Computer Vision and Digital Image Processing

Assignment 3

WEI LI

Objective

The purpose of the assignment is to explore the spatial-domain convolution while using the provided simulated cake image to determine the efficiency of the Sobel edge detection algorithm.

Background

Spatial domain convolution algorithms are widely used in image processing. By filtering out the image pixels we want, we can achieve several tasks like edge detection.

Algorithms/Functions

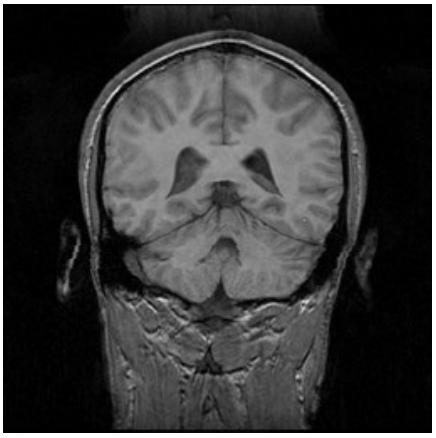
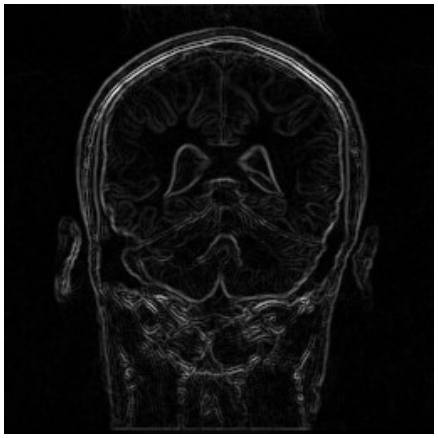
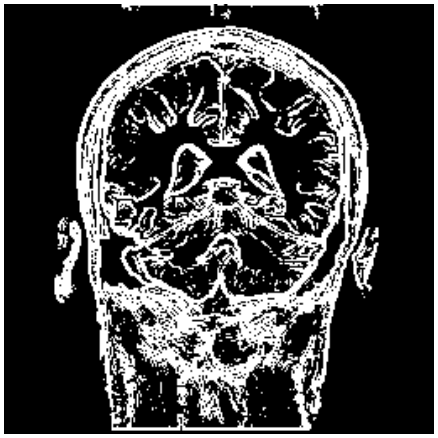
I use Sobel edge detection algorithm and 3x3 average filtering algorithm. Both are written in ‘sobel.c’ and ‘smooth.c’ files. I also write the read and write function in ‘wr.c’ file.

Results

Part 1 Results

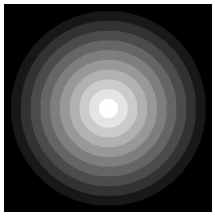
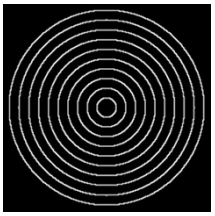
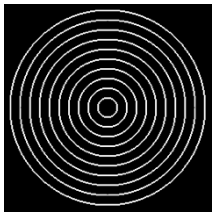
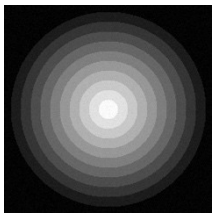
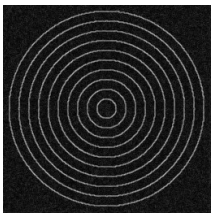
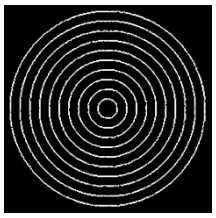
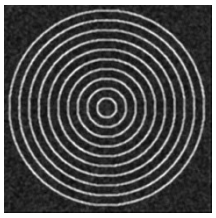
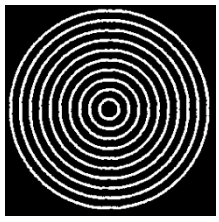
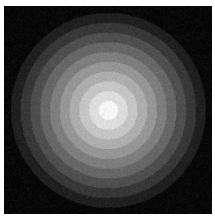
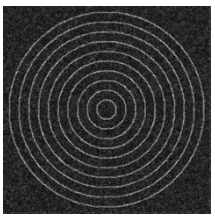
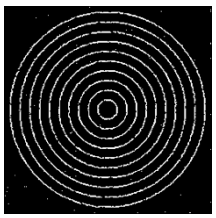
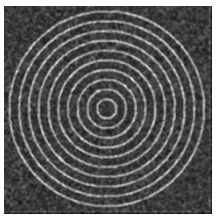
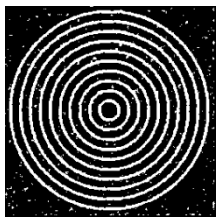
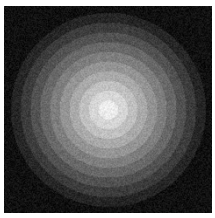
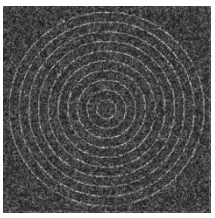
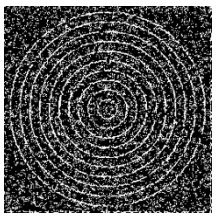
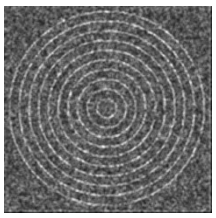
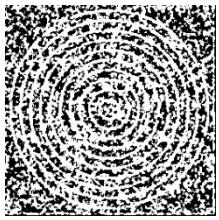
Test results on the mri.pgm image.

Table 1: Results of Sobel on mri.pgm image

Original Image	**Sobel Image**	**Binary Image Threshold=30**
		

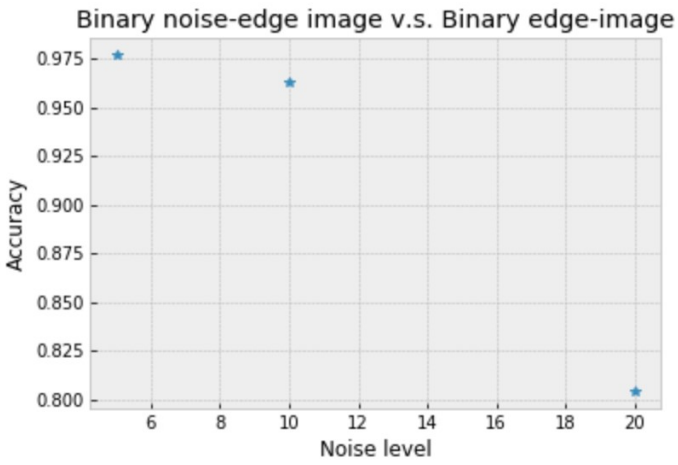
Part 2 Results

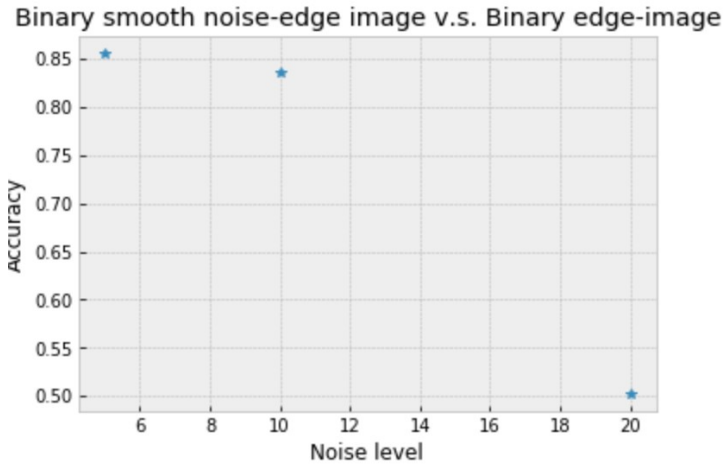
Table 2: Sobel Edge Detection Results on cake.pgm image

Noise Level	Original Image + Noise	Sobel Image	Sobel -> Binarized	Smoothed -> Sobel	Smoothed -> Sobel -> Binarized
No Noise				**N/A**	**N/A**
First noise level=5					
Second noise level=10					
Third noise level=20					

Two plots:

I use threshold = 90 to plot the accuracy based on three noise level(5,10,20)





Observations

Based on the part 1 results, we can observe the differences by changing the threshold of the output binary edge image. And based on the part 2 results, we can further notice that the accuracy would drop when the noise level increases.

Conclusions

This assignment I successfully implement Sobel edge detection algorithm and try a bit on selecting the best threshold of binary edge image. And I also implement the 3x3 averaging filter on the noisy edge image to compare the accuracy with binary edge image.

Code:

Part1:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include <limits.h>
#include "wr.h"
#include "sobel.h"

int main(int argc, char *argv[]){
      int min = INT_MAX;     //set minimum doubled value
      int max = 0;          //set maximum doubled value

      unsigned sizeX;              //image width
      unsigned sizeY;              //image height
    unsigned levels;            //image brightness
      unsigned char *image;       //image 1D array
    unsigned char *outputMRI;   //mri image output
    int *data;                  //data type can be either int or float
    int *data_sobel;
    int *data_sobel_t;

      //read image from file
      readImage("mri.pgm", &sizeX, &sizeY, &levels, &image);

      data = (int*)calloc(sizeof(int),(sizeX+2)*(sizeY+2));       //allocate 258x258
    data_sobel_t = data_sobel =(int*)calloc(sizeof(int),(sizeX+2)*(sizeY+2));
      outputMRI = (unsigned char *)malloc(sizeX*sizeY);

/************************ Image Processing ******************************/

    //put original image pixel values in data array, prepare for the padding
    if(data){
        for(int i=1; i<sizeX+1; i++){
            for(int j=1; j<sizeY+1; j++){
                data[i*(sizeX+2) + j] = image[(i-1)*sizeX + (j-1)];
            }
        }
    }

    //Apply Sobel filter
    sobel(data, data_sobel);

    //Normalization
    for(int i = 0; i < (sizeX+2)*(sizeY+2); i++){
        if (data_sobel[i] < min)
            min = data_sobel[i];
        if (data_sobel[i] > max)
            max = data_sobel[i];
    }
    for(int i = 0; i < (sizeX+2)*(sizeY+2); i++){
        data_sobel[i] = levels*(data_sobel[i]-min)/(max-min);
    }

    for(int i=1; i<sizeX+1; i++){
        for(int j=1; j<sizeY+1; j++){
            outputMRI[(i-1)*sizeX + (j-1)] = data_sobel[i*(sizeX+2) + j];
        }
    }

    //write sobel image to file
    writeImage("EdgeImage.pgm", sizeX, sizeY, levels, outputMRI);

    for(int i = 0; i < (sizeX+2)*(sizeY+2); i++){
```

```
        data_sobel[i] = data_sobel[i];
    }

    //Set a threshold T
    int T = 30;
    for(int i = 0; i < (sizeX+2)*(sizeY+2); i++){
        if(data_sobel[i] > T)
            data_sobel[i] = levels;
        else
            data_sobel[i] = 0;
    }

    for(int i=1; i<sizeX+1; i++){
        for(int j=1; j<sizeY+1; j++){
             outputMRI[(i-1)*sizeX + (j-1)] = data_sobel[i*(sizeX+2) + j];
        }
    }

      //write image to file
      writeImage("BinaryEdgeImage.pgm", sizeX, sizeY, levels, outputMRI);

      free(data);
}
```

Part2:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include <limits.h>
#include "wr.h"
#include "sobel.h"
#include "smooth.h"
#include "time.h"

int main(int argc, char *argv[]){
      int min = INT_MAX;      //set minimum value
      int max = 0;          //set maximum value
    int T = 90;             //set a threshold value for best binary image output

    unsigned sizeX;                 //image width
    unsigned sizeY;                 //image height
    unsigned levels;                //image brightness
      unsigned char *outputImage, *outputImage1, *outputImage2, *outputImage3;     //image output
    unsigned char *outputNoise;
      unsigned char *image;           //image 1D array

    int *data;                      //data type can be either int or float
    int *data_sobel;
    int *data_sobel_t;
    int *data_smooth;
    int *intImage;

      //read image from file
      readImage("cake.pgm", &sizeX, &sizeY, &levels, &image);

      intImage = (int*)malloc(sizeof(int)*sizeX*sizeY);
      data = (int*)calloc(sizeof(int),(sizeX+2)*(sizeY+2));       //allocate 258x258
    data_smooth = (int*)calloc(sizeof(int),(sizeX+2)*(sizeY+2));
    data_sobel_t =(int*)calloc(sizeof(int),(sizeX+2)*(sizeY+2));
    data_sobel =(int*)calloc(sizeof(int),(sizeX+2)*(sizeY+2));
      outputImage = (unsigned char *)malloc(sizeX*sizeY);
    outputImage1 = (unsigned char *)malloc(sizeX*sizeY);
    outputImage2 = (unsigned char *)malloc(sizeX*sizeY);
    outputImage3 = (unsigned char *)malloc(sizeX*sizeY);
    outputNoise = (unsigned char *)malloc(sizeX*sizeY);
```

```
/************************ Image Processing ******************************/
    //put original image pixel values in data array, prepare for the padding
    if(data){
        for(int i=1; i<sizeX+1; i++){
            for(int j=1; j<sizeY+1; j++){
                data[i*(sizeY+2) + j] = image[(i-1)*sizeX + (j-1)];
            }
        }
    }

    //Apply Sobel filter
    sobel(data, data_sobel);

    //Normalization
    for(int i = 0; i < (sizeX+2)*(sizeY+2); i++){
        if (data_sobel[i] < min)
            min = data_sobel[i];
        if (data_sobel[i] > max)
            max = data_sobel[i];
    }
    for(int i = 0; i < (sizeX+2)*(sizeY+2); i++){
        data_sobel[i] = levels*(data_sobel[i]-min)/(max-min);
    }

    //Allocate filtered image pixels to 256x256
    for(int i = 1; i < sizeX+1; i++){
        for(int j=1; j<sizeY+1; j++){
            outputImage[(i-1)*sizeX + (j-1)] = data_sobel[i*(sizeX+2) + j];
        }
    }

    //write image to file
    writeImage("EdgeImage.pgm", sizeX, sizeY, levels, outputImage);

    //make binary image
    for(int i = 0; i < (sizeX+2)*(sizeY+2); i++){
        if(data_sobel[i] > T)
            data_sobel[i] = levels;
        else
            data_sobel[i] = 0;
    }

    //Allocate filtered image pixels to 256x256
    for(int i = 1; i < sizeX+1; i++){
        for(int j=1; j<sizeY+1; j++){
            outputImage1[(i-1)*sizeX + (j-1)] = data_sobel[i*(sizeX+2) + j];
        }
    }

    //write image to file
    writeImage("BinaryEdgeImage.pgm", sizeX, sizeY, levels, outputImage1);

    //Convert image file to int type and generate random noise r on image pixels
    for(int i = 0; i < sizeX*sizeY; i++){
        int r = rand()%41 - 20;    //noise from -20 to 20
        //int r = rand()%21 - 10;    //noise from -10 to 10
        //int r = rand()%11 - 5;    //noise from -5 to 5
        intImage[i] = image[i];
        intImage[i] = intImage[i] + r;
    }

    //Normalization
    for(int i = 0; i < sizeX*sizeY; i++){
        if (intImage[i] < min)
            min = intImage[i];
        if (intImage[i] > max)
            max = intImage[i];
    }
```

```
for(int i = 0; i < sizeX*sizeY; i++){
    intImage[i] = 255*(intImage[i]-min)/(max-min);
}
//Convert int to unsigned char
for(int i = 0; i < sizeX*sizeY; i++){
    outputNoise[i] = intImage[i];
}

//write noisy image to file
writeImage("NoisyImage.pgm", sizeX, sizeY, levels, outputNoise);

//put original image pixel values in data array, prepare for the padding
if(data){
    for(int i=1; i<sizeX+1; i++){
        for(int j=1; j<sizeY+1; j++){
            data[i*(sizeY+2) + j] = intImage[(i-1)*sizeX + (j-1)];
        }
    }
}

//Apply Sobel filter
sobel(data, data_sobel);

for(int i=0; i<(sizeX+2)*(sizeY+2); i++){
    data_sobel_t[i] = data_sobel[i];
}

//Normalization
for(int i = 0; i < (sizeX+2)*(sizeY+2); i++){
    if (data_sobel[i] < min)
        min = data_sobel[i];
    if (data_sobel[i] > max)
        max = data_sobel[i];
}
for(int i = 0; i < (sizeX+2)*(sizeY+2); i++){
    data_sobel[i] = levels*(data_sobel[i]-min)/(max-min);
}

//Allocate filtered image pixels to 256x256
for(int i = 1; i < sizeX+1; i++){
    for(int j=1; j<sizeY+1; j++){
         outputImage[(i-1)*sizeX + (j-1)] = data_sobel[i*(sizeX+2) + j];
    }
}

  //write image to file
  writeImage("NoisyEdgeImage.pgm", sizeX, sizeY, levels, outputImage);

//make binary image based on thredhold T
for(int i = 0; i < (sizeX+2)*(sizeY+2); i++){
    if(data_sobel[i] > T)
        data_sobel[i] = levels;
    else
        data_sobel[i] = 0;
}

//Allocate filtered image pixels to 256x256
for(int i = 1; i < sizeX+1; i++){
    for(int j=1; j<sizeY+1; j++){
        outputImage2[(i-1)*sizeX + (j-1)] = data_sobel[i*(sizeX+2) + j];
    }
}

//write image to file
writeImage("BinaryNoisyEdgeImage.pgm", sizeX, sizeY, levels, outputImage2);

//Apply averaging filter
smooth(data_sobel_t, data_smooth);
```

```
    //Normalization
    for(int i = 0; i < (sizeX+2)*(sizeY+2); i++){
        if (data_smooth[i] < min)
            min = data_smooth[i];
        if (data_smooth[i] > max)
            max = data_smooth[i];
    }
    for(int i = 0; i < (sizeX+2)*(sizeY+2); i++){
        data_smooth[i] = levels*(data_smooth[i]-min)/(max-min);
    }

    //Allocate filtered image pixels to 256x256
    for(int i = 1; i < sizeX+1; i++){
        for(int j=1; j<sizeY+1; j++){
            outputImage[(i-1)*sizeX + (j-1)] = data_smooth[i*(sizeX+2) + j];
        }
    }

    //write averaging image to file
    writeImage("SmoothNoisyEdgeImage.pgm", sizeX, sizeY, levels, outputImage);

    //make binary image
    for(int i = 0; i < (sizeX+2)*(sizeY+2); i++){
        if(data_smooth[i] > T)
            data_smooth[i] = levels;
        else
            data_smooth[i] = 0;
    }

    //Allocate filtered image pixels to 256x256
    for(int i = 1; i < sizeX+1; i++){
        for(int j=1; j<sizeY+1; j++){
            outputImage3[(i-1)*sizeX + (j-1)] = data_smooth[i*(sizeX+2) + j];
        }
    }

    //write image to file
    writeImage("BinarySmoothNoisyEdgeImage.pgm", sizeX, sizeY, levels, outputImage3);


    int counter1 = 0;
    float accu1, accu2;
    //calculate the accuracy between binary noisy edge image and binary edge image
    for(int i = 0; i < (sizeX+2)*(sizeY+2); i++){
        if(outputImage1[i]==outputImage2[i]){
            counter1++;
        }
    }
    accu1 = (double)counter1/((sizeX+2)*(sizeY+2));
    printf("accuracy1 : %lf \n", accu1);

    int counter2 = 0;
    //calculate the accuracy between binary smooth noisy edge image and binary edge image
    for(int i = 0; i < (sizeX+2)*(sizeY+2); i++){
        if(outputImage1[i]==outputImage3[i]){
            counter2++;
        }
    }
    accu2 = (double)counter2/((sizeX+2)*(sizeY+2));
    printf("accuracy2 : %lf \n", accu2);


      free(data);
}
```