

画像工学 レポート

画像の平滑化とエッジ抽出

IE5 (9) 片岡 駿之介

2016年11月14日

1 課題

画像処理の中には、画像内に含まれる雑音を除去したり、画像の持つ特徴を抽出したりといった目的で、画像に対してフィルタリングと呼ばれる処理を行うことがある。本課題では、フィルタリングの中でも「平滑化」と「エッジ抽出」に着目している。「平滑化」は、主に画像中に含まれているインパルスノイズを取り除く目的で使用される。「エッジ抽出」は、画像中の特徴を知りたい時に画像中のエッジのみを取り出す目的で使用される。

今回、課題において「平滑化」については「加重平均フィルタ」と「メディアンフィルタ」、「エッジ抽出」に関しては「ラプラシアンフィルタ」を実装し、それぞれの出力結果を確かめる。

2 処理の流れ

今回の演習では、PGM ファイルを以下の手順で処理していくこととした。

1. 入力されたコマンドが正しいかチェック（正しくない場合は Usage を表示して終了）
2. コマンドで指定された PGM ファイルを読み込む
3. 読み込んだ PGM ファイルのヘッダ解析
4. ヘッダが正しければ画素値を 2 次元配列に取り込む
5. コマンドで指定されたフィルタリング処理を施す
6. コマンドで指定されたファイルへ画像を出力する

この処理によって得られた PGM ファイルを ImageMagick を用いて表示し、処理前後で画像がどのように変化するかを比較検討する。

なお、縁に関しては元の画像の画素値を代入している。

3 ソースリスト

ソースコード 1 filter.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <ctype.h>
5
6 FILE *fp;
7
8 int width;
9 int height;
10 int max_value;
11 int status;
12 char buffer[128];
13
14 void image_open(char *file_name) {
15     if ((fp = fopen(file_name, "r")) == NULL) {
16         printf("file_open_error!!\n");
17         exit(-1); /* (3) エラーの場合は通常、異常終了する */
18     }
19 }
20
21 int main(int argc,char *argv[]) {
22
23     /* 入力コマンドのチェック */
24     if(argc != 5) {
25         printf("Usage:./filter<filter_type><kernel_size><input_pgm_filename><output_pgm_filename>\n");
26         exit(0);
27     } else if (argc == 5) {
28         image_open(argv[3]);
29         printf("filter_type:%s\n", argv[1]);
30         printf("kernel_size:%s\n", argv[2]);
31     }
32
33     /* ヘッダ取得部 */
34     int ch;
35
36     while (status < 3) {
37         ch = getc(fp);
38
39         if(ch == '#') { // コメントのスキップ
40             while ((ch = getc(fp)) != '\n')
41                 break;
42         }
43
44         if(ch == 'P') { // マジックナンバーのチェック
45             if(getc(fp) != '5') {
46                 printf("Magic_Number is wrong.\n");
47                 break;
48             } else {
49                 // printf("Magic Number is P5\n");
50             }
51         }
52     }
```

```

53     if(isdigit((unsigned char)ch)) { // 数値取得部
54         buffer[0] = ch;
55         int i=1;
56         while(1) {
57             char c = getc(fp);
58             if(isdigit((unsigned char)c)) {
59                 buffer[i]=c;
60                 i++;
61             } else
62                 break;
63         }
64         buffer[i] = '\0';
65
66         switch (status) {
67             case 0: // width 取得部
68                 width = atoi(buffer);
69                 printf("width=%d\n", width);
70                 break;
71             case 1: // height 取得部
72                 height = atoi(buffer);
73                 printf("height=%d\n", height);
74                 break;
75             case 2: // max_value 取得部
76                 max_value = atoi(buffer);
77                 printf("max_value=%d\n", max_value);
78                 break;
79         }
80         status++; // 次のステータスへ
81     }
82 }
83
84 /* 画素値取得部 */
85 int image[width][height];
86
87 for(int i = 0; i < height; i++) {
88     for(int j = 0; j < width; j++) {
89         image[j][i] = (int)getc(fp);
90     }
91 }
92
93 /* フィルタ処理部 */
94
95 int kernel_size = atoi(argv[2]);
96 int output_image[width][height];
97
98 for(int i=0; i < height; i++) {
99     for(int j=0; j < width; j++) {
100         output_image[j][i] = image[j][i];
101     }
102 }
103
104 if(strcmp(argv[1], "average") == 0) { //加重平均フィルタ
105
106     int padding = ((kernel_size - 1)/2);
107     printf("padding:%d\n", padding);
108
109     int i, j;

```

```

110
111     for(i=padding; i < (height-padding); i++) {
112         for(j=padding; j < (width-padding); j++) {
113             int sum = 0;
114
115             for(int k = -padding; k <= padding; k++) {
116                 for(int l = -padding; l <= padding; l++) {
117                     sum += image[j+k][i+l];
118                 }
119             }
120             output_image[j][i] = (int)(sum / (kernel_size * kernel_size));
121         }
122     }
123
124 } else if(strcmp(argv[1], "median") == 0) { //メディアンフィルタ
125     int padding = ((kernel_size - 1)/2);
126     printf("padding:%d\n", padding);
127
128     int i, j;
129
130     for(i=padding; i < (height-padding); i++) {
131         for(j=padding; j < (width-padding); j++) {
132             int sum = 0;
133             int median_array[kernel_size * kernel_size];
134             int median_cnt = 0;
135
136             for(int m = 0; m < kernel_size * kernel_size; m++)
137                 median_array[m] = 0;
138
139             for(int k = -padding; k <= padding; k++) {
140                 for(int l = -padding; l <= padding; l++) {
141                     median_array[median_cnt] = image[j+k][i+l];
142                     median_cnt++;
143                 }
144             }
145
146             for(int m = 0; m < (kernel_size * kernel_size); m++) {
147                 for(int n = 0; n < (kernel_size * kernel_size); n++) {
148                     if (median_array[n+1] < median_array[n]) {
149                         int buf = median_array[n+1];
150                         median_array[n+1] = median_array[n];
151                         median_array[n] = buf;
152                     }
153                 }
154             }
155
156             output_image[j][i] = (int)median_array[(kernel_size * kernel_size) / 2];
157         }
158     }
159 } else if(strcmp(argv[1], "laplacian") == 0) { //ラプラシアンフィルタ
160     if(kernel_size != 3) {
161         printf("ラプラシアンフィルタを適用する場合はカーネルサイズを3にしてください\n");
162         exit(0);
163     }
164
165     int padding = ((kernel_size - 1)/2);
166     printf("padding:%d\n", padding);

```

```
167     int laplacian[3][3] = {{1, 1, 1}, {1, -8, 1}, {1, 1, 1}};
168     int i, j;
169
170     for(i=padding; i < (height-padding); i++) {
171         for(j=padding; j < (width-padding); j++) {
172             int sum = 0;
173
174             for(int k = -padding; k <= padding; k++) {
175                 for(int l = -padding; l <= padding; l++) {
176                     sum += (image[j+k][i+l] * laplacian[k+padding][l+padding]);
177                 }
178             }
179             if(sum < 0)
180                 output_image[j][i] = 0;
181             else if(sum > 255)
182                 output_image[j][i] = 255;
183             else
184                 output_image[j][i] = (int)sum;
185         }
186     }
187 }
188
189 /* 画像出力部 */
190
191 FILE *output = fopen(argv[4], "wb");
192 char header[30];
193 sprintf(header, "P5%d%d%d\n", width, height, max_value);
194 fputs(header, output);
195 for(int i = 0; i < height; i++) {
196     for(int j = 0; j < width; j++) {
197         fputc((char)output_image[j][i], output);
198     }
199 }
200 printf("image was output.\n");
201 fclose(output);
202 }
```

4 結果

結果の確認には、配布された「town.pgm」を用いた。

以下に、それぞれのフィルタを適用した場合の適用前後の比較画像を示す。



図 1 適用前



図 2 適用後

図 3 加重平均フィルタを適用（カーネルサイズ：3）



図 4 適用前



図 5 適用後

図 6 加重平均フィルタを適用（カーネルサイズ：9）



図 7 適用前



図 8 適用後

図 9 メディアンフィルタを適用（カーネルサイズ：3）



図 10 適用前



図 11 適用後

図 12 メディアンフィルタを適用（カーネルサイズ：9）



図 13 適用前

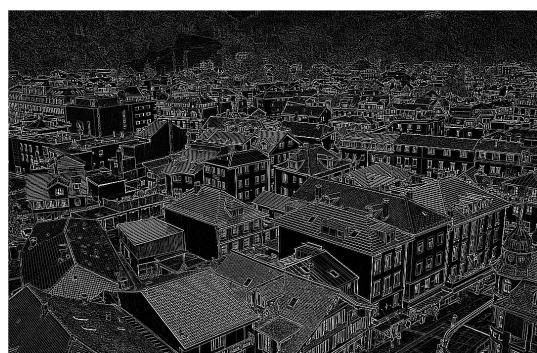


図 14 適用後

図 15 ラプラシアンフィルタを適用（カーネルサイズ：3）

また、今回はノイズ除去の効果を評価するために、メディアンフィルタを用いて、それぞれカーネルサイズが3と9でのノイズ除去の効果を比較した。ノイズ除去効果の確認には、Photoshopにてノイズを10%載せた「town_noise.pgm」を使用した。



図 16 適用前



図 17 適用後

図 18 ノイズ重畠画像に対してメディアンフィルタを適用（カーネルサイズ：3）



図 19 適用前



図 20 適用後

図 21 ノイズ重畠画像に対してメディアンフィルタを適用（カーネルサイズ：9）

5 考察

加重平均フィルタでは、画像全体が均一にボケているのが確認できる。また、カーネルサイズを大きくすると、より画像がボケていることも確認できる。

メディアンフィルタでは、同じカーネルサイズでも加重平均フィルタに比べてエッジが保存されつつ、画像がなめらかになっていることがわかる。

ラプラシアンフィルタでは、画像中のエッジが期待通り抽出できていることが確認できる。

ノイズ除去効果については、カーネルサイズ：3のときはあまり効果が得られなかったが、カーネルサイズ：9のときはかなりノイズが除去されていることがわかるが、画像自体もかなりボケてしまっている。ノイズ除去に関しては、やはりガウシアンフィルタやバイラテラルフィルタといった、より高度なフィルタを使用

することが望まれる。

6 感想

今回、一番躊躇ったところは、pgm ファイルを先頭から読み取っていく際に、配列を縦横逆に取り込んでしまったところである。一般的な間隔でコードを書くと、pgm ファイルから読み取った画素値が先頭から順に縦方向に配列に格納されてしまい、後にフィルタリング処理を行う際に不都合が生じる。このバグに 5 時間ぐらい悩まされたのが悔しいところである。