Computer Vision Programming Language: Java

Project #5 Edge Detection

Andres Quintero

Due Date:

Soft copy: 3/10/2020

Hard copy: 3/12/2020

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Step 0: open the image and read the image header

dynamically allocate mirrorFramedAry and all the edge arrays

Step 1: loadImage (mirrorFramedAry)

// load input file to mirrorFramedAry begin at (1,1)

Step 2: mirrowFramed (mirrorFramedAry)

Step 3: process the mirrorFramedAry, from left to right and top to bottom

begin at (1, 1) // process all pixels!!!

RobertRightDiag(i,j)🡨 abs(convoluteRobert (i,j, maskRobertRightDiag))

RobertLeftDiag(i,j) 🡨 abs (convoluteRobert (i,j, maskRobertLeftDiag))

SobelRightDiag(i,j) 🡨 abs(convoluteSobel (i,j, maskSobelRightDiag))

SobelLeftDiag(i,j)🡨 abs (convoluteSobel (i,j, maskSobelLeftDiag))

GradiantEdge(i,j) 🡨 computeGradient(i,j)

Step 4: repeat step 3 until all pixels inside of the frame are processed.

Step 5: addTwoArys (RobertRightDiag, RobertLeftDiag, edgeSum)

output RobertRightDiag to deBugOut file // with caption

output RobertLeftDiag to deBugOut file // with caption

output input image header to RobertEdgeOut file

output edgeSum to RobertEdgeOut file // begin at edgeSum[1][1]

Step 6: addTwoArys (SobelRightDiag, SobelLeftDiag, edgeSum)

output SobelRightDiag to deBugOut file // with caption

output SobelLeftDiag to deBugOut file // with caption

output input image header to SobelEdgeOut file

output edgeSum to SobelEdgeOut file // begin at edgeSum[1][1]

Step 7: output input image header to GradiantEdgeOut file

output GradiantEdge to GradiantEdgeOut file //begin at GradiantEdge[1][1]

Step 8: close all files

**Source code:**

import java.util.\*;

import java.io.\*;

import java.lang.Math;

public class Main {

public static void main(String[] args) {

// Declaring varibles and opening all Files

int numRows, numCols, minVal, maxVal;

Scanner inFile = null;

PrintWriter RobertEdgeOut = null;

PrintWriter SobelEdgeOut = null;

PrintWriter GradientEdgeOut = null;

PrintWriter prettyOut = null;

PrintWriter deBugOut = null;

try {

inFile = new Scanner(new File(args[0]));

} catch (FileNotFoundException err) {

System.out.println("Error in opening inputFile: " + err);

}

try {

RobertEdgeOut = new PrintWriter(args[1]);

SobelEdgeOut = new PrintWriter(args[2]);

GradientEdgeOut = new PrintWriter(args[3]);

prettyOut = new PrintWriter(args[4]);

deBugOut = new PrintWriter(args[5]);

} catch (FileNotFoundException err) {

System.out.println("Error in opening files: " + err);

}

//

// Main stuff

//

// Reading header and setting values

numRows = inFile.nextInt();

numCols = inFile.nextInt();

minVal = inFile.nextInt();

maxVal = inFile.nextInt();

// Masks hard code okay

int[][] maskRobertRightDiag = {{1,-1}, {-1,1}};

int[][] maskRobertLeftDiag = {{-1,1}, {1,-1}};

int[][] maskSobelRightDiag = {{0,1,2} , {-1,0,1}, {-2,-1,0}};

int[][] maskSobelLeftDiag = {{2,1,0} , {1,0,-1}, {0,-1,-2}};

// Dynamic 2D array

int[][] mirrorFramedAry = new int[numRows+2][numCols+2];

int[][] RobertRightDiag = new int[numRows+2][numCols+2];

int[][] RobertLeftDiag = new int[numRows+2][numCols+2];

int[][] SobelRightDiag= new int[numRows+2][numCols+2];

int[][] SobelLeftDiag = new int[numRows+2][numCols+2];

int[][] GradientEdge = new int[numRows+2][numCols+2];

int[][] edgeSum = new int[numRows+2][numCols+2];

// STEP 1

loadImage(mirrorFramedAry, inFile);

// STEP 2

mirrorFraming(mirrorFramedAry);

// STEP 3 - STEP 4

for(int i = 1; i < mirrorFramedAry.length-1; i++){

for(int j = 1; j < mirrorFramedAry[0].length-1; j++){

//for every pixel do the edge decttions

RobertRightDiag[i][j] = Math.abs( convoluteRobert(i, j, maskRobertRightDiag, mirrorFramedAry) );

RobertLeftDiag[i][j] = Math.abs( convoluteRobert(i, j, maskRobertLeftDiag, mirrorFramedAry) );

SobelRightDiag[i][j] = Math.abs( convoluteSobel(i, j, maskSobelRightDiag, mirrorFramedAry) );

SobelLeftDiag[i][j] = Math.abs( convoluteSobel(i, j, maskSobelLeftDiag, mirrorFramedAry) );

GradientEdge[i][j] = computeGradient(i, j, mirrorFramedAry);

}

}

// STEP 5

deBugOut.println("RobertRightDiag:");

print2DArrayFrame(RobertRightDiag, deBugOut);

deBugOut.println("RobertLeftDiag:");

print2DArrayFrame(RobertLeftDiag, deBugOut);

addTwoArrays(RobertRightDiag, RobertLeftDiag, edgeSum);

RobertEdgeOut.println(numRows + " " + numCols + " " + findMinVal(edgeSum) + " " + findMaxVal(edgeSum));

print2DArray(edgeSum, RobertEdgeOut);

// STEP 6

addTwoArrays(SobelRightDiag, SobelLeftDiag, edgeSum);

deBugOut.println("SobelRightDiag:");

print2DArrayFrame(SobelRightDiag, deBugOut);

deBugOut.println("SobelLeftDiag:");

print2DArrayFrame(SobelLeftDiag, deBugOut);

SobelEdgeOut.println(numRows + " " + numCols + " " + findMinVal(edgeSum) + " " + findMaxVal(edgeSum));

print2DArray(edgeSum, SobelEdgeOut);

// STEP 7

GradientEdgeOut.println(numRows + " " + numCols + " " + findMinVal(GradientEdge) + " " + findMaxVal(GradientEdge));

print2DArray(GradientEdge, GradientEdgeOut);

// STEP 8

inFile.close();

RobertEdgeOut.close();

SobelEdgeOut.close();

GradientEdgeOut.close();

prettyOut.close();

deBugOut.close();

}

//Functions

static int findMaxVal(int[][] Ary){

int max = 0;

for(int i = 1; i< Ary.length-1; i++){

for(int j = 1; j < Ary[0].length-1; j++){

if(Ary[i][j] > max){max = Ary[i][j];}

}

}

return max;

}

static int findMinVal(int[][] Ary){

int min = Integer.MAX\_VALUE;

for(int i = 1; i< Ary.length-1; i++){

for(int j = 1; j < Ary[0].length-1; j++){

if(Ary[i][j] < min){min = Ary[i][j];}

}

}

return min;

}

static void addTwoArrays(int[][] Ary1, int[][] Ary2, int[][] Ary3){

for(int i = 0; i < Ary3.length; i++){

for(int j = 0; j < Ary3[0].length; j++){

Ary3[i][j] = Ary2[i][j] + Ary2[i][j];

}

}

}

static int computeGradient(int i, int j, int[][] mirrorFramedAry){

int x , a, b;

x = mirrorFramedAry[i][j];

a = mirrorFramedAry[i][j+1];

b = mirrorFramedAry [i+1][j];

return (int) Math.sqrt( Math.pow((x-b), 2) + Math.pow((x-a), 2) );

}

static int convoluteSobel(int i, int j, int[][] mask, int[][] mirrorFramedAry){

int total = 0;

int rowOffset = i - mask.length/2;

int colOffset = j - mask[0].length/2;

for(int r = 0; r < mask.length; r++){

for(int c = 0; c < mask[0].length; c++){

total += mask[r][c] \* mirrorFramedAry[r+rowOffset][c+colOffset];

}

}

return total;

}

static int convoluteRobert(int i, int j, int[][] mask, int[][] mirrorFramedAry){

int total = 0;

for(int r = 0; r < 2; r++){

for(int c = 0; c < 2; c++){

total += mask[r][c] \* mirrorFramedAry[r+i][c+j];

}

}

return total;

}

static void mirrorFraming(int[][] frameAry){

int rows = frameAry.length;

int cols = frameAry[0].length;

for(int i = 0; i < cols; i++){

frameAry[0][i] = frameAry[1][i];

frameAry[rows-1][i] = frameAry[rows-2][i];

}

for(int i = 0; i < rows; i++){

frameAry[i][0] = frameAry[i][1];

frameAry[i][cols-1] = frameAry[i][cols-2];

}

}

static void loadImage(int[][] mirrorFrameAry, Scanner image){

int value;

for (int i = 1; i < mirrorFrameAry.length-1; i++ ){

for (int j = 1; j < mirrorFrameAry[0].length-1 ; j++ ) {

value = image.nextInt();

mirrorFrameAry[i][j] = value;

}

}

}

//MY Functions

static void print2DArrayFrame(int[][] array, PrintWriter outFile){

for(int i = 0; i< array.length; i++){

for(int j = 0; j < array[0].length; j++){

outFile.print(array[i][j] + " ");

}

outFile.println();

}

}

static void print2DArray(int[][] array, PrintWriter outFile){

for(int i = 1; i< array.length-1; i++){

for(int j = 1; j < array[0].length-1; j++){

outFile.print(array[i][j] + " ");

}

outFile.println();

}

}

}

**Robert Edge Image - Histogram, Bi-Gaussian Curve, PrettyPrint**

**A screenshot of a social media post

Description automatically generated**

**A screenshot of a social media post

Description automatically generated**

A close up of a map

Description automatically generated

**END OF ROBERT IMAGE OUTPUTS**

**Sobel Edge Image - Histogram, Bi-Gaussian Curve, PrettyPrint**

**A close up of a device

Description automatically generatedA close up of a device

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**A**

**A screenshot of a cell phone

Description automatically generated**

**A close up of a map

Description automatically generated**

**END OF SOBEL IMAGE OUTPUTS**

**Gradient Edge Image - Histogram, Bi-Gaussian Curve, PrettyPrint**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**END OF GRADIENT IMAGE OUTPUTS**