double currTy = tyStep * stepWidthTy;

            for (int rotStep = -numOfStepsRot; rotStep <= numOfStepsRot;
            rotStep++) {
                double currRot = rotStep * stepWidthRot;

                ///what is the error now?
                registeredImg = transformImg(unregisteredImg, width,
height, currTx, currTy, currRot);
            }
        }
    }
}

```

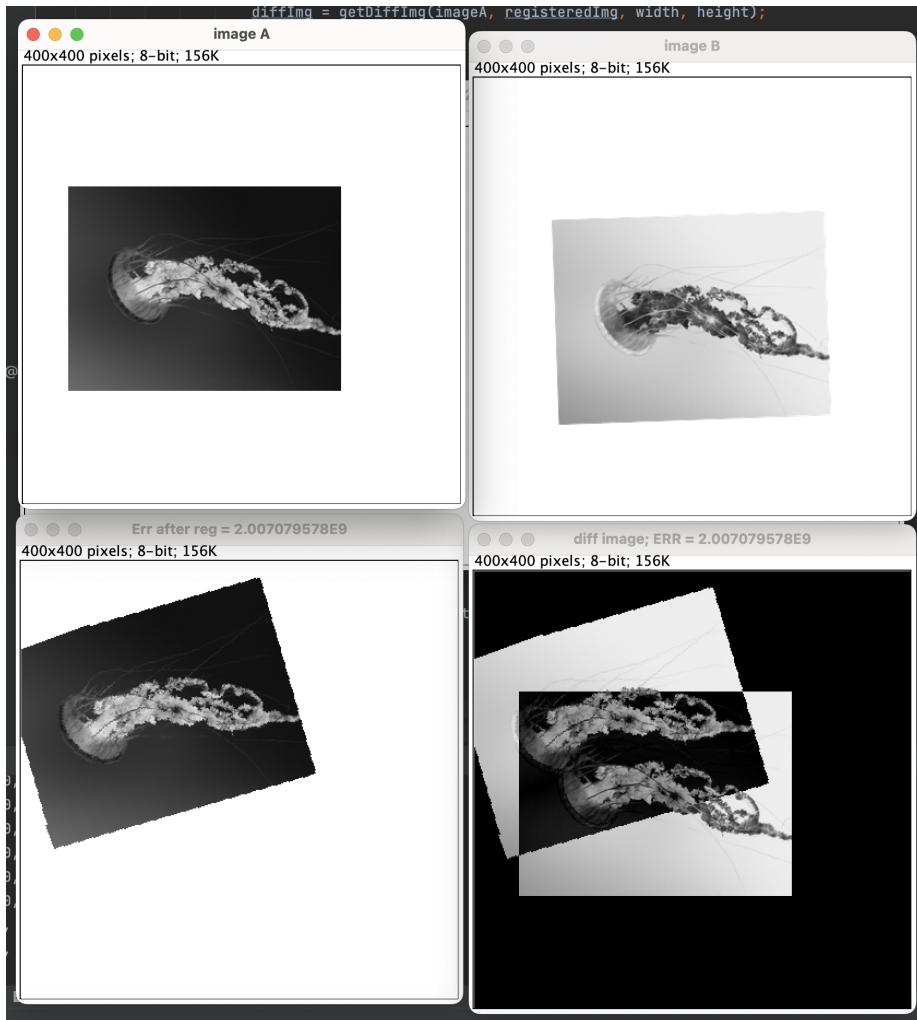
```
//Gut oder schlechter Wert? Neuer Fehler ausrechnen!
//if we found a new best solution we replace the other
double currErr = getImgDiffSSE(imageA, registeredImg,
width, height);
if (currErr < minERR) {
    minERR = currErr;
    bestTx = currTx;
    bestTy = currTy;
    bestRot = currRot;
    System.out.println("New best solution found with " +
bestTx + ", " + bestTy + " , " + bestRot + " at ERR = " + minERR);

    //find best solutions
    registeredImg = transformImg(unregisteredImg, width,
height, bestTx, bestTy, bestRot);
    diffImg = getDiffImg(imageA, registeredImg, width,
height);
}

}
}
ImageJUtility.showNewImage(registeredImg, width, height, "Err after
reg = " + minERR);
ImageJUtility.showNewImage(diffImg, width, height, "diff image; ERR =
" + minERR);

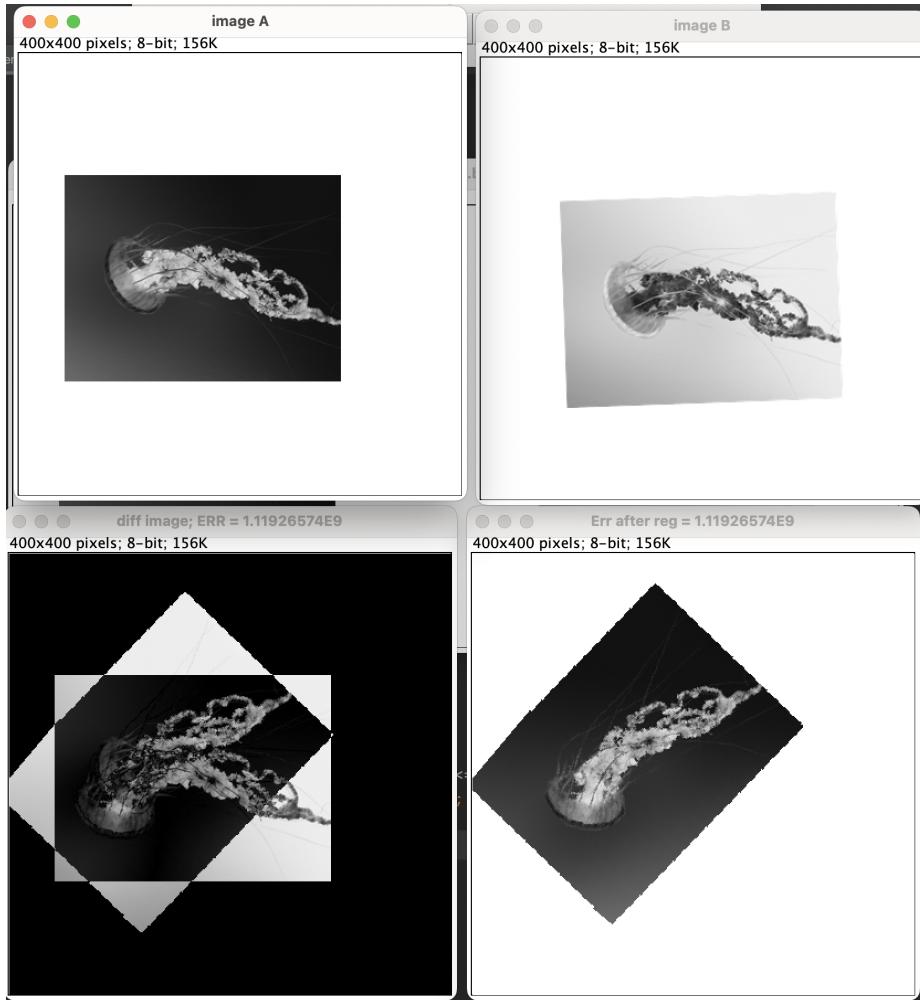
return registeredImg;
}
```

Testfälle:



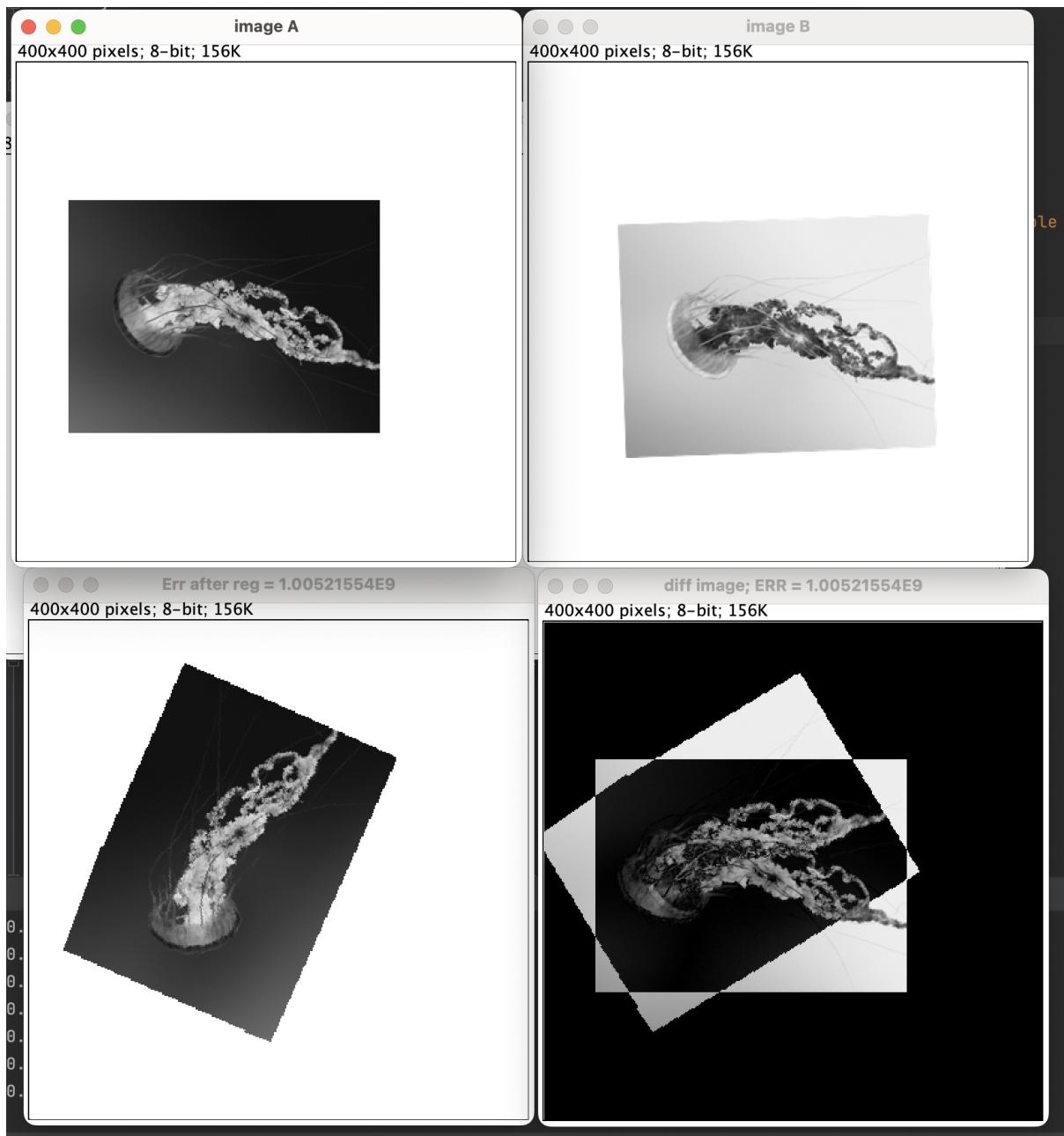
New best solution found with 20.0, 20.0 , 20.0 at ERR = 2.007079578E9

```
double stepWidthTx = 2.0;  
double stepWidthTy = 2.0;  
double stepWidthRot = 2.0;
```



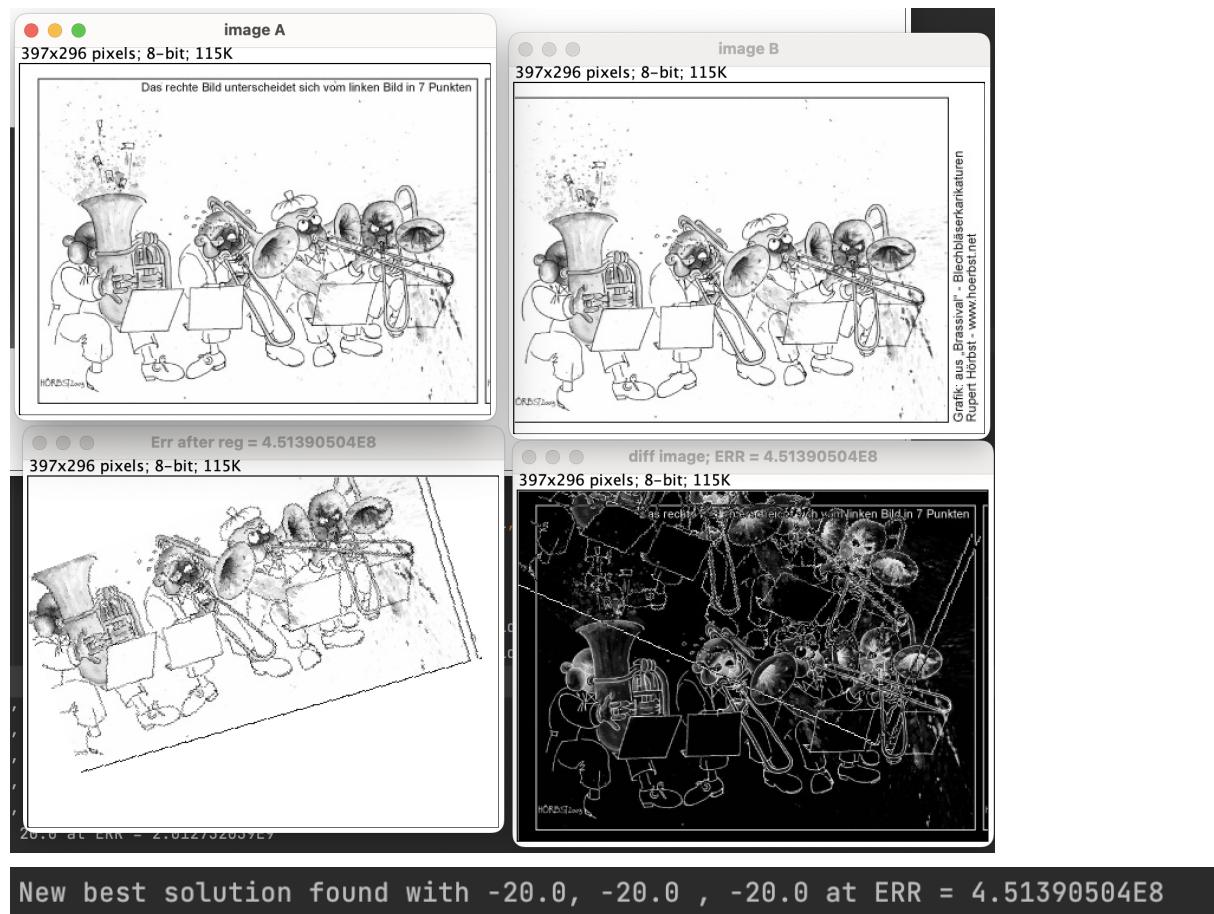
New best solution found with 40.0, 50.0 , 50.0 at ERR = 1.11926574E9

```
double stepWidthTx = 5.0;  
double stepWidthTy = 5.0;  
double stepWidthRot = 5.0;
```



New best solution found with 35.0, 60.0 , 35.0 at ERR = 1.00521554E9

```
double stepWidthTx = 7.0;  
double stepWidthTy = 6.0;  
double stepWidthRot = 7.0;
```



```
double stepWidthTx = 2.0;
double stepWidthTy = 2.0;
double stepWidthRot = 2.0;
```



```
double stepWidthTx = 7.0;
double stepWidthTy = 6.0;
double stepWidthRot = 7.0;
```

New best solution found with 35.0, 60.0 , 7.0 at ERR = 8.667079829E9



New best solution found with -4.0, 20.0 , 20.0 at ERR = 8.951865285E9

```
double stepWidthTx = 2.0;
double stepWidthTy = 2.0;
double stepWidthRot = 2.0;
```

d) [6] Implementieren Sie auch die Möglichkeit zur Registrierung mittels Mutual Information Metrik und testen Sie ausführlich.

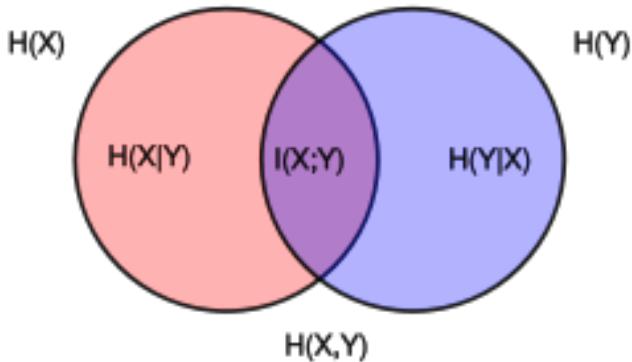
Lösungsidee

Bei Mutual Information wird im Gegensatz zu Sum Of Squared Errors der größtmögliche Wert gesucht, damit eine erfolgreiche Registrierung stattfinden kann.

Die Berechnung zweier Bilder mit der Mutual Information Metrik folgendermaßen:

$$I(X,Y) = H(Y) + H(X) - H(X,Y)$$

$H(X)$ und $H(Y)$ beschreiben die Entropie der Bilder X und Y. $H(X,Y)$ beschreibt die zusammengefasste Entropie der Bilder X und Y. Die zusammengefasste Entropie wird berechnet, indem ein Histogramm gebildet wird dass beide Werte der Bilder berücksichtigt.



Die Entropie eines Bildes $H(X)$ ist wie folgt definiert:
$$H(A) = -\sum_i p_i \ln(p_i)$$

Mutual Information

```
public double MutualInformation(double[][] imgData1, double[][] imgData2, int width, int height) {
    return getEntropyOfImg(imgData1, width, height) +
        getEntropyOfImg(imgData2, width, height) -
        getEntropyOfImages(imgData1, imgData2, width, height);
}
```

Histogramm

Um die Entropie eines Bildes zu ermitteln, wird ein Histogramm benötigt. Des Weiteren wird, eine zweite Methode zum Bilden eines zusammengefassten Histogramms zweier Bilder benötigt.

```
public static double[] histogram(double[][] image, int width, int height){

    double[] histogram = new double[256];
    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            histogram[(int) image[x][y]]++;
        }
    }

    return histogram;
}

public static double[][] histogram(double[][] image1, double[][] image2, int width, int height){
    double[][] histogram = new double[256][256];

    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            histogram[(int) image1[x][y]][(int) image2[x][y]]++;
        }
    }

    return histogram;
}
```

Entropie

Die Entropie von einem (`getEntropyOfImg`) bzw zwei Bildern (`getEntropyOfImages`) berechnet

```
public double getEntropyOfImg(double[][] imgData, int width, int height) {

    double[] histogram = histogram(imgData, width, height);
    int pixels = width * height;

    double result = 0.0;
    for (int i = 0; i < histogram.length; i++) {
        double p = histogram[i] / (double) pixels;
        if (p > 0) result += p * (Math.log10(p) / (double) Math.log10(2));

    }
    return -result;
} //getEntropyOfImg
```

```
public double getEntropyOfImages(double[][] imgA, double[][] imgB, int width, int height) {

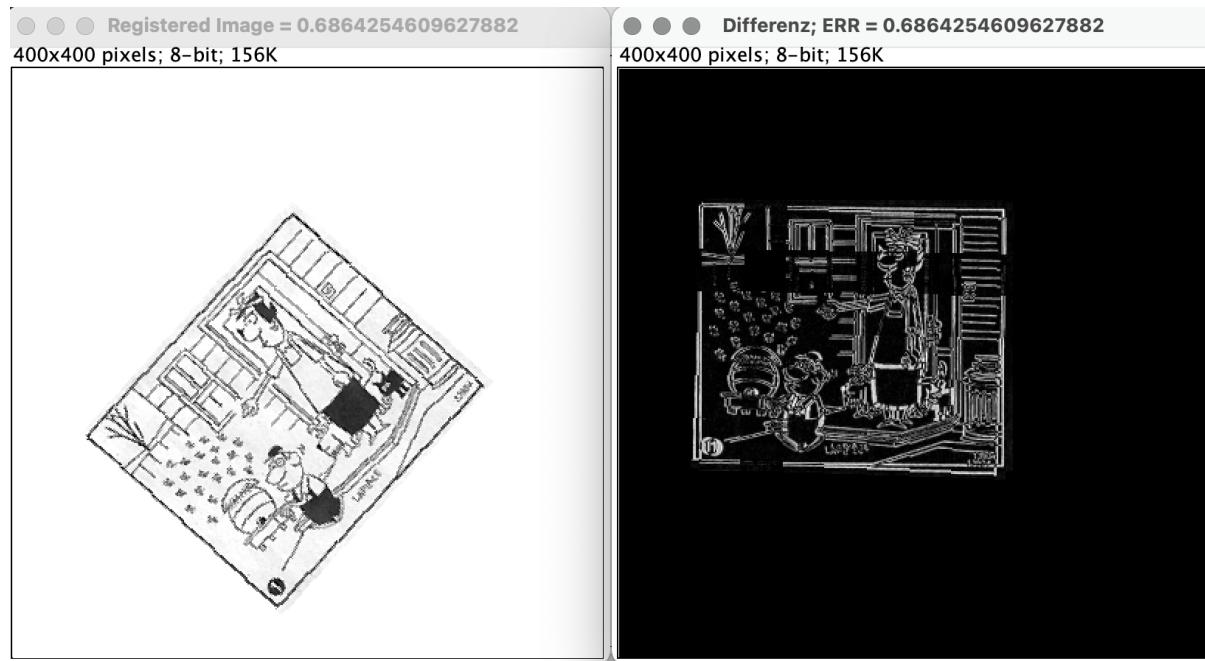
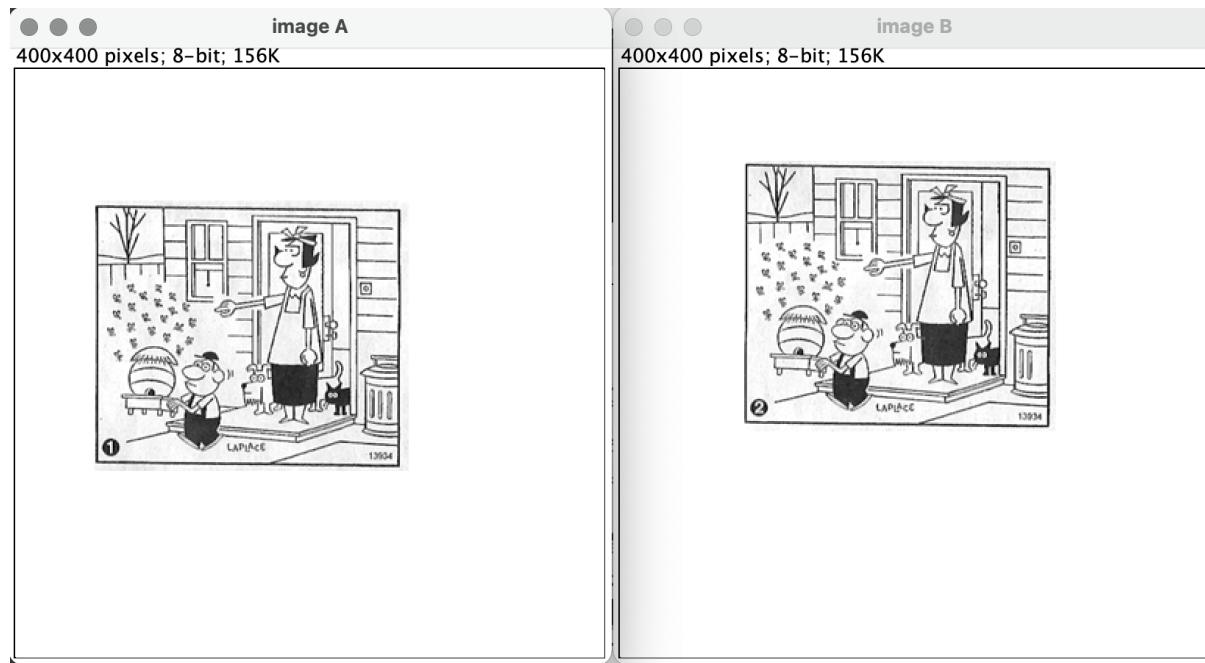
    double[][] histogramOfTwo = histogram(imgA, imgB, width, height);
```

```
int pixels = width * height;

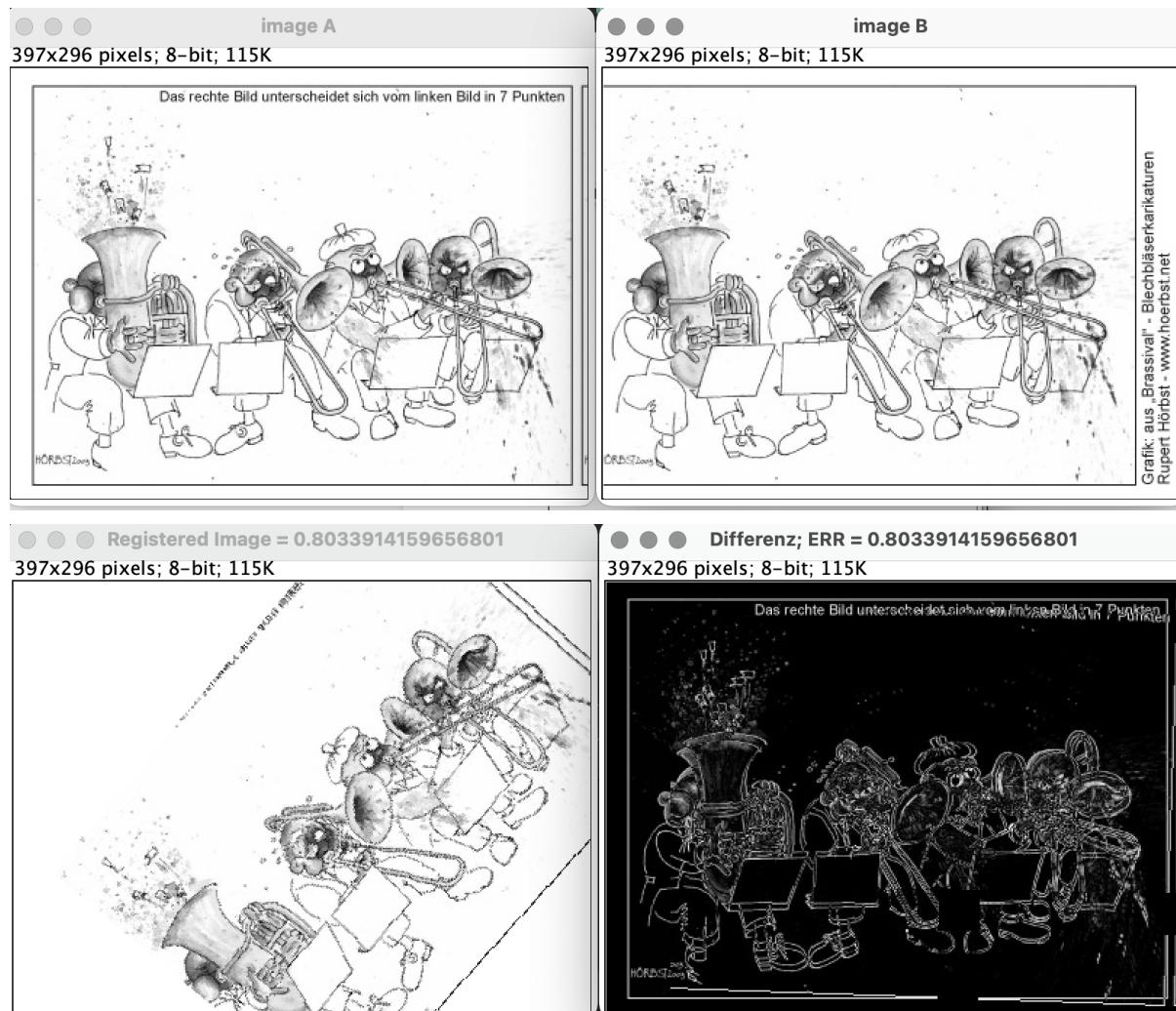
double result = 0.0;

for (int i = 0; i < histogramOfTwo.length; i++) {
    for (int j = 0; j < histogramOfTwo.length; j++) {
        double p = (histogramOfTwo[i][j]) / (double) pixels;
        if (p > 0) result += p * (Math.log10(p)) / (double) Math.log10(2));
    }
}
return -result;
} //getEntropyOfImages
```

Die Registrierung geschieht wie zuvor bei der Sum of Squared Errors. Diesmal muss jedoch der größtmögliche Wert gesucht werden um mittels Mutual Information eine erfolgreiche Registrierung durchführen zu können.



New best solution found with **40.0, 25.0, 0.0** at ERR = **0.6864254609627882**



New best solution found with **45.0, 20.0, 0.0** at ERR = **0.8033914159656801**



New best solution found with **40.0, 20.0, 0.0** at ERR = **0.7025445944326432**