# Name: Marcio Adriano de Campos Junior

**No. USP: 11871625**

**report.doc Digital filters**

**Question 1.3:**

Horizontal Edge Detector in africa.tif

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Time | Average | Minimum | Maximum |
| non-separable version | 307ms | 127.66 | 0 | 255 |
| separable version | 104 ms | 127.66 | 0 | 255 |

**Comparison non-separable:**

Enter the part of the code written for vertical edge detector:

1.     static public ImageAccess detectEdgeVertical\_NonSeparable(ImageAccess input) {

2.         int nx = input.getWidth();

3.         int ny = input.getHeight();

4.         double arr[][] = new double[3][3];

5.         double pixel;

6.         ImageAccess out = new ImageAccess(nx, ny);

7.         for (int x = 0; x < nx; x++) {

8.             for (int y = 0; y < ny; y++) {

9.                 input.getNeighborhood(x, y, arr);

10.                 pixel = arr[2][0]+arr[2][1]+arr[2][2]-arr[0][0]-arr[0][1]-arr[0][2];

11.                 pixel = pixel / 6.0;

12.                 out.putPixel(x, y, pixel);

13.             }

14.         }

15.         out = rescale(out);

16.         return out;

17.     }

18.

Enter the part of the code written for horizontal edge detector:

1.     static public ImageAccess detectEdgeHorizontal\_NonSeparable(ImageAccess input) {

2.         int nx = input.getWidth();

3.         int ny = input.getHeight();

4.         double arr[][] = new double[3][3];

5.         double pixel;

6.         ImageAccess out = new ImageAccess(nx, ny);

7.         for (int x = 0; x < nx; x++) {

8.             for (int y = 0; y < ny; y++) {

9.                 input.getNeighborhood(x, y, arr);

10.                 pixel = arr[2][2]+arr[1][2]+arr[0][2]-arr[0][0]-arr[1][0]-arr[2][0];

11.                 pixel = pixel / 6.0;

12.                 out.putPixel(x, y, pixel);

13.             }

14.         }

15.         out = rescale(out);

16.         return out;

17.     }

18.

**Comparison Separable:**

Enter the part of the code written for vertical edge detector:

1.     static public ImageAccess detectEdgeVertical\_Separable(ImageAccess input) {

2.         int nx = input.getWidth();

3.         int ny = input.getHeight();

4.         ImageAccess out = new ImageAccess(nx, ny);

5.         double rowin[]  = new double[nx];

6.         double rowout[] = new double[nx];

7.         for (int y = 0; y < ny; y++) {

8.             input.getRow(y, rowin);

9.             doDifference3(rowin, rowout);

10.             out.putRow(y, rowout);

11.         }

12.

13.         double colin[]  = new double[ny];

14.         double colout[] = new double[ny];

15.         for (int x = 0; x < nx; x++) {

16.             out.getColumn(x, colin);

17.             doAverage3(colin, colout);

18.             out.putColumn(x, colout);

19.         }

20.         out = rescale(out);

21.         return out;

22.     }

23.

Enter the part of the code written for horizontal edge detector:

1.     static public ImageAccess detectEdgeHorizontal\_Separable(ImageAccess input) {

2.         int nx = input.getWidth();

3.         int ny = input.getHeight();

4.         ImageAccess out = new ImageAccess(nx, ny);

5.

6.         double colin[]  = new double[ny];

7.         double colout[] = new double[ny];

8.         for (int x = 0; x < nx; x++) {

9.             input.getColumn(x, colin);

10.             doDifference3(colin, colout);

11.             out.putColumn(x, colout);

12.         }

13.

14.         double rowin[]  = new double[nx];

15.         double rowout[] = new double[nx];

16.         for (int y = 0; y < ny; y++) {

17.             out.getRow(y, rowin);

18.             doAverage3(rowin, rowout);

19.             out.putRow(y, rowout);

20.         }

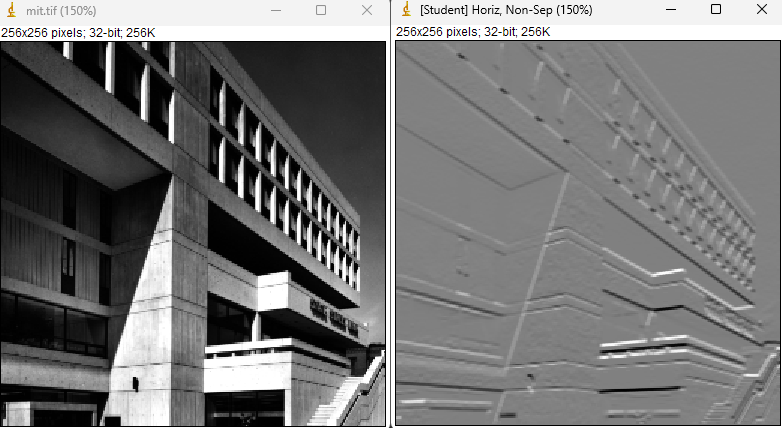
21.         out = rescale(out);

22.         return out;

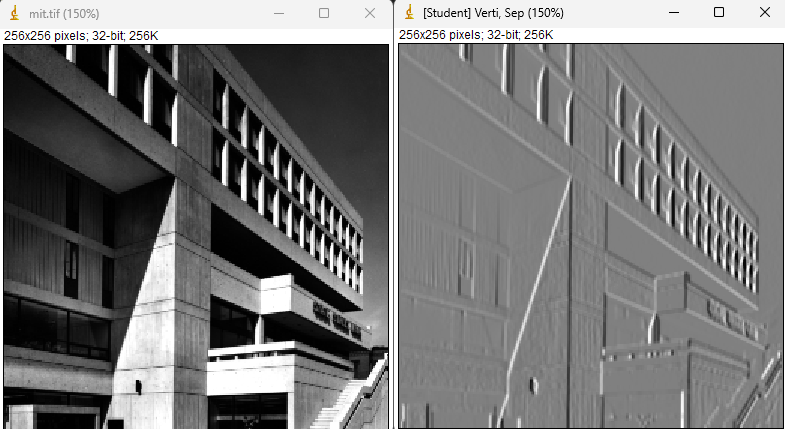
23.     }

24.

**Image results – Horizontal edge**



**Image results – Vertical edge**



**Question 2:**

Moving average 5\*5 in africa.tif

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Time | Average | Minimum | Maximum |
| non-separable version | 398 ms | 71.44 | 0 | 255 |
| separable version | 95 ms | 71.44 | 0 | 255 |
| recursive version | 111 ms | 71.44 | 0 | 255 |

**Comparison:**

1. non-separable version

1.     static public ImageAccess doMovingAverage5\_NonSeparable(ImageAccess input) {

2.         int nx = input.getWidth();

3.         int ny = input.getHeight();

4.         double arr[][] = new double[5][5];

5.         double pixel = 0;

6.         ImageAccess out = new ImageAccess(nx, ny);

7.         for (int x = 0; x < nx; x++) {

8.             for (int y = 0; y < ny; y++) {

9.                 input.getNeighborhood(x, y, arr);

10.

11.                 for(int i = 0 ; i < 5; i++){

12.                     for(int j = 0 ; j < 5; j++){

13.                         pixel = pixel + arr[i][j];

14.                 }}

15.                 pixel = pixel / 25.0;

16.                 out.putPixel(x, y, pixel);

17.             }

18.         }

19.         out = rescale(out);

20.         return out;

21.     }

1. separable version

1.     static public ImageAccess doMovingAverage5\_Separable(ImageAccess input) {

2.         int nx = input.getWidth();

3.         int ny = input.getHeight();

4.         ImageAccess out = new ImageAccess(nx, ny);

5.

6.         double colin[]  = new double[ny];

7.         double colout[] = new double[ny];

8.

9.         for (int x = 0; x < nx; x++) {

10.             input.getColumn(x, colin);

11.

12.             Average5(colin, colout);

13.

14.             out.putColumn(x, colout);

15.         }

16.

17.         double rowin[]  = new double[nx];

18.         double rowout[] = new double[nx];

19.         for (int y = 0; y < ny; y++) {

20.             out.getRow(y, rowin);

21.             Average5(rowin, rowout);

22.             out.putRow(y, rowout);

23.         }

24.         out = rescale(out);

25.         return out;

26.     }

27.

28.     static private void Average5(double vin[], double vout[]){

29.

30.         int n = vin.length;

31.

32.         vout[0] = (vin[0] + 2.0 \* vin[1] + 2.0 \* vin[2]) / 5.0;

33.         vout[1] = (vin[0] + vin[1] + vin[2] + 2.0 \*vin[3]) / 5.0;

34.

35.         for (int k = 2; k < n-2; k++) {

36.             vout[k] = (vin[k-2] + vin[k-1] + vin[k] + vin[k+1] + vin[k+2]) / 5.0;

37.         }

38.         vout[n-2] = (vin[n-1] + vin[n-2] + vin[n-3] + 2 \* vin[n-4]) / 5.0;

39.         vout[n-1] = (vin[n-1] + 2.0 \* vin[n-2] + 2.0 \* vin[n-3]) / 5.0;

40.

41.     }

1. recursive version

1.     static public ImageAccess doMovingAverage5\_Recursive(ImageAccess input) {

2.         int nx = input.getWidth();

3.         int ny = input.getHeight();

4.         ImageAccess out = new ImageAccess(nx, ny);

5.

6.         double colin[]  = new double[ny];

7.         double colout[] = new double[ny];

8.

9.         for (int x = 0; x < nx; x++) {

10.             input.getColumn(x, colin);

11.             recursiveAverage5(colin, colout, 0);

12.             out.putColumn(x, colout);

13.         }

14.

15.         double rowin[]  = new double[nx];

16.         double rowout[] = new double[nx];

17.         for (int y = 0; y < ny; y++) {

18.             out.getRow(y, rowin);

19.             recursiveAverage5(rowin, rowout, 0);

20.             out.putRow(y, rowout);

21.         }

22.         out = rescale(out);

23.         return out;

24.     }

25.

26.     static private void recursiveAverage5(double vin[], double vout[], int k) {

27.         int n = vin.length;

28.

29.         if (k == 0) {

30.             vout[0] = (vin[0] + 2.0 \* vin[1] + 2.0 \* vin[2]) / 5.0;

31.             recursiveAverage5(vin, vout, k + 1);

32.         } else if (k == 1) {

33.             vout[1] = (vin[0] + vin[1] + vin[2] + 2.0 \* vin[3]) / 5.0;

34.             recursiveAverage5(vin, vout, k + 1);

35.         } else if (k >= 2 && k < n - 2) {

36.             vout[k] = (vin[k - 2] + vin[k - 1] + vin[k] + vin[k + 1] + vin[k + 2]) / 5.0;

37.             recursiveAverage5(vin, vout, k + 1);

38.         } else if (k == n - 2) {

39.             vout[n - 2] = (vin[n - 1] + vin[n - 2] + vin[n - 3] + 2.0 \* vin[n - 4]) / 5.0;

40.             recursiveAverage5(vin, vout, k + 1);

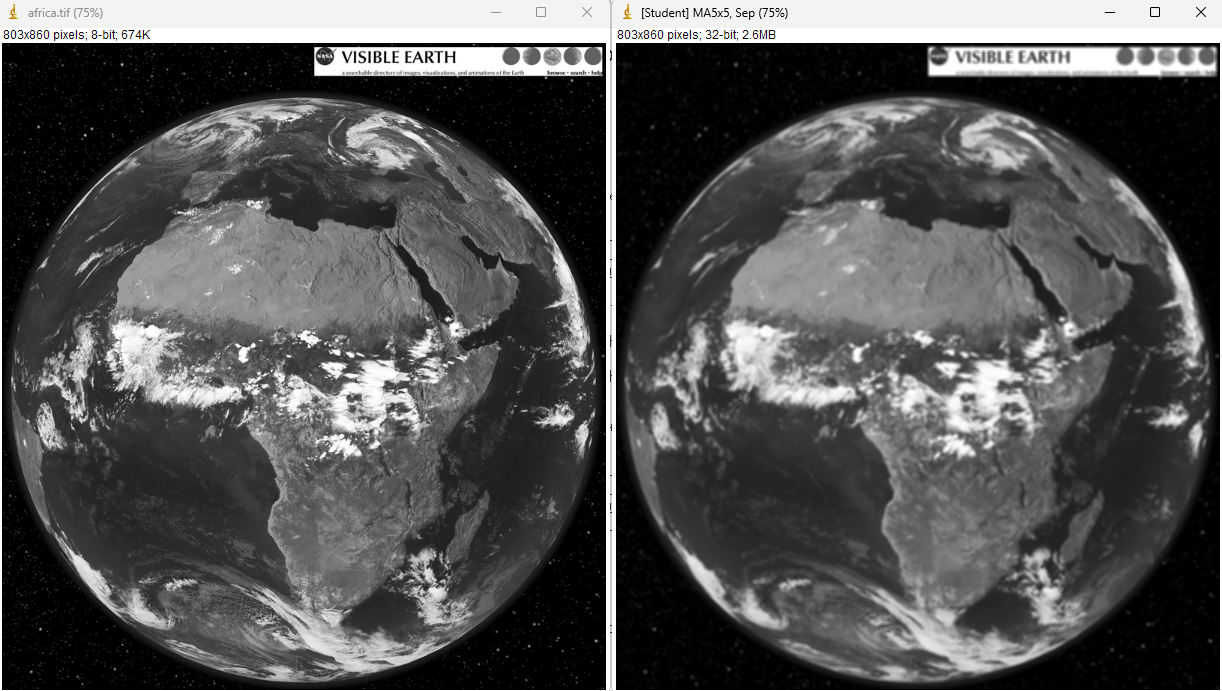
41.         } else if (k == n - 1) {

42.             vout[n - 1] = (vin[n - 1] + 2.0 \* vin[n - 2] + 2.0 \* vin[n - 3]) / 5.0;

43.         }

44.     }

**Image results**



**Question 3.1:**

Segmenting with the Smoothing Operator

L = 15

T = 35

Enter the part of the code you wrote.

Note: imagej presented an error when trying to implement the recursive mode due to memory issues, so the mode without recursion was maintained.

1.      static public ImageAccess doMovingAverageL\_Recursive(ImageAccess input, int length) {

2.         int nx = input.getWidth();

3.         int ny = input.getHeight();

4.         ImageAccess out = new ImageAccess(nx, ny);

5.

6.         int halfWindow = length / 2;

7.

8.         for (int row = 0; row < nx; row++) {

9.             for (int col = 0; col < ny; col++) {

10.                 double sum = 0.0;

11.                 int count = 0;

12.

13.                 for (int i = -halfWindow; i <= halfWindow; i++) {

14.                     for (int j = -halfWindow; j <= halfWindow; j++) {

15.                         int rowIndex = row + i;

16.                         int colIndex = col + j;

17.

18.                         // Espelhamento nas bordas

19.                         if (rowIndex < 0) rowIndex = -rowIndex;

20.                         if (rowIndex >= nx) rowIndex = 2 \* nx - rowIndex - 2;

21.                         if (colIndex < 0) colIndex = -colIndex;

22.                         if (colIndex >= ny) colIndex = 2 \* ny - colIndex - 2;

23.

24.                         sum += input.getPixel(rowIndex, colIndex);

25.                         count++;

26.                     }

27.                 }

28.                 out.putPixel(row, col, sum / count);

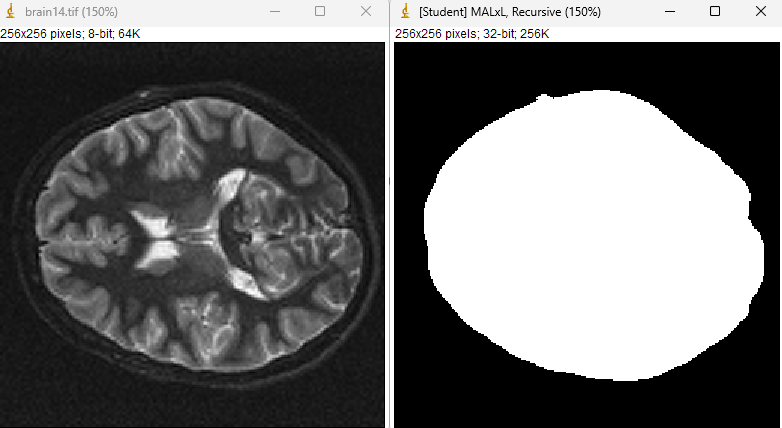
29.             }

30.         }

31.

32.         return rescale(out);}

**Image results**



**Question 3.2:**

Describe the procedure

Enter the part of the code you wrote

**Question 4:**

Sobel Operator

Enter the part of the code written for this exercise

1.     static public ImageAccess doSobel(ImageAccess input) {

2.         int nx = input.getWidth();

3.         int ny = input.getHeight();

4.         double arr[][] = new double[3][3];

5.         double pixel;

6.         double pixel\_x;

7.         double pixel\_y;

8.         ImageAccess out = new ImageAccess(nx, ny);

9.         for (int x = 0; x < nx; x++) {

10.             for (int y = 0; y < ny; y++) {

11.                 input.getNeighborhood(x, y, arr);

12.                 pixel\_x = arr[2][1] + 2\*arr[2][1] + arr[2][1] - arr[0][0]- 2\*arr[0][1]- arr[0][2];

13.                 pixel\_x = Math.pow(pixel\_x, 2);

14.                 pixel\_y = - arr[2][0] - 2\*arr[1][0] - arr[0][0] + arr[0][2] + 2\*arr[1][2] + arr[2][2];

15.                 pixel\_x = Math.pow(pixel\_y, 2);

16.                 pixel = Math.sqrt(pixel\_x + pixel\_y);

17.

18.                 out.putPixel(x, y, pixel);

19.             }

20.         }

21.         out = rescale(out);

22.         return out;

23.     }