Sean Webster

Computer Vision
Homework 3
Due 4/4/2016

Objective: The objective of this assignment was to explore special-domain convolution while utilizing simulated images to determine the efficiency of the famous "Sobel" edge detection algorithm.

Background: The "Sobel" edge detection algorithm is a well-known algorithm for detecting edges. It works by applying a kernel operator along an image, multiplying its neighbor pixels by degrees to detect lines.

Algorithm: To apply the Sobel kernel to the image, a buffering boundary must be created, due to the kernel being of odd dimensions. I did this by creating a buffer of zeros in the image array. The Sobel kernel was then applied by looping through each pixel, and applying the respective kernel with the equation

$$G_x = |(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)|$$

For the horizontal kernel, and

$$G_y = |(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)|$$

For the vertical kernel. Adding them together gave the complete Sobel image.

To calculate a threshold, any value above the thresholding value was pushed to 255, and the others to zero. I used the guess and check method to determine threshold.

To calculate noise, a random number generator was applied to the image.

Calculating the Gaussian Blur was similar to the Sobel, in that it was a kernel that needed padding to be applied. The kernel equation was

$$1/16z_1 + 1/8z_2 + 1/16z_3 + 1/8z_4 + 1/4z_5 + 1/8z_6 + 1/16z_7 + 1/8z_8 + 1/16z_9$$

The accuracy was calculated by dividing the number of matching pixels by the total amount.

Results:

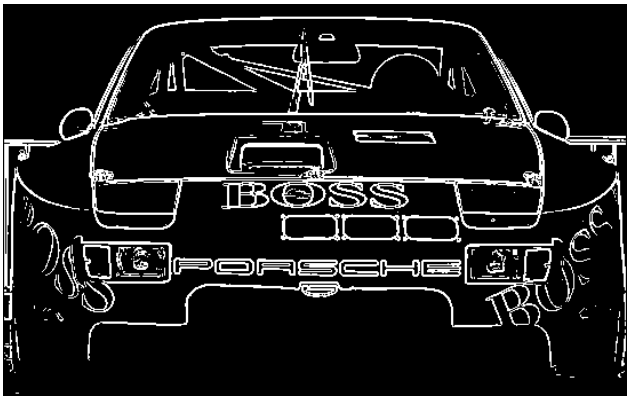


Figure 1(left). Sobel Image of Porsche.
Figure 2(right). Threshold of 60, Sobel of Porsche.

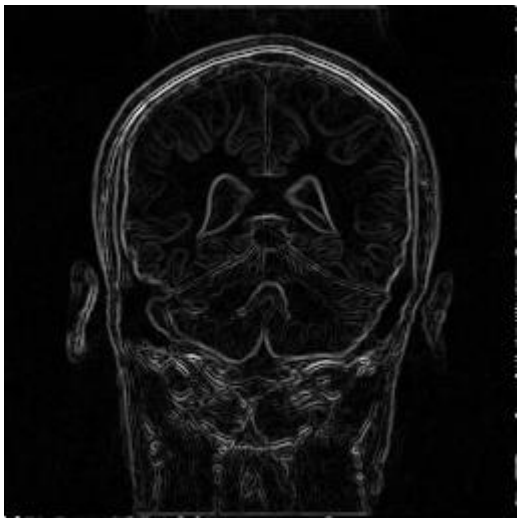


Figure 3(left). Sobel Image of MRI.
Figure 4(right). Threshold of 50, Sobel of MRI.

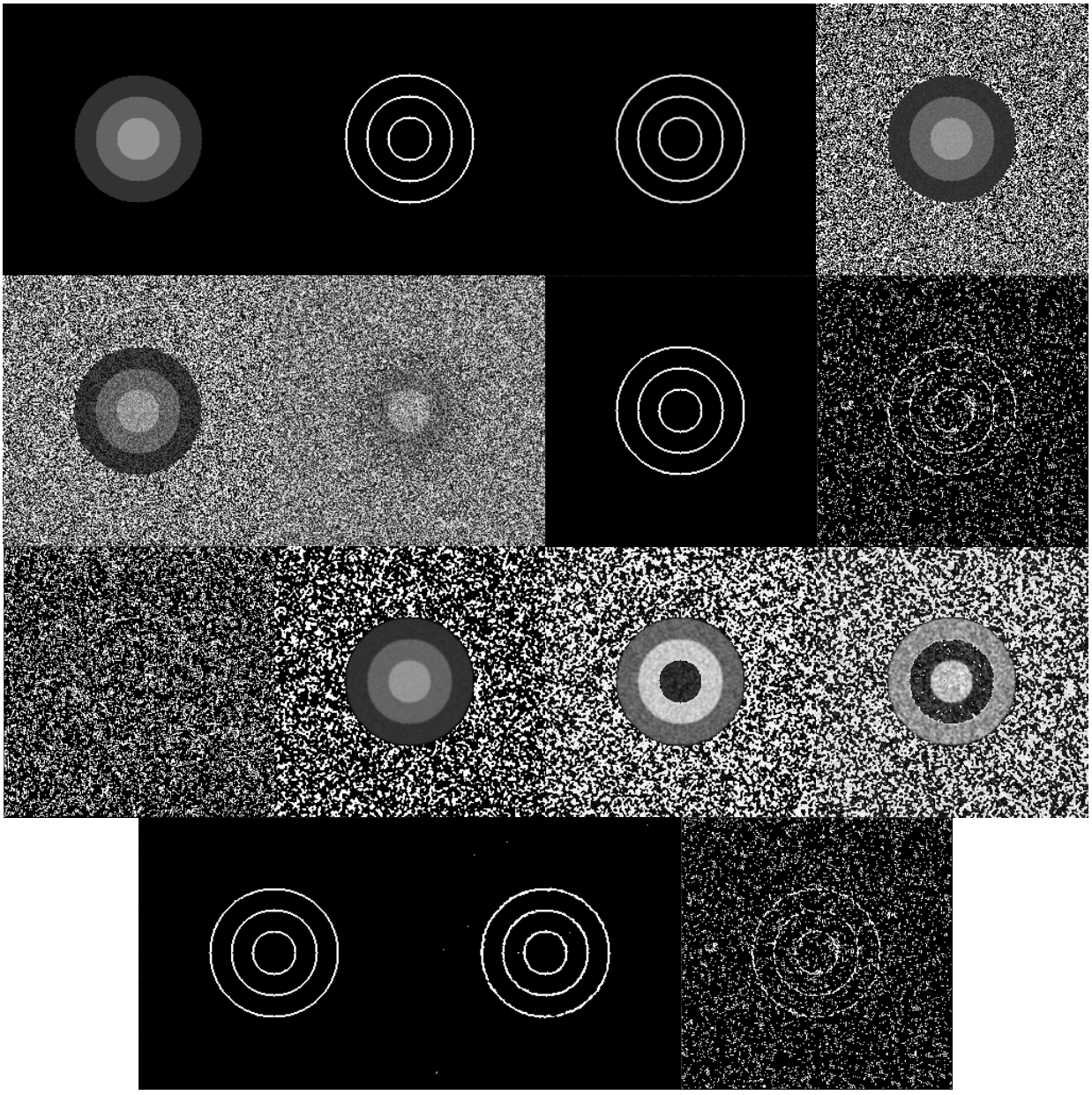


Figure 5-8(top row, left-right). Original Circle Image, Sobel of Circles, Thresholded Image, Circles with Noise at Mag. 10.

Figure 9-12(Second row, left-right). Circles with Noise at Mag. 50, Circles with Noise at Mag. 100, Sobel of Noise 10, Sobel of Noise 50

Figure 13-16(Third row, left-right). Sobel of Noise 100, Gaussian smoothed circles of Noise 10, Gaussian smoothed circles of Noise 50, Gaussian smoothed circles of Noise 100

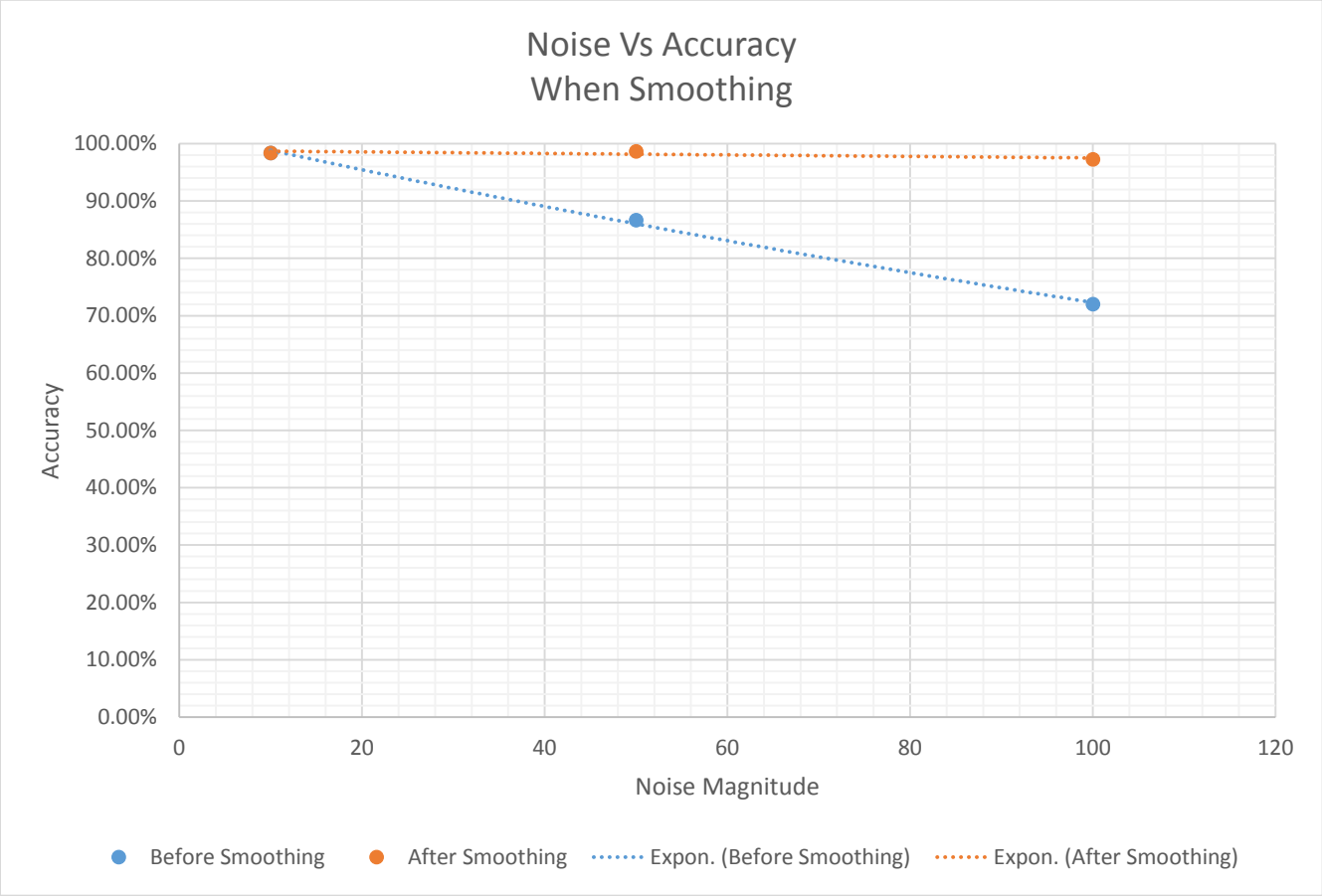
Figure 17-20(Fourth row, left-right). Sobel of Gaussian smoothed circles of Noise 10, Sobel Of Gaussian smoothed circles of Noise 50, Sobel of Gaussian smoothed circles of Noise 100

**note: all circle thresholds = 100*

Table 1 – Noise Vs Accuracy

	Accuracy	
Noise Magnitude	Before Smoothing	After Smoothing
10	0.983932	0.983917
50	0.866837	0.986908
100	0.720505	0.972794

Chart 1. Noise Vs Accuracy



Conclusions and Observations:

Based on the results of the noisy Sobel circles, it was shown that Gaussian Smoothing does indeed improve problems that occur with noise. With little noise, the problem isn't fixed as well as with lots of noise.

During this assignment, I thought that the laplacian of the image was the Sobel of it, but after further googling, I corrected myself. I also ran into many problems due to data type conversions, mostly being float numbers not working properly.

The Sobel Edge Detector Operator is quite a powerful operator, and the only one of its kind that I have used so far. I can see its shortcomings when it comes to noise, but beyond those, it works rather well.

Readme:

Sean Webster

Homework 3 readme

Source Code Files
- HW3.c
- arrays.c
- Spacial.c
- Circles.c

Header Files
- arrays.h
- Spacial.h
- Circles.h

Executables
- HW3

Instructions for running
- Run program, then convert images out of raw format

Inputs

mri	256 x 256	65.5 KB
Porsche	546 x 342	183 KB

Outputs

mri_sobel	256 x 256	65.5 KB
mri_thresh50	256 x 256	65.5 KB
porsche_sobel	546 x 342	183 KB
porsche_thresh60	546 x 342	183 KB
circles	256 x 256	64 KB
circles_sobel	256 x 256	64 KB
circles_thresh100	256 x 256	64 KB
circles_noise10	256 x 256	64 KB
circles_noise50	256 x 256	64 KB
circles_noise100	256 x 256	64 KB
circles_noise_gauss10	256 x 256	64 KB
circles_noise_gauss50	256 x 256	64 KB
circles_noise_gauss100	256 x 256	64 KB
circles_noise_gauss_thresh10	256 x 256	64 KB
circles_noise_gauss_thresh50	256 x 256	64 KB
circles_noise_gauss_thresh100	256 x 256	64 KB

XV Commands
- (echo P5; echo 256 256; echo 255; cat mri_sobel) xv -
- (echo P5; echo 256 256; echo 255; cat mri_thresh50) xv -
- (echo P5; echo 546 342; echo 255; cat porsche_sobel) xv -

```
(echo P5; echo 546 342; echo 255; cat porsche_thresh60) xv -
(echo P5; echo 256 256; echo 255; cat circles) xv -
(echo P5; echo 256 256; echo 255; cat circles_sobel) xv -
(echo P5; echo 256 256; echo 255; cat circles_thresh100) xv -
(echo P5; echo 256 256; echo 255; cat circles_noise10) xv -
(echo P5; echo 256 256; echo 255; cat circles_noise50) xv -
(echo P5; echo 256 256; echo 255; cat circles_noise100) xv -
(echo P5; echo 256 256; echo 255; cat circles_noise_gauss10) xv -
(echo P5; echo 256 256; echo 255; cat circles_noise_gauss50) xv -
(echo P5; echo 256 256; echo 255; cat circles_noise_gauss100) xv -
(echo P5; echo 256 256; echo 255; cat circles_noise_gauss_thresh10) xv -
(echo P5; echo 256 256; echo 255; cat circles_noise_gauss_thresh50) xv -
(echo P5; echo 256 256; echo 255; cat circles_noise_gauss_thresh100) xv -
```

Part 1:

```
/*--------------- HW3.c ---------------

by:       Sean Webster
          Computer Vision
          Due 4/4/2016

-----------------------------------------*/

#include <math.h>
   #include <stdlib.h>
   #include <stdio.h>
   #include <string.h>

   #include "arrays.h"
   #include "Spacial.h"
   #include "Circles.h"

   int main()
   {
         /*----------------------------Part 1---------------------------------
  --*/
         int ysize = 342;
         int xsize = 546;
         int padding = 1;


         float **inImage = ImageInit(xsize, ysize);
         float **padImage = ImageInit(xsize + 2*padding, ysize + 2*padding);
         float **sobelImage = ImageInit(xsize, ysize);
         float **threshImage = ImageInit(xsize, ysize);


         OpenImage("porsche", xsize, ysize, inImage);
         PadImage(inImage, padImage, xsize, ysize, padding);
         Sobel(padImage, sobelImage, xsize + 2*padding, ysize + 2*padding);
         SaveImage("porsche", "_sobel", xsize, ysize, sobelImage);
         BinThresh(sobelImage, threshImage, xsize, ysize, 60);
         SaveImage("porsche", "_thresh60", xsize, ysize, threshImage);


         ysize = 256;
         xsize = 256;
         OpenImage("mri", xsize, ysize, inImage);
         PadImage(inImage, padImage, xsize, ysize, padding);
         Sobel(padImage, sobelImage, xsize + 2 * padding, ysize + 2 * padding);
         SaveImage("mri", "_sobel", xsize, ysize, sobelImage);
         BinThresh(sobelImage, threshImage, xsize, ysize, 50);
         SaveImage("mri", "_thresh50", xsize, ysize, threshImage);

         free(inImage);
         free(padImage);
         free(sobelImage);
         free(threshImage);

         /*----------------------------Part 2---------------------------------
  --*/
         xsize = ysize = 256;
         float **circImage = ImageInit(xsize, ysize);
         float **padCirc = ImageInit(xsize + 2 * padding, ysize + 2 * padding);
         float **sobelCirc = ImageInit(xsize, ysize);
         float **threshCirc = ImageInit(xsize, ysize);
         float **noiseCirc = ImageInit(xsize, ysize);
         float **padNoiseCirc = ImageInit(xsize + 2 * padding, ysize + 2 *
  padding);
         float **sobelNoiseCirc = ImageInit(xsize, ysize);
         float **noiseGaussCirc = ImageInit(xsize, ysize);
```

```
64.         float **padNoiseGaussCirc = ImageInit(xsize + 2 * padding, ysize + 2 *
  padding);
65.         float **sobelNoiseGaussCirc = ImageInit(xsize, ysize);
66.         float **sobelNoiseGaussThreshCirc = ImageInit(xsize, ysize);
67.
68.         float **test = ImageInit(xsize + 2, ysize +2);
69.
70.         int c = 256 / 2 - 1;
71.         CreateCircle(circImage, ysize, xsize, c, c, 50, 60);
72.         CreateCircle(circImage, ysize, xsize, c, c, 50, 40);
73.         CreateCircle(circImage, ysize, xsize, c, c, 50, 20);
74.
75.
76.         SaveImage("circles", "", xsize, ysize, circImage);
77.         PadImage(circImage, padCirc, xsize, ysize, padding);
78.         Sobel(padCirc, sobelCirc, xsize + 2 * padding, ysize + 2 * padding);
79.         SaveImage("circles", "_sobel", xsize, ysize, sobelCirc);
80.
81.         BinThresh(sobelCirc, threshCirc, xsize, ysize, 100);
82.         SaveImage("circles", "_thresh100", xsize, ysize, threshCirc);
83.
84.         // Noise = 10
85.         AddUniNoise(circImage, noiseCirc, xsize, ysize, 10);
86.         SaveImage("circles", "_noise10", xsize, ysize, noiseCirc);
87.
88.         PadImage(noiseCirc, padNoiseCirc, xsize, ysize, padding);
89.         Sobel(padNoiseCirc, sobelNoiseCirc, xsize + 2 * padding, ysize + 2 *
  padding);
90.         BinThresh(sobelNoiseCirc, sobelNoiseCirc, xsize, ysize, 100);
91.         SaveImage("circles", "_noise_sobel10", xsize, ysize, sobelNoiseCirc);
92.         AccuracyReport(threshCirc, sobelNoiseCirc, xsize, ysize);
93.
94.         GaussianSmooth(padNoiseCirc, noiseGaussCirc, xsize + 2 * padding,
  ysize + 2 * padding);
95.         SaveImage("circles", "_noise_gauss10", xsize, ysize, noiseGaussCirc);
96.         PadImage(noiseGaussCirc, padNoiseGaussCirc, xsize, ysize, padding);
97.         Sobel(padNoiseGaussCirc, sobelNoiseGaussCirc, xsize + 2 * padding,
  ysize + 2 * padding);
98.         BinThresh(sobelNoiseGaussCirc, sobelNoiseGaussThreshCirc, xsize,
  ysize, 100);
99.         SaveImage("circles", "_noise_sobel_gauss_thresh10", xsize, ysize,
  sobelNoiseGaussThreshCirc);
100.        AccuracyReport(threshCirc, sobelNoiseGaussThreshCirc, xsize, ysize);
101.
102.        // Noise = 50
103.        AddUniNoise(circImage, noiseCirc, xsize, ysize, 50);
104.        SaveImage("circles", "_noise50", xsize, ysize, noiseCirc);
105.
106.        PadImage(noiseCirc, padNoiseCirc, xsize, ysize, padding);
107.        Sobel(padNoiseCirc, sobelNoiseCirc, xsize + 2 * padding, ysize + 2 *
  padding);
108.        BinThresh(sobelNoiseCirc, sobelNoiseCirc, xsize, ysize, 100);
109.        SaveImage("circles", "_noise_sobel50", xsize, ysize, sobelNoiseCirc);
110.        AccuracyReport(threshCirc, sobelNoiseCirc, xsize, ysize);
111.
112.        GaussianSmooth(padNoiseCirc, noiseGaussCirc, xsize + 2 * padding,
  ysize + 2 * padding);
113.        SaveImage("circles", "_noise_gauss50", xsize, ysize, noiseGaussCirc);
114.        PadImage(noiseGaussCirc, padNoiseGaussCirc, xsize, ysize, padding);
115.        Sobel(padNoiseGaussCirc, sobelNoiseGaussCirc, xsize + 2 * padding,
  ysize + 2 * padding);
116.        BinThresh(sobelNoiseGaussCirc, sobelNoiseGaussThreshCirc, xsize,
  ysize, 100);
117.        SaveImage("circles", "_noise_sobel_gauss_thresh50", xsize, ysize,
  sobelNoiseGaussThreshCirc);
118.        AccuracyReport(threshCirc, sobelNoiseGaussThreshCirc, xsize, ysize);
119.
```

```
120.
121.        // Noise = 100
122.        AddUniNoise(circImage, noiseCirc, xsize, ysize, 100);
123.        SaveImage("circles", "_noise100", xsize, ysize, noiseCirc);
124.
125.        PadImage(noiseCirc, padNoiseCirc, xsize, ysize, padding);
126.        Sobel(padNoiseCirc, sobelNoiseCirc, xsize + 2 * padding, ysize + 2 *
  padding);
127.        BinThresh(sobelNoiseCirc, sobelNoiseCirc, xsize, ysize, 100);
128.        SaveImage("circles", "_noise_sobel100", xsize, ysize, sobelNoiseCirc);
129.        AccuracyReport(threshCirc, sobelNoiseCirc, xsize, ysize);
130.
131.        GaussianSmooth(padNoiseCirc, noiseGaussCirc, xsize + 2 * padding,
  ysize + 2 * padding);
132.        SaveImage("circles", "_noise_gauss100", xsize, ysize, noiseGaussCirc);
133.        PadImage(noiseGaussCirc, padNoiseGaussCirc, xsize, ysize, padding);
134.        Sobel(padNoiseGaussCirc, sobelNoiseGaussCirc, xsize + 2 * padding,
  ysize + 2 * padding);
135.        BinThresh(sobelNoiseGaussCirc, sobelNoiseGaussThreshCirc, xsize,
  ysize, 100);
136.        SaveImage("circles", "_noise_sobel_gauss_thresh100", xsize, ysize,
  sobelNoiseGaussThreshCirc);
137.        AccuracyReport(threshCirc, sobelNoiseGaussThreshCirc, xsize, ysize);
138.
139.
140.        free(circImage);
141.        free(padCirc);
142.        free(sobelCirc);
143.        free(threshCirc);
144.        free(noiseCirc);
145.        free(padNoiseCirc);
146.        free(sobelNoiseCirc);
147.        free(noiseGaussCirc);
148.        free(padNoiseGaussCirc);
149.        free(sobelNoiseGaussCirc);
150.        free(sobelNoiseGaussThreshCirc);
151.
152.        return 0;
153.  }
```

```
154.  /*--------------- Spacial.h ---------------
155.
156.              by:   Sean Webster
157.
158.              PURPOSE
159.              function prototypes for Spacial.c
160.
161.  ------------------------------------------*/
162.
163.
164.  #include "arrays.h"
165.  #include <math.h>
166.  #include <stdlib.h>
167.  #include <stdio.h>
168.  #include <float.h>
169.
170.
171.  #ifndef SPACIAL_H
172.  #define SPACIAL_H
173.
174.  void PadImage(float **oData, float **nData, int xsize, int ysize, int
  pSize);
175.  void Sobel(float **inData, float **lineData, int xsize, int ysize);
```

```
176.  void BinThresh(float **inData, float **outData, int xsize, int ysize, int
  thresh);
177.  void AddUniNoise(float** inData, float **outData, int xsize, int ysize, int
  intens);
178.  void GaussianSmooth(float **inData, float **outData, int xsize, int ysize);
179.
#endif


180.  /*--------------- Spacial.c ---------------
181.  PURPOSE: Holds functions for spacial image manipulation
182.
183.  by:         Sean Webster
184.              Computer Vision
185.              Due 4/4/2016
186.  ----------------------------------------------*/
187.  #include "Spacial.h"
188.
189.
190.  /*-------------------- PadImage()-----------------------------------
191.
192.  PURPOSE
193.  Pads image, adding 0's for spacial filtering
194.
195.  INPUT PARAMETERS
196.  -inData:    old 2D array to be input
197.  -outData:   new 2D array to be input
198.  -xsize:           size of array in x direction
199.  -ysize:           size of array in y direction
200.  ------------------------------------------------------------------------*/
201.  void PadImage(float **inData, float **outData, int xsize, int ysize, int
  pSize)
202.  {
203.        int i, j;
204.        for (i = 0; i < ysize; i++)
205.        {
206.              for (j = 0; j < xsize; j++)
207.              {
208.                        outData[i + pSize][j + pSize] = inData[i][j];
209.              }
210.        }
211.  }
212.
213.
214.  /*-------------------- Sobel()-------------------------------------
215.
216.  PURPOSE
217.  Masks image with sobel operator
218.
219.  INPUT PARAMETERS
220.  -inData:    old 2D array to be input
221.  -outData:   new 2D array to be input
222.  -xsize:           size of array in x direction
223.  -ysize:           size of array in y direction
224.  ------------------------------------------------------------------------*/
225.  void Sobel(float **inData, float **outData, int xsize, int ysize)
226.  {
227.        float **temp1 = ImageInit(xsize, ysize);
228.        float **temp2 = ImageInit(xsize, ysize);
229.
230.        int vertKernel[3][3] = {     { -1,  0,  1 },
231.                                                { -2,  0,  2 },
232.                                                { -1,  0,  1 } };
233.        int horzKernel[3][3] = {     { 1,  2,  1 },
234.                                                { 0,  0,  0 },
235.                                                { -1, -2, -1 } };
236.
```

```
237.
238.        int x, y, i, j;
239.        double pixel_value;
240.        double max = -DBL_MAX;
241.
242.        //Vertical Sobel Mask
243.        for (y = 1; y < ysize - 1; y++)
244.        {
245.              for (x = 1; x < xsize - 1; x++)
246.              {
247.                    pixel_value = 0.0;
248.                    for (j = -1; j <= 1; j++) {
249.                          for (i = -1; i <= 1; i++) {
250.                                pixel_value += vertKernel[j + 1][i + 1] *
  inData[y + j][x + i];
251.                          }
252.                    }
253.
254.                    temp1[y - 1][x - 1] = pixel_value;
255.              }
256.        }
257.
258.        // Horizontal Sobel Mask
259.        for (y = 1; y < ysize - 1; y++)
260.        {
261.              for (x = 1; x < xsize - 1; x++)
262.              {
263.                    pixel_value = 0.0;
264.                    for (j = -1; j <= 1; j++) {
265.                          for (i = -1; i <= 1; i++) {
266.                                pixel_value += horzKernel[j + 1][i + 1] *
  inData[y + j][x + i];
267.                          }
268.                    }
269.                    temp2[y - 1][x - 1] = pixel_value;
270.              }
271.        }
272.
273.        // Take magnitude
274.        for (y = 0; y < ysize - 3; y++)
275.        {
276.              for (x = 0; x < xsize - 3; x++)
277.              {
278.                    outData[y][x] = sqrt(pow(temp1[y][x], 2) + pow(temp2[y][x],
  2));
279.                    // Get max for normalizing
280.                    if (outData[y][x] > max) max = outData[y][x];
281.              }
282.        }
283.        // Normalize
284.        for (y = 0; y < ysize - 3; y++)
285.        {
286.              for (x = 0; x < xsize - 3; x++)
287.              {
288.                    outData[y][x] = 255 * outData[y][x] / max;
289.              }
290.        }
291.
292.        free(temp1);
293.        free(temp2);
294.  }
295.
296.  /*-------------------- BinThresh()-------------------------------------
297.
298.  PURPOSE
299.  Takes binary threshold of image
300.
```

```
INPUT PARAMETERS
-inData:    old 2D array to be input
-outData:   new 2D array to be input
-xsize:           size of array in x direction
-ysize:           size of array in y direction
-thresh:    threshold value
-------------------------------------------------------------------*/
void BinThresh(float **inData, float **outData, int xsize, int ysize, int thresh)
{
      int x, y;
      for (y = 1; y < ysize - 1; y++)
      {
            for (x = 1; x < xsize - 1; x++)
            {
                  if (inData[y][x] > thresh)
                        outData[y][x] = 255;
                  else
                        outData[y][x] = 0;
            }
      }
}

/*-------------------- AddUniNoise()-----------------------------------

PURPOSE
Adds uniform noise to image

INPUT PARAMETERS
-inData:    old 2D array to be input
-outData:   new 2D array to be input
-xsize:           size of array in x direction
-ysize:           size of array in y direction
-intens:    intensity of noise
-------------------------------------------------------------------*/
void AddUniNoise(float** inData, float **outData, int xsize, int ysize, int intens)
{
      int r;
      int i = intens;
      float noise, k;
      int x, y;
      for (y = 0; y < ysize; y++)
      {
            for (x = 0; x < xsize; x++)
            {
                  r = rand();
                  k = (2 * r - RAND_MAX);
                  k /= RAND_MAX;
                  noise = k*i;
                  outData[y][x] = inData[y][x] + noise;
            }
      }
}

/*-------------------- GaussianSmooth()-------------------------------

PURPOSE
Applies gaussian smoothing operator to image

INPUT PARAMETERS
-inData:    old 2D array to be input
-outData:   new 2D array to be input
-xsize:           size of array in x direction
-ysize:           size of array in y direction
-------------------------------------------------------------------*/
void GaussianSmooth(float **inData, float **outData, int xsize, int ysize)
```

```
366.  {
367.        float gaussKernel[3][3] = {   { 1.0/16, 1.0/8, 1.0/16 },
368.                                      { 1.0/8 , 1.0/4, 1.0/8  },
369.                                      { 1.0/16, 1.0/8, 1.0/16 } };
370.
371.        int x, y, i, j;
372.        double pixel_value;
373.        double max, min;
374.
375.        min = DBL_MAX;
376.        max = -DBL_MAX;
377.        for (y = 1; y < ysize - 1; y++)
378.        {
379.              for (x = 1; x < xsize - 1; x++)
380.              {
381.                    pixel_value = 0.0;
382.                    for (j = -1; j <= 1; j++) {
383.                          for (i = -1; i <= 1; i++) {
384.                                pixel_value += gaussKernel[j + 1][i + 1] *
  inData[y + j][x + i];
385.                          }
386.                    }
387.
388.                    outData[y - 1][x - 1] += pixel_value;
389.              }
390.        }
```

```
1. /*--------------- Circles.h ---------------
2.
3. by:   Sean Webster
4.
5. PURPOSE
6. function prototypes for Circles.c
7.
8. ----------------------------------------------*/
9.
10.   #include <math.h>
11.
12.   #ifndef CIRCLES_H
13.   #define CIRCLES_H
14.
15.   void CreateCircles(float **data, int height, int width, int xloc, int yloc,
  int intensity, int rad);
16.   #endif
```

```
17.   /*--------------- Circles.c ---------------
18.   PURPOSE: Creates Circles
19.
20.   by:         Sean Webster
21.               Computer Vision
22.               Due 4/4/2016
23.   ----------------------------------------------*/
24.
25.   #include "Circles.h"
26.   /*-------------------- CreateCircle()-----------------------------------
27.
28.   PURPOSE
29.   Creates a circle in image
30.
31.   INPUT PARAMETERS
32.   -inData:    old 2D array to be input
```

```
33.   -outData:   new 2D array to be input
34.   -xsize:           size of array in x direction
35.   -ysize:           size of array in y direction
36.   -xloc:            x location of circle
37.   -yloc:            y location of circle
38.   -intensity: intensity of circle
39.   -radius:    radius of circle
40.   ----------------------------------------------------------------------*/
41.   void CreateCircle(float **data, int ysize, int xsize, int xloc, int yloc,
  int intensity, int rad)
42.   {
43.         int x, y;
44.         for (y = 0; y < ysize; y++)
45.         {
46.               for (x = 0; x < xsize; x++)
47.               {
48.                     if (pow(x - xloc, 2) + pow(y - yloc, 2) - pow(rad, 2) <= 0)
49.                           data[y][x] += intensity;
50.               }
51.         }
52.   }
```

```
53.   /*--------------- arrays.h ---------------
54.
55.   by:   Sean Webster
56.
57.   PURPOSE
58.   function prototypes for arrays.c
59.
60.   ----------------------------------------------*/
61.
62.   #include <math.h>
63.   #include <stdlib.h>
64.   #include <stdio.h>
65.   #include <string.h>
66.
67.   #ifndef ARRAYS_H
68.   #define ARRAYS_H
69.
70.   float **ImageInit(int xsize, int ysize);
71.   int OpenImage(char *fn, int xsize, int ysize, float **data);
72.   void SaveImage(char *fn, char* out, int xsize, int ysize, float **data_ptr);
73.   void SaveImageAsText(float *fn, char* out, int xsize, int ysize, float
  **data_ptr);
74.   void AccuracyReport(float **image1, float **image2, int xsize, int ysize);
75.   #endif
```

```
76.   /*--------------- arrays.c ---------------
77.
```

```
78.   by:   Sean Webster
79.
80.   PURPOSE
81.   Array based functions for inputting images
82.
83.   ------------------------------------------*/
84.
85.   #include "arrays.h"
86.
87.
88.   /*-------------------- ImageInit()-------------------------------------
89.
90.   PURPOSE
91.   Initializes image float array in memory and to 0
92.
93.   INPUT PARAMETERS
94.   -xsize:     size of array in x direction
95.   -ysize: size of array in y direction
96.   ------------------------------------------------------------------------*/
97.   float **ImageInit(int xsize, int ysize)
98.   {
99.         float **data;
100.        int i, j;
101.        data = (float **)malloc(sizeof(float *) * ysize);
102.        for (i = 0; i < ysize; i++)
103.        {
104.              data[i] = (float *)malloc(sizeof(float) * xsize);
105.        }
106.        for (i = 0; i < ysize; i++)
107.        {
108.              for (j = 0; j < xsize; j++)
109.              {
110.                    data[i][j] = 0;
111.              }
112.        }
113.        return data;
114.  }
115.
116.
117.
118.
119.  /*-------------------- OpenImage()-------------------------------------
120.
121.  PURPOSE
122.  Reads a raw image file into a 2D array
123.
124.  INPUT PARAMETERS
125.  -data:      2D array to be input
126.  -xsize:     size of array in x direction
127.  -ysize: size of array in y direction
128.  -fn:  name of input image
129.  ------------------------------------------------------------------------*/
130.  int OpenImage(char *fn, int xsize, int ysize, float **data)
131.  {
132.        int j;
133.        int   k;
134.        FILE *fp_inp;
135.        unsigned char pixel;
136.        // Read image contents into 2D array
137.        if ((fp_inp = fopen(fn, "rb")) == NULL) return -1;
138.        for (j = 0; j < ysize; j++)
139.        {
140.              for (k = 0; k < xsize; k++)
141.              {
142.                    fread(&pixel, sizeof(char), 1, fp_inp);
143.                    data[j][k] = (float)pixel;
144.              }
```

```
145.        }
146.
147.        (void)fclose(fp_inp);
148.        return 0;
149.  }
150.
151.  /*-------------------- SaveImage()-----------------------------------
152.
153.  PURPOSE
154.  Writes a 2D array to a raw image file
155.
156.  INPUT PARAMETERS
157.  -data:      2D array to be input
158.  -xsize:     size of array in x direction
159.  -ysize: size of array in y direction
160.  -fn:  name of input image
161.  -out: string to append to output image
162.  ----------------------------------------------------------------------*/
163.  void SaveImage(float *fn, char* out, int xsize, int ysize, float **data_ptr)
164.  {
165.        int j, k;
166.        FILE *out_file;
167.        char out_name[40];
168.        char name[15];
169.        unsigned char pixel;
170.        strcpy(name, fn);
171.        strcpy(out_name, name);
172.        strcat(out_name, out);
173.        out_file = fopen(out_name, "wb");
174.        for (j = 0; j < ysize; j++)
175.        {
176.              for (k = 0; k < xsize; k++)
177.              {
178.                    pixel = (unsigned char)data_ptr[j][k];
179.                    fwrite(&pixel, sizeof(char), 1, out_file);
180.              }
181.        }
182.        (void)fclose(out_file);
183.  }
184.
185.  void SaveImageAsText(float *fn, char* out, int xsize, int ysize, float
    **data_ptr)
186.  {
187.        int j, k;
188.        FILE *out_file;
189.        char out_name[20];
190.        char name[15];
191.        unsigned char pixel;
192.        strcpy(name, fn);
193.        strcpy(out_name, name);
194.        strcat(out_name, out);
195.        strcat(out_name, ".txt");
196.        out_file = fopen(out_name, "wb");
197.        for (j = 0; j < ysize - 1; j++)
198.        {
199.              for (k = 0; k < xsize - 1; k++)
200.              {
201.                    pixel = (unsigned char)data_ptr[j][k];
202.                    fwrite(&pixel, sizeof(char), 1, out_file);
203.              }
204.              fwrite("\n", sizeof(char), 1, out_file);
205.        }
206.        (void)fclose(out_file);
207.  }
208.
209.  /*-------------------- SaveImage()-----------------------------------
210.
```

```
211.  PURPOSE
212.  Writes to console report of accuracy
213.
214.  INPUT PARAMETERS
215.  -image1:    2D image array to be compared
216.  -image2:    2D image array to be compared
217.  -xsize:           size of array in x direction
218.  -ysize:           size of array in y direction
219.  ------------------------------------------------------------------------*/
220.  void AccuracyReport(float **image1, float **image2, int xsize, int ysize)
221.  {
222.        int x, y;
223.        double m = xsize*ysize;
224.        float accuracy = 0;
225.        for (y = 0; y < ysize - 1; y++)
226.        {
227.              for (x = 0; x < xsize - 1; x++)
228.              {
229.                    if (image1[y][x] == image2[y][x]) accuracy++;
230.              }
231.        }
232.        printf("Image2 contains %lf of the same pixels as Image2\n", accuracy
  / m);
233.  }
```