# BVA2 – **Exercise 04: *Hough Transformation***

**[18 points]**

## Hough Transformation for Detection of Circles (*iris of human eye*)

The goal of this exercise is to utilize Hough Transformation of circles to detect the iris of human eye. The iris part of human eye has an almost circular shape and thus can be perfectly detected utilizing Hough Transform approaches, even in cases where parts of the eye are occluded, e.g. by the eyelid.

To accomplish this exercise, the following aspects need to be handled:

- (a) Conversion from RGB to grayscale
- (b) in case of noise → apply a smoothing filter
- (c) edge detection
- (d) utilizing a rectangular ROI to derive parameter range for search
- (e) calculation of the Hough Parameter Space
- (f) detect highest fitness in Parameter Space
- (g) Result presentation: (I) best fitting circle as 2D overlay in the input image, (II) 2D Hough-Image for best radius and (III) a MIP (*maximum intensity projection*) calculated in direction of the radius.

### Details regarding the mentioned tasks:

For implementation, the interface definitions provided in `HoughTransformIrisDetection_` must be utilized:

- `public static double[][]` **`cropImage`**`(double[][]inImg, int width, int height, Rectangle roi)` – crop the input image according to provided ROI (region of interest)
- `HoughSpace` **`genHoughSpace`**`(double[][] edgeImage, int width, int height)` – calculate Hough Parameter Space for edge Image cropped according to user-provided ROI. Search granularity to be defined by the user
- `Vector<Point>` **`getPointsOnCircle`**`(int x, int y, int radius)`
  returns coordinates on circle – step size / precision of at least 0.5 pixels useful
- `void` **`plotBestRadiusSpace`**`(HoughSpace houghSpace)`
  plot the parameter space in x/y direction for the best radius parameter
- `public void` **`plotRadiusMIPSpace`**`(HoughSpace houghSpace)`
  plot the MIP (maximum intensity projection) calculated in direction of the radius

(a) utilize strategies from exercise 01 for conversion RGB → Grayscale

(b) regarding smoothing: evaluate the filter kernel and radius that fits best. Can and should the radius be kept constant at varying image sizes?

(c) as first approach utilize gradient magnitude derived by horizontal and vertical Sobel kernel run as fundament for calculation of circle-fitness. Later on evaluate if better results are achievable if edges are interpreted in a binary way (*edge [0;1] instead of gradient magnitude [0;510]*), similar to Canny edge detection strategy. Binary edges of constant width 1 can be approximated as follows:

- apply Sobel edge detection and calculate the gradient magnitude $\left|\vec{g}\right| = \sqrt{g_x^2 + g_y^2}$
- calculate all edge pixels with gradient magnitude > threshold T (chosen to conserve both, the mild and the significant edges)

- calculate the gradient direction $\alpha$ for all of those edge candidates and check the neighbour pixel in direction of the gradient (*8 neighbours in N8 to be targeted by [0°;360°]*). An edge candidate only remains an edge in the result image if the neighbour in the target direction shows a gradient magnitude lower or equal compared to the current value. $$\alpha = \arccos\left(\frac{g_x}{|\vec{g}|}\right)$$

(d) A rectangular ROI is provided by the user to restrict the search area. It is expected that the iris is fully located within the ROI. The ROI is utilized to crop the image and to derive the range per dimension in parameter space (*e.g. no need to check a radius > width/2 or > height/2*). The search space should be defined with reasonable limits to reduce runtime.

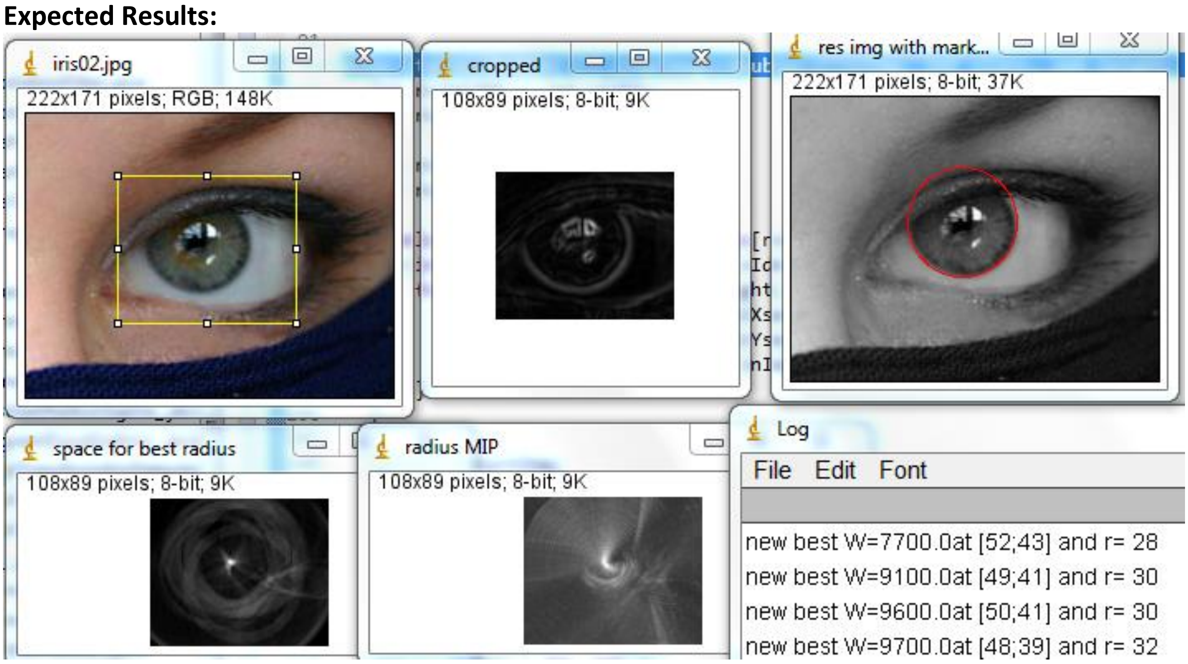
(e) Calculate the 3D parameter space. Fitness is thereby derived from circular shape evaluated according to gradient magnitude or the binary edge results (see (c)). **ATTENTION**: *Is the fitness influenced by the particular radius and why?* If that's the case evaluate and implement a strategy for normalization.

(f) Detect the maximum, i.e. best (x,y,radius) coordinate.

(g) Show the results ➔ the best circle in the original input image (*preferably as red-circle*). Furthermore plot the MIP (maximum intensity projection) in direction of the radius-dimension and the x/y 2D image for best radius. *Discuss these two plots – what kind of conclusions on result quality can be stated based on these evaluations?*

**Relevant literature**: article *"Probing the Uniqueness and Randomness of IrisCodes: Results From 200 Billion Iris Pair Comparisons"* by mathematician ***John Daugman***, utilizing results of iris detection for human-specific encoding provided on the elearning platform. Allows for identification of a person similar to fingerprints and other biometric features.

**Expected Results:**




ROI in yellow ➔ utilized for cropping (edge detection)
Best result, i.e. highest fitness is found iterating over entire 3D Parameter Space (see the Log). Best ***P(x,y,radius)*** is utilized to mark iris with red circle. 2D image for best radius and 2D MIP of radius are plotted in the lower row.

# 1 Hough Transformation

## 1.1 Lösungsidee

### 1.1.1 (a) Conversion from RGB to grayscale

Diese Aufgabe wird mit der bereits in der Klasse *ImageJUtility* implementierten Methode *getGrayscaleImgFromRBG* gelöst. Jedoch wird die Implementierung um einen weiteren Case erweitert, in dem die gewichtete Berechnung, anstatt des Mittelwerts genommen wird.

$$grayValue = 0.299 * red + 0.587 * green + 0.114 * blue$$

### 1.1.2 (b) in case of noise → apply a smoothing filter

Diese Aufgabe wird mit der *Anisotropen Diffusion* von Ruhsam Christoph der dritten Übung gelöst und wird hier nicht näher erläutert. $\kappa = 20; iterations = 10$

### 1.1.3 (c) edge detection

Diese Aufgabe wird mit dem *Sobel-Operator*, der gemeinsam in der Übung entwickelt wurde, gelöst. Dazu wird zuerst der SobelH auf das Bild nach der Anisotropen Diffusion angewandt. Danach der SobelV auf das Bild nach Anisotropen Diffusion. Diese beiden Bilder werden mit dem Satz des Pythagoras zu einem gemerged.

### 1.1.4 (d) utilizing a rectangular ROI to derive parameter range for search

Dieser Teil war bereits in dem Template vorhanden. Deshalb wird auch hier keine Lösungsidee angegeben.

### 1.1.5 (e) calculation of the Hough Parameter Space

Hier wird zuerst überprüft, ob es sich bei dem Bild um Hoch- oder Querformat handelt. Der **Radius** ergibt sich durch $min(width, height)/2$. Danach wird für jedes Pixel jeder Radius von $minRadius$ bis $radius$ durchlaufen. Hier wird für jeden Radius und jedes Grad von 0 bis 360 der „Hit“ berechnet. Der x-Wert und y-Wert des Hit-Pixels ergibt sich durch:

$$xHit = Math.floor(x - rad * (Math.cos(t * Math.PI/180)))$$
$$yHit = Math.floor(y - rad * (Math.sin(t * Math.PI/180)))$$

, wobei $x$ und $y$ die Koordinaten des aktuellen Pixels sind, $rad$ der aktuelle Radius und $t$ das aktuelle Grad ist. Sobald es einen Hit gibt, wird im HoughSpace an der Stelle $hs.houghSpace[x][y][rad]$ der aktuelle Wert um 1 inkrementiert.

### 1.1.6 (f) detect highest fitness in Parameter Space

Um den besten fitness-Wert zu finden, wird der gesamte HoughSpace durchlaufen und der max-Wert an der Stelle $hs.houghSpace[x][y][rad]$ gesucht.

### 1.1.7 (g) Result presentation

Diese Aufgabe wird unter dem Punkt 1.3 Tests und Auswertung behandelt.

## 1.2 Code

Listing 1: HoughTransformIrisDetection_.java

```java
import ij.IJ;
import ij.ImagePlus;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageProcessor;

import java.awt.*;
import java.awt.image.BufferedImage;
import java.util.Vector;


public class HoughTransformIrisDetection_ implements PlugInFilter {


    public int setup(String arg, ImagePlus imp) {
        if (arg.equals("about")) {
            showAbout();
            return DONE;
        }
        return DOES_RGB + DOES_STACKS + SUPPORTS_MASKING + ROI_REQUIRED;
    } //setup


    public void run(ImageProcessor ip) {
        BufferedImage buffImage = ip.getBufferedImage();

        int width = ip.getWidth();
        int height = ip.getHeight();

        //convert to grayscale
        //use weighted conversion instead of mean
        //0.299 * red + 0.587 * green + 0.114 * blue
        double[][] grayscaleImg = ImageJUtility.getGrayscaleImgFromRGB(ip,
        ↪  ImageJUtility.CONVERSION_MODE_RGB_GRAYSCALE_WEIGHTED);

        //apply convolution filter for smoothing
        ImageJUtility.showNewImage(grayscaleImg, width, height, "gray");
        int[][] grayscaleImgInt = ImageJUtility.anisotropicDiffusion(grayscaleImg, width, height);
        grayscaleImg = ImageJUtility.convertToDoubleArr2D(grayscaleImgInt, width, height);
        ImageJUtility.showNewImage(grayscaleImg, width, height, "gray after ad");

        //perform edge detection
        double[][] grayscaleImgSobelH = ImageJUtility.sobel(ip, grayscaleImg,
        ↪  ImageJUtility.SOBELH);
        double[][] grayscaleImgSobelV = ImageJUtility.sobel(ip, grayscaleImg,
        ↪  ImageJUtility.SOBELV);

        // merge the sobelV and sobelH images
        for (int i = 0; i < width; i++) {
            for (int j = 0; j < height; j++) {
                grayscaleImg[i][j] = Math.round(Math.sqrt(
                        Math.pow(grayscaleImgSobelV[i][j], 2) +
                                Math.pow(grayscaleImgSobelH[i][j], 2)
                ));
            }
        }

        // show grayscale image with anisotropic diffusion after sobel
        ImageJUtility.showNewImage(grayscaleImg, width, height, "with AD");

```

```java
        //now restrict to sub-image
        Rectangle roiSelection = ip.getRoi();
        int roiWidth = roiSelection.width;
        int roiHeight = roiSelection.height;

        // crop the image to roi and show the cropped image
        double[][] croppedImg = ImageJUtility.cropImage(grayscaleImg, roiWidth, roiHeight,
        ↪  roiSelection);
        ImageJUtility.showNewImage(croppedImg, roiWidth, roiHeight, "cropped");

        //now generate the hough space
        HoughSpace houghSpace = genHoughSpace(croppedImg, roiWidth, roiHeight);

        //now chart the result ==> pixels in red
        double a = houghSpace.bestX - houghSpace.bestR * Math.cos(0 * Math.PI / 180);
        double b = houghSpace.bestY - houghSpace.bestR * Math.sin(90 * Math.PI / 180);
        ip.setColor(Color.RED);
        ip.drawOval(roiSelection.x + (int) a, roiSelection.y + (int) b, 2 * houghSpace.bestR, 2 *
        ↪  houghSpace.bestR);

        //finally plot 2D image for best radius and MIP image in direction of the radius
        plotBestRadiusSpace(houghSpace);
        plotRadiusMIPSpace(houghSpace);
    } //run

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

    public void plotBestRadiusSpace(HoughSpace hs) {
        double[][] bestRadii = new double[hs.width][hs.height];

        for (int x = 0; x < hs.width; x++) {
            for (int y = 0; y < hs.height; y++) {
                bestRadii[x][y] = scaleValueBetween(hs.houghSpace[x][y][hs.bestR],
                        0, 255, 0, (int) hs.houghSpace[hs.bestX][hs.bestY][hs.bestR]);
            }
        }

        ImageJUtility.showNewImage(bestRadii, hs.width, hs.height, "BestRadiusSpace");
    }

    public void plotRadiusMIPSpace(HoughSpace hs) {
        double[][] bestRadii = new double[hs.width][hs.height];

        for (int x = 0; x < hs.width; x++) {
            for (int y = 0; y < hs.height; y++) {
                double bestR = 0;
                for (int r = 0; r < hs.houghSpace[x][y].length; r++) {
                    if (hs.houghSpace[x][y][r] > bestR) {
                        bestR = hs.houghSpace[x][y][r];
                    }
                }
                int best = (int) hs.houghSpace[hs.bestX][hs.bestY][hs.bestR];
                bestRadii[x][y] = scaleValueBetween(bestR, 0, 255, 0, best);
            }
        }
        ImageJUtility.showNewImage(bestRadii, hs.width, hs.height, "RadiusMIPSpace");
    }


    public HoughSpace genHoughSpace(double[][] edgeImage, int width, int height) {
```

```java
        // first calculate the parameter range
        // then evaluate fitness for each parameter permutation
        int radius;
        if (height < width) {
            radius = height / 2;
        } else {
            radius = width / 2;
        } // sets a 3D space array of ints to hold 'hits' in x, y, and r planes
        int minRadius = 10;
        HoughSpace hs = new HoughSpace(width, height, radius, minRadius);
        for (int rad = minRadius; rad < radius; rad++) {
            for (int x = 0; x < width; x++) {
                for (int y = 0; y < height; y++) {
                    for (int t = 0; t <= 360; t++) {
                        Integer a = (int) Math.floor(x - rad * Math.cos(t * Math.PI / 180));
                        Integer b = (int) Math.floor(y - rad * Math.sin(t * Math.PI / 180));
                        if (edgeImage[x][y] > 25) {
                            if (!((0 > a || a > width - 1) || (0 > b || b > height - 1))) {
                                if (!(a.equals(x) && b.equals(y))) {
                                    hs.houghSpace[a][b][rad] += 1;
                                }
                            }
                        }
                    }
                }
            }
        } // then evaluate fitness for each parameter permutation
        double max = 0;
        for (int x = 0; x < width; x++) {
            for (int y = 0; y < height; y++) {
                for (int r = 5; r < radius; r++) {
                    if (hs.houghSpace[x][y][r] > max) {
                        max = hs.houghSpace[x][y][r];
                        hs.bestX = x;
                        hs.bestY = y;
                        hs.bestR = r;
                    }
                }
            }
        }

        return hs;
    }

    public class HoughSpace {
        double[][][] houghSpace;
        int width;
        int height;

        int bestX;
        int bestY;
        int bestR;

        int minRadius;
        int radiusRange;

        double bestWeight = 0.0;

        public HoughSpace(int width, int height, int radiusRange, int minRadius) {
            this.width = width;
            this.height = height;
            this.bestR = -1;
            this.bestX = -1;
```

```java
            this.bestY = -1;
            this.bestWeight = 0.0;
            this.minRadius = minRadius;
            this.radiusRange = radiusRange;

            //initialize the array
            houghSpace = new double[width][height][radiusRange];
        }

    }

    private double scaleValueBetween(double value, int from, int to, int min, int max) {
        return (to - from) * ((value - min) / (max - min)) + from;
    }


} //class HoughTransformIrisDetectionTemplate_

```

## 1.3 Tests und Auswertung

### 1.3.1 Iris 4

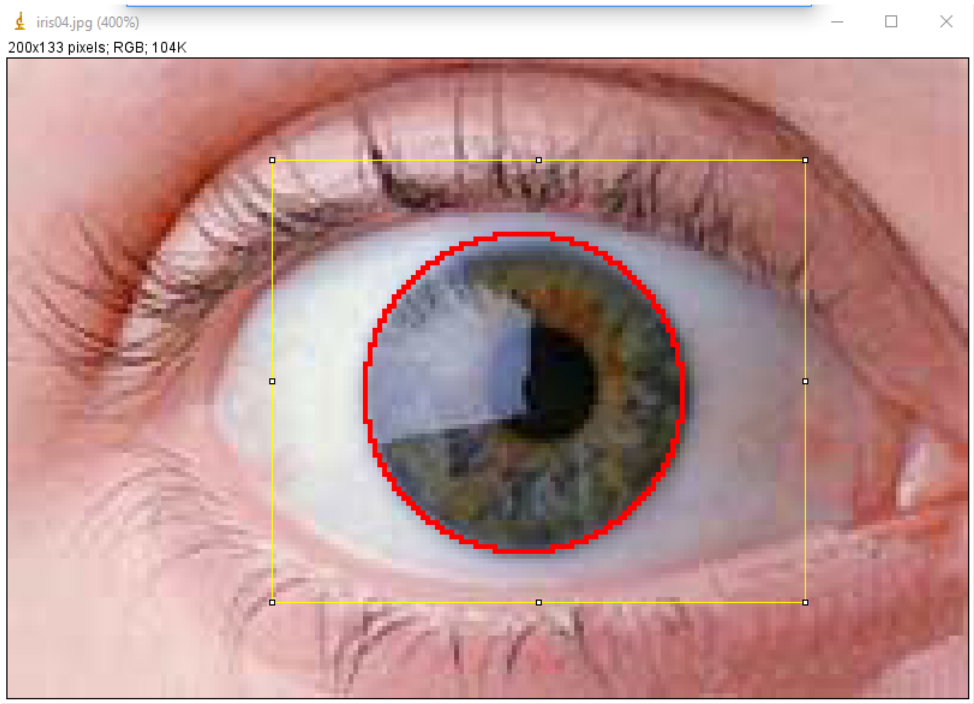



Abbildung 1: Original Image (detected iris)

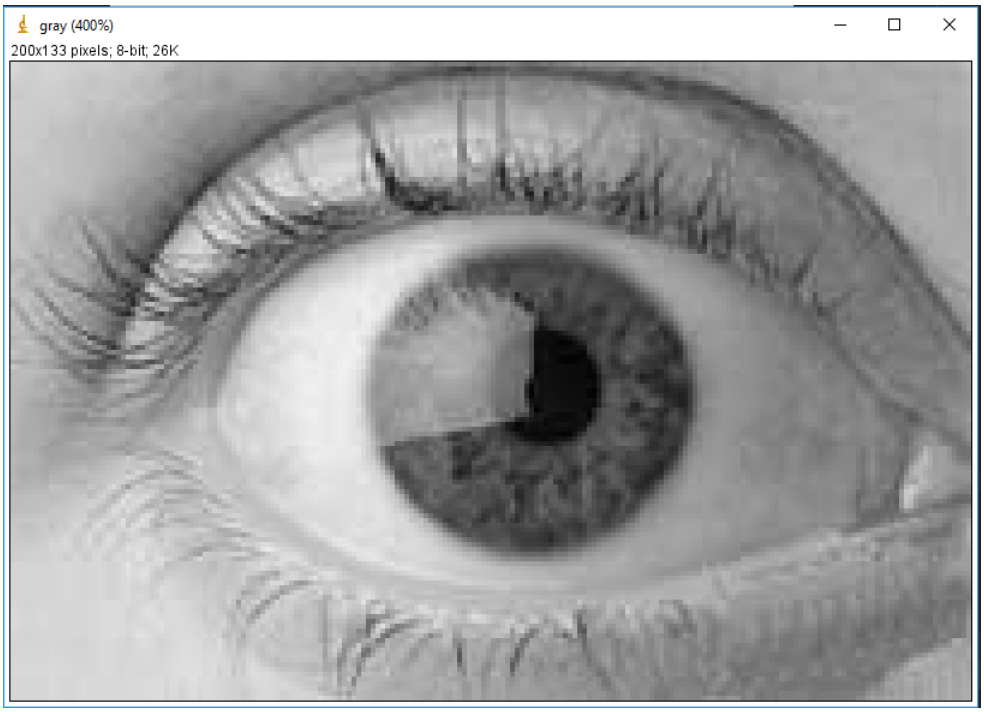



Abbildung 2: Gray Image after Conversion

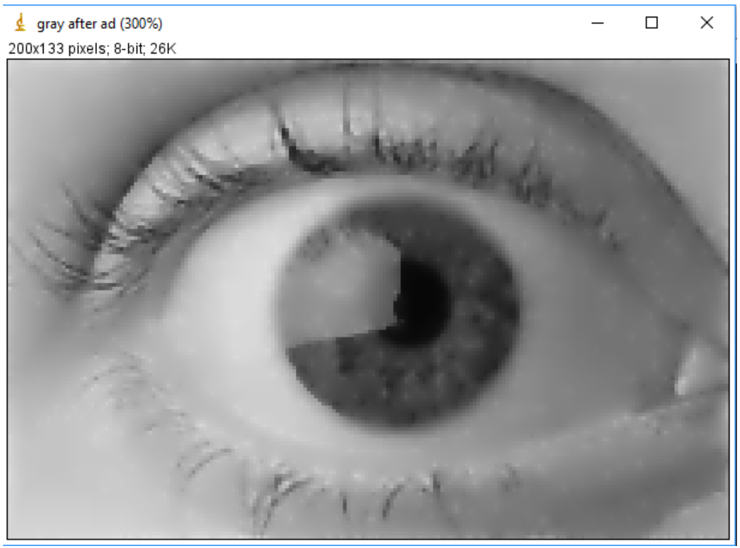



Abbildung 3: Gray Image after Conversion and Anisotropic Diffusion

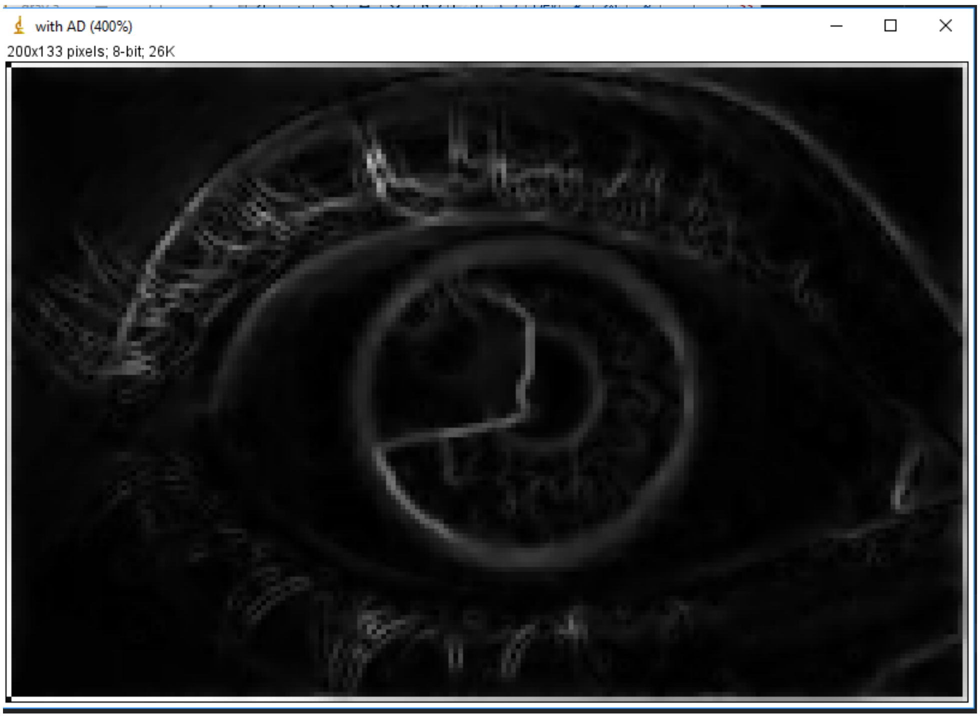



Abbildung 4: Image after Sobel


Abbildung 5: Cropped Image


Abbildung 6: Best Radius Space

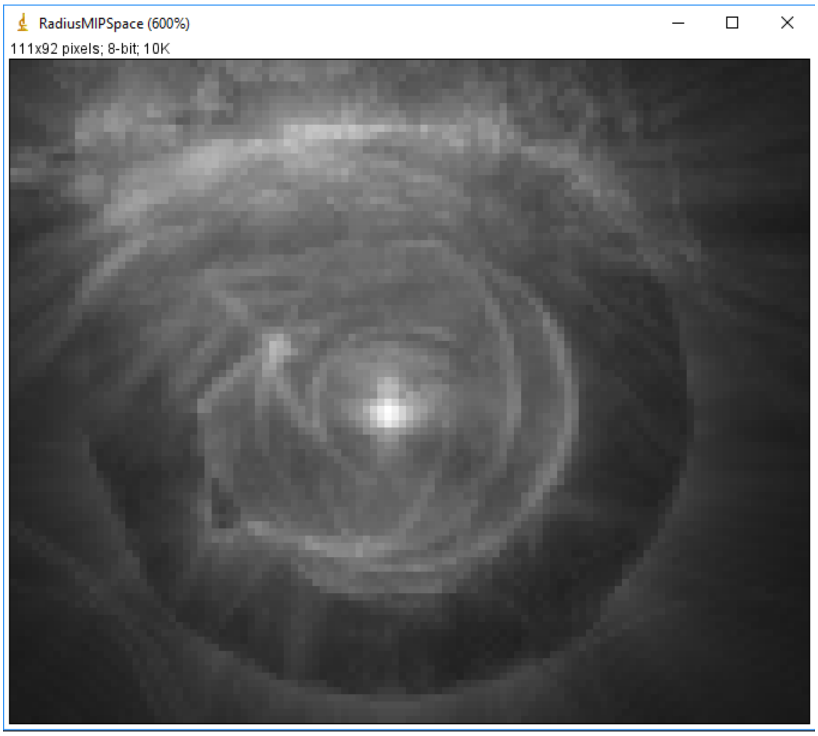



Abbildung 7: Radius MIP Space

#### 1.3.2 Iris 5

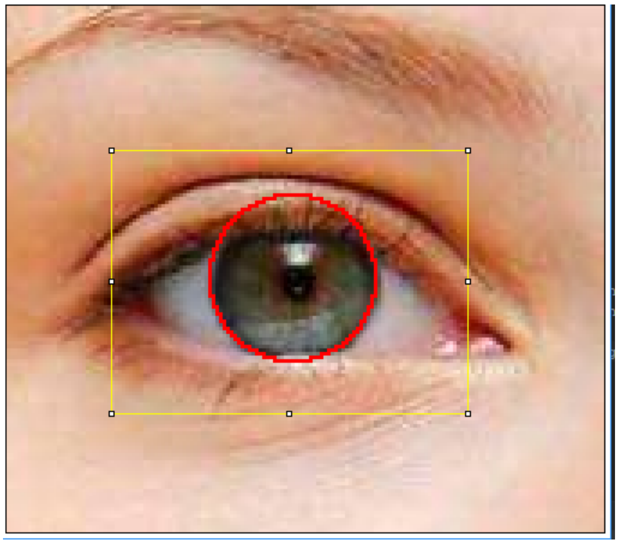


Abbildung 8: Original Image (detected iris)

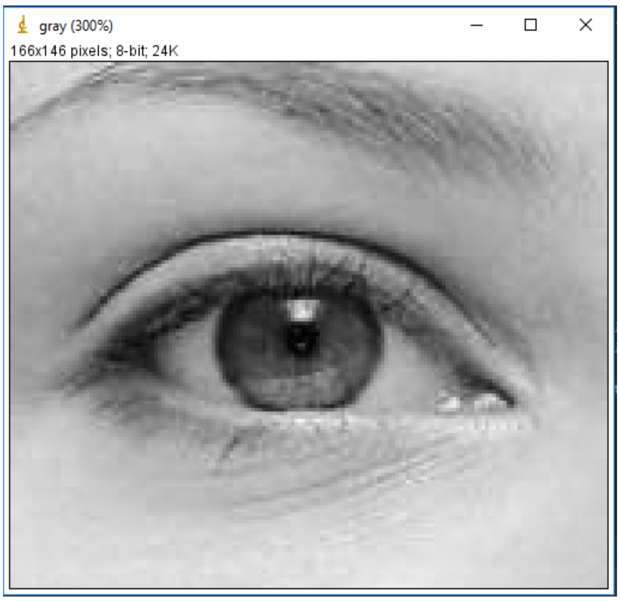



Abbildung 9: Gray Image after Conversion

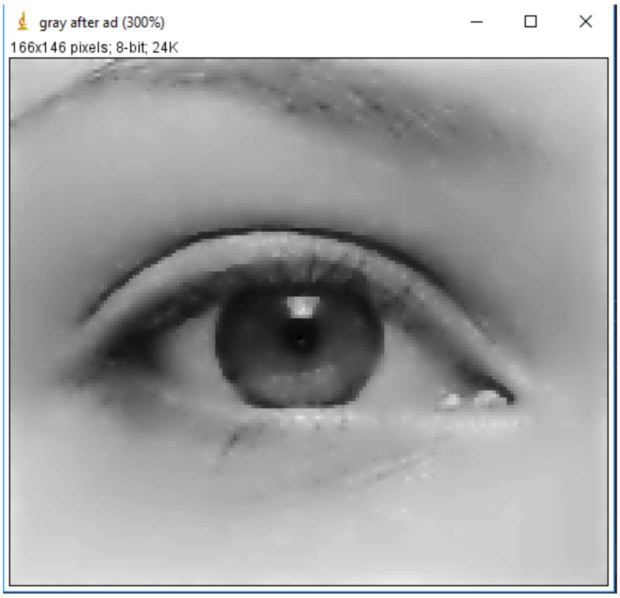



Abbildung 10: Gray Image after Conversion and Anisotropic Diffusion

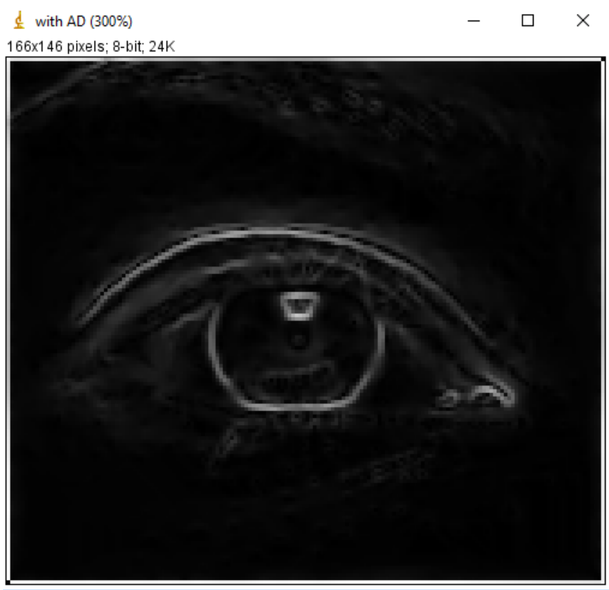



Abbildung 11: Image after Sobel


Abbildung 12: Cropped Image

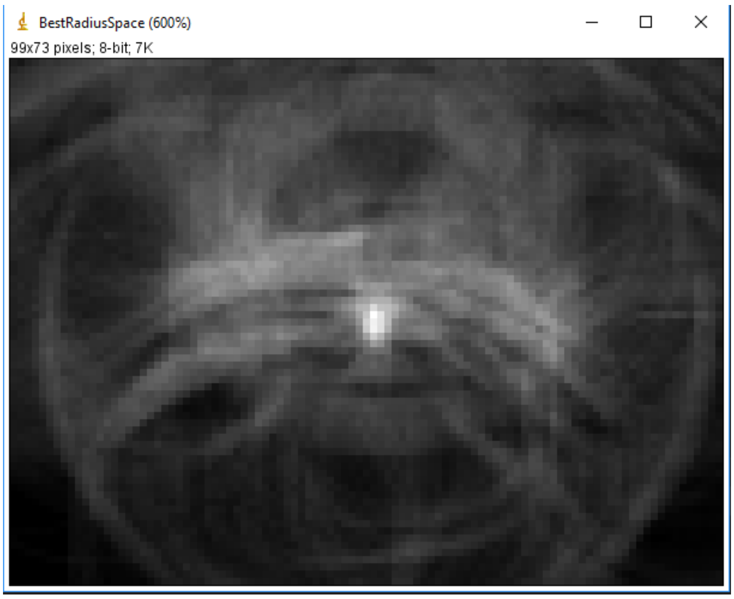



Abbildung 13: Best Radius Space

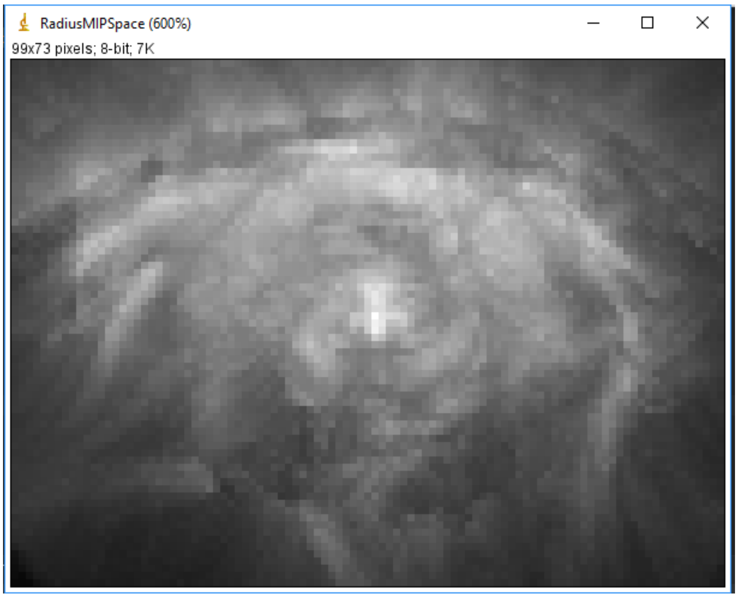



Abbildung 14: Radius MIP Space

#### 1.3.3 Iris 6

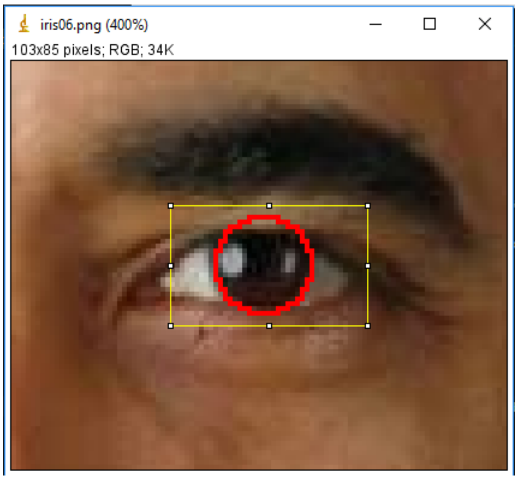



Abbildung 15: Original Image (detected iris)

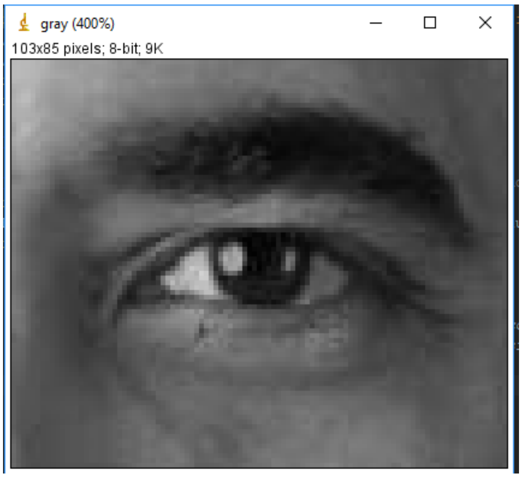



Abbildung 16: Gray Image after Conversion

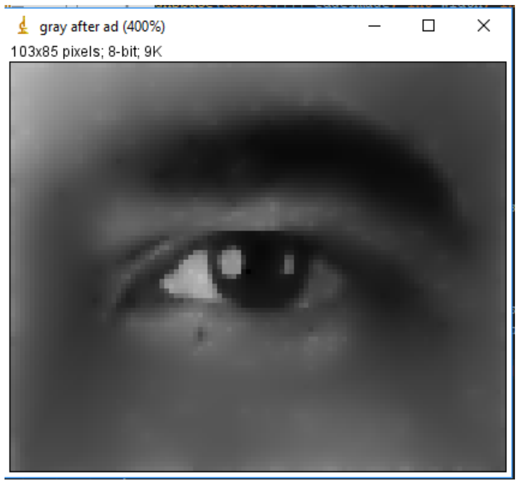



Abbildung 17: Gray Image after Conversion and Anisotropic Diffusion

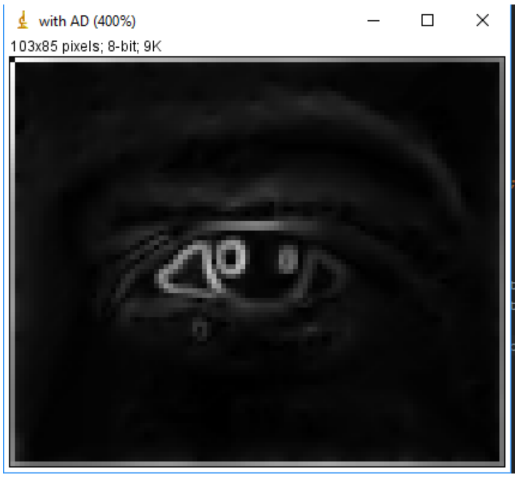



Abbildung 18: Image after Sobel


Abbildung 19: Cropped Image

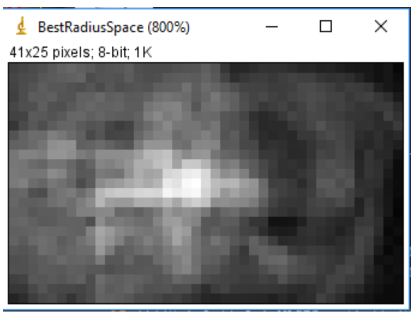



Abbildung 20: Best Radius Space

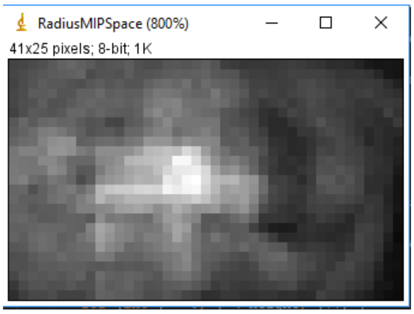



Abbildung 21: Radius MIP Space