

E. E. Experiment (computer vision)

조현중 201410935

담당교수 김원준

2019 09 30

1 Abstract

앞서 사용한 HOG weight과 Harris edge detecting을 사용하여, 두 이미지의 엣지를 비교하고 가장 비슷한 엣지를 선으로 잇는 프로그램을 작성한다. 이미지에 Harris edge detecting을 적용하여 엣지를 확인한뒤, 두 이미지의 엣지의 HOG weight을 비교하는 방식으로 구현한다.

2 Experiment

2.1 Source code

소스파일은 헤더 파일과 메인 파일로 나뉘, 간단한 메인소스를 두고, 헤더에서 함수와 그와 관련된 상수를 정의 하였다.

2.1.1 main.cpp

```
1  #include <opencv2/imgproc.hpp>
2  #include <opencv2/highgui.hpp>
3  #include <math.h>
4  #include <iostream>
5  #include <stdlib.h>
6
7  //header
8  #include "visionFunctions.hpp"
9
10
11 int main() {
12     //open img
13     int height, width;
14
15     //calc vision weight of ref Img.
16     Mat refImg = imread("ref.bmp", cv::IMREAD_COLOR);
17     height = refImg.rows;
18     width = refImg.cols;
19     pixel * visionWeightRef = setVision(refImg);
20
21     //calc vision weight of compare Img.
22     Mat compImg = imread("tarShift_1.bmp", cv::IMREAD_COLOR);
23     height = compImg.rows;
24     width = compImg.cols;
25     pixel * visionWeightComp = setVision(compImg);
26
27     findSameEdge(refImg, compImg, visionWeightRef, visionWeightComp);
28     waitKey(5000);
29 }
```

2.1.2 visionFunction.hpp

```
1  #pragma once
2  #pragma warning(disable : 4996)
```

```

3 //define for debug
4 #define DEBUG_
5 //if define SVAEIMG, all filtered IMG be save
6 #define SAVEIMG
7
8 #define EXPENDEGE 0 //TODO
9 #define ZEROPADDING 1 //TODO
10 #define CHANNEL 3 //RGB
11
12 using namespace cv;
13
14
15 //-----
16 //define constant
17
18
19 //Gradient filter size
20 //TODO
21 #define FILTER_H 3
22 #define FILTER_W 3
23
24 //gaussian filter
25 #define SIGMA_GAU 1
26 #define FILTER_GAU_SIZE 3
27
28
29 #define HOG_SIZE 17
30
31 //Harris edge detect
32 //edge detect Window size
33 #define WINDOW_SIZE 5
34 //edge detect THRESHOLD 0~1024
35 #define THRESHOLD 100
36 float CONT_k = 0.04;
37
38 //JUST PI
39 #define PI 3.1415
40 //count variable for filesave
41 int count = 0;
42
43 //-----
44 //struct definition
45 struct pixel {
46     //gradient y
47     float H;
48     //gradient x
49     float W;
50     //raw edge weight or true/false
51     float edge;
52     //0 <= x < 180
53     float phase;
54     //0 to 255
55     float magnitude;
56     float hog[9];
57 } typedef pixel;
58
59 //gradient filter definition
60 int filterX[9] = {
61     -1, -1, -1,
62     0, 0, 0,
63     1, 1, 1
64 };
65 int filterY[9] = {
66     -1, 0, 1,
67     -1, 0, 1,
68     -1, 0, 1
69 };
70
71 //-----
72 //vision functions
73 void harris(pixel * output, int height, int width);
74 void gaussian(Mat * inoutImg, int sigma, int sizeFilter);
75 void pointCircling(Mat * inoutImg, int y, int x, int sizeRadius, uchar b, uchar g, uchar r);
76 void calcGredient(Mat * inputImg, pixel * output);
77 void hog(pixel * output, int y, int x, int tileSize, int height, int width);
78 pixel * setVision(Mat origImg);
79 void findSameEdge(Mat refImg, Mat compImg, pixel * visionRef, pixel * visionComp);
80 int expendEdge(int x, int max);
81
82
83 int expendEdge(int x, int max) {
84     if (x < 0) {
85         x = 0;
86     }

```

```

87     else if (x >= max) {
88         x = max - 1;
89     }
90     return x;
91 }
92
93
94 //vision main function
95 pixel * setVision(Mat origImg) {
96     std::cout << "set #" << count << " photo vision weight" << std::endl;
97     int height = origImg.rows;
98     int width = origImg.cols;
99     pixel * output = (pixel*)calloc(height * width, sizeof(pixel));
100
101     //define iteration variable
102     int i, j, h, w;
103
104     Mat origImgGray = Mat::zeros(height, width, CV_8UC1);
105
106     //low pass filter
107     std::cout << "- lowpass filtering" << std::endl;
108     gaussian(&origImg, SIGMA_GAU, FILTER_GAU_SIZE);
109
110     //conv. color to gray
111     for (i = 0; i < height; i++) {
112         //X
113         for (j = 0; j < width; j++) {
114             origImgGray.at<uchar>(i, j) = (origImg.at<Vec3b>(i, j)[0] + origImg.at<Vec3b>(i, j)[1] +
115                 origImg.at<Vec3b>(i, j)[2]) / 3;
116         }
117     }
118
119     //gradient
120     std::cout << "- calculate gradient and phase" << std::endl;
121     calcGredient(&origImgGray, output);
122
123     //harris edge detect
124     std::cout << "- harris edge detecting" << std::endl;
125     harris(output, height, width);
126
127     //get edge's HOG weight
128     std::cout << "- calculate HOG weight" << std::endl;
129     for (i = 0; i < height; i++) {
130         for (j = 0; j < width; j++) {
131             if (output[i * width + j].edge > 0) {
132                 hog(output, i, j, HOG_SIZE, height, width);
133             }
134         }
135     }
136
137     std::cout << "- edge circling" << std::endl;
138     int a = 0;
139     for (i = 0; i < height; i++) {
140         for (j = 0; j < width; j++) {
141             if (output[i * width + j].edge > 0) {
142                 pointCircling(&origImg, i, j, 2, 0, 255, 255);
143             }
144         }
145     }
146     count++;
147     return output;
148 }
149
150 void findSameEdge(Mat refImg, Mat compImg, pixel * visionRef, pixel * visionComp) {
151     std::cout << "Detect same edge" << std::endl;
152
153     Mat concatImg;
154     hconcat(refImg, compImg, concatImg);
155
156     int height = refImg.rows;
157     int width = refImg.cols;
158     float min = INT_MAX;
159
160     //define iteration variable
161     int refi, refj, compi, compj, k;
162     int compx = 0, compy = 0;
163     float subWeight = 0;
164
165     for (refi = 0; refi < height; refi++) {
166         for (refj = 0; refj < width; refj++) {
167             if (visionRef[refi * width + refj].edge > 0) {
168                 for (compi = 0; compi < height; compi++) {
169                     for (compj = 0; compj < width; compj++) {

```

```

170         if (visionComp[comp_i * width + comp_j].edge > 0) {
171             for (k = 0; k < 9; k++) {
172                 subWeight += fabs(visionRef[ref_i * width + ref_j].hog[k] -
173                     visionComp[comp_i * width + comp_j].hog[k]);
174             }
175             if (min > subWeight) {
176                 min = subWeight;
177                 comp_x = comp_j;
178                 comp_y = comp_i;
179             }
180             subWeight = 0;
181         }
182     }
183 }
184 line(concatImg, Point(ref_j, ref_i), Point(comp_x + width, comp_y), Scalar(255, 0, 0), 2, 9,
185     ↪ 0);
186
187     min = INT_MAX;
188 }
189 }
190 }
191 //line(concatImg, Point(52, 52), Point(264, 23), Scalar(255, 0, 0), 2, 9, 0);
192
193 imshow("concat", concatImg);
194 #ifdef SAVEIMG
195 imwrite("R_sameEdge" + std::to_string(count) + ".bmp", concatImg);
196 #endif //SAVEIMG
197
198 waitKey(5000);
199 }
200
201 //input grayscale Img and pixel struct's pointer(size must be same to input img)
202 //each pixel's gradient, quantized radian and normalized magnitude(0 to 255) will be save at pixel struct
203 void calcGredient(Mat * inputImg, pixel* output) {
204     int height = inputImg->rows;
205     int width = inputImg->cols;
206
207     int max = INT_MIN + 1, min = INT_MAX;
208     int i, j, h, w;
209
210     for (i = 0; i < height; i++) {
211         for (j = 0; j < width; j++) {
212
213             //convolution calculation
214             for (h = 0; h < FILTER_H; h++) {
215                 for (w = 0; w < FILTER_W; w++) {
216                     //countinue when outbound
217                     //if ((i - 1 + h < 0) || (i - 1 + h >= height)) continue;
218                     //if ((j - 1 + w < 0) || (j - 1 + w >= width)) continue;
219
220                     //calc MAC gradient filter
221                     output[i * width + j].W += (inputImg->at<uchar>(i - 1 + h, j - 1 + w)) * (filterX[h *
222                         ↪ FILTER_W + w]);
223                     output[i * width + j].H += (inputImg->at<uchar>(i - 1 + h, j - 1 + w)) * (filterY[h *
224                         ↪ FILTER_W + w]);
225                     output[i * width + j].W += (inputImg->at<uchar>(expedEdge(i - 1 + h, height),
226                         ↪ expedEdge(j - 1 + w, width))) * (filterX[h * FILTER_W + w]);
227                     output[i * width + j].H += (inputImg->at<uchar>(expedEdge(i - 1 + h, height),
228                         ↪ expedEdge(j - 1 + w, width))) * (filterY[h * FILTER_W + w]);
229                 }
230             }
231
232             //conv radian to degree 0 ~ 179
233             output[i * width + j].phase = 180 * (atan2(output[i * width + j].H, output[i * width + j].W) /
234                 ↪ PI);
235             if (output[i * width + j].phase < 0) {
236                 output[i * width + j].phase = output[i * width + j].phase + 180;
237             }
238             if (output[i * width + j].phase >= 180) {
239                 output[i * width + j].phase = 0;
240             }
241
242             //calc magnitude
243             output[i * width + j].magnitude = sqrt(pow(output[i * width + j].W, 2) + pow(output[i * width +
244                 ↪ j].H, 2));
245
246             //set max, min for normalize
247             max = __max(output[i * width + j].magnitude, max);
248             min = __min(output[i * width + j].magnitude, min);
249         }
250     }
251 }

```

```

246 //magnitude normalize
247 for (i = 0; i < height; i++) {
248     for (j = 0; j < width; j++) {
249         output[i * width + j].magnitude = 255 * (output[i * width + j].magnitude - min) / (max - min);
250         //outputImg.at<uchar>(i, j) = output[i * width + j].magnitude;
251     }
252 }
253 }
254
255
256 void hog(pixel* output, int y, int x, int tileSize, int height, int width) {
257     int tileW, tileH, blkI, blkJ, sumBlk = 0;
258     tileW = tileH = tileSize;
259     int i, j, h, w;
260     int startX = x - (int)(tileW / 2), startY = y - (int)(tileH / 2);
261     float quantPhase = 0, phaseWeight = 0;
262
263     for (h = 0; h < tileH; h++) {
264         for (w = 0; w < tileW; w++) {
265             //continue when outbound
266             if ((h + startY < 0) || (h + startY >= height)) continue;
267             if ((w + startX < 0) || (w + startX >= width)) continue;
268             //calc hog weight
269             quantPhase = output[(h + startY) * width + (w + startX)].phase / 20;
270             phaseWeight = quantPhase - (int)quantPhase;
271             output[y * width + x].hog[(int)quantPhase] +=
272                 (1 - phaseWeight) * output[(h + startY) * width + (w + startX)].magnitude;
273             output[y * width + x].hog[(int)(quantPhase + 1 >= 9 ? 0 : quantPhase + 1)] +=
274                 (phaseWeight) * output[(h + startY) * width + (w + startX)].magnitude;
275         }
276     }
277
278     //sum of all block pixel's magnitude
279     for (int k = 0; k < 9; k++) {
280         sumBlk += pow(output[y * width + x].hog[k], 2) + 0.000001;
281     }
282
283     //L-2 normalize weight
284     for (int k = 0; k < 9; k++) {
285         output[y * width + x].hog[k] = output[y * width + x].hog[k] / sqrt(sumBlk);
286     }
287 }
288
289 //size of filter must odd number
290 void gaussian(Mat * inoutImg, int sigma, int sizeFilter) {
291     //iteration definition
292     int i, j, h, w;
293
294     int filterHalf = (int)(sizeFilter / 2);
295     int height = inoutImg->rows;
296     int width = inoutImg->cols;
297
298     float* gaussianFilter = (float*)calloc(sizeFilter * sizeFilter, sizeof(float));
299     float sum = 0;
300     Mat tmp = Mat::zeros(height, width, CV_8UC3);
301
302     for (h = 0; h < sizeFilter; h++) {
303         for (w = 0; w < sizeFilter; w++) {
304             gaussianFilter[h * sizeFilter + w] = pow(h - filterHalf, 2) + pow(w - filterHalf, 2);
305             gaussianFilter[h * sizeFilter + w] = (1 / (pow(sigma, 2) * 2 * PI)) * exp((-1) *
306                 ↪ (gaussianFilter[h * sizeFilter + w] / (2 * sigma * sigma)));
307             sum += gaussianFilter[h * sizeFilter + w];
308         }
309     }
310
311     sum = 1 / sum;
312
313     for (i = 0; i < height; i++) {
314         for (j = 0; j < width; j++) {
315             //convolution calculation
316             for (h = 0; h < sizeFilter; h++) {
317                 for (w = 0; w < sizeFilter; w++) {
318                     //continue when outbound
319                     if ((i - filterHalf + h < 0) || (i - filterHalf + h >= height)) continue;
320                     if ((j - filterHalf + w < 0) || (j - filterHalf + w >= width)) continue;
321                     //calc MAC gaussian filter and normalize
322                     tmp.at<Vec3b>(i * width + j)[0] += (float)(inoutImg->at<Vec3b>(i - filterHalf + h, j -
323                         ↪ filterHalf + w)[0]) * sum * (gaussianFilter[h * sizeFilter + w]);
324                     tmp.at<Vec3b>(i * width + j)[1] += (float)(inoutImg->at<Vec3b>(i - filterHalf + h, j -
325                         ↪ filterHalf + w)[1]) * sum * (gaussianFilter[h * sizeFilter + w]);
326                     tmp.at<Vec3b>(i * width + j)[2] += (float)(inoutImg->at<Vec3b>(i - filterHalf + h, j -
327                         ↪ filterHalf + w)[2]) * sum * (gaussianFilter[h * sizeFilter + w]);

```

```

325     }
326     }
327 }
328
329
330 for (i = 0; i < height; i++) {
331     for (j = 0; j < width; j++) {
332         inoutImg->at<Vec3b>(i, j)[0] = tmp.at<Vec3b>(i * width + j)[0];
333         inoutImg->at<Vec3b>(i, j)[1] = tmp.at<Vec3b>(i * width + j)[1];
334         inoutImg->at<Vec3b>(i, j)[2] = tmp.at<Vec3b>(i * width + j)[2];
335     }
336 }
337
338 #ifdef SAVEIMG
339     imwrite("R_gaussian"+std::to_string(count)+".bmp", *inoutImg);
340 #endif //SAVEIMG
341 }
342
343 //input Img and x, y then the pixel will be circled
344 void pointCircling(Mat * inoutImg, int y, int x, int sizeRadius, uchar b, uchar g, uchar r) {
345     Scalar c;
346     Point pCenter;
347     pCenter.x = x;
348     pCenter.y = y;
349     c.val[0] = b;
350     c.val[1] = g;
351     c.val[2] = r;
352     circle(*inoutImg, pCenter, sizeRadius, c, 2, 8, 0);
353 #ifdef SAVEIMG
354     imwrite("R_edgeCircled" + std::to_string(count) + ".bmp", *inoutImg);
355 #endif //SAVEIMG
356 }
357
358 //input pixel struct then edge detect from it's gradient
359 void harris(pixel* output, int height, int width) {
360     int i, j, h, w;
361     float max = INT_MIN;
362     float det = 0, tr = 0;
363     float MatA = 0, MatB = 0, MatC = 0, MatD = 0;
364     int convHalf = (int)(WINDOW_SIZE / 2);
365     for (i = 0; i < height; i++) {
366         //X
367         for (j = 0; j < width; j++) {
368             //convolution calculation
369             for (h = 0; h < WINDOW_SIZE; h++) {
370                 for (w = 0; w < WINDOW_SIZE; w++) {
371                     //continue when outboud
372                     if ((i - convHalf + h < 0) || (i - convHalf + h >= height)) continue;
373                     if ((j - convHalf + w < 0) || (j - convHalf + w >= width)) continue;
374                     //calc matrix
375                     MatA += pow(output[(i - convHalf + h) * width + j - convHalf + w].W, 2);
376                     MatB += output[(i - convHalf + h) * width + j - convHalf + w].W * output[(i - convHalf
377                     ↵ + h) * width + j - convHalf + w].H;
378                     MatD += pow(output[(i - convHalf + h) * width + j - convHalf + w].H, 2);
379                 }
380             }
381             det = MatA * MatD - pow(MatB, 2);
382             tr = (MatA + MatD) * (MatA + MatD) * CONT_k;
383
384             //raw data
385             output[i * width + j].edge = det - tr;
386
387             max = __max(max, (det - tr));
388             //reset
389             MatA = MatB = MatC = MatD = 0;
390         }
391     }
392     //X
393     for (j = 0; j < width; j++) {
394         #ifdef DEBUG
395             if (output[i * width + j].edge > 0)
396                 std::cout << output[i * width + j].edge;
397         #endif // DEBUG
398         //normalize -255 to 254
399         output[i * width + j].edge = 1024 * (output[i * width + j].edge / max);
400         #ifdef DEBUG
401             if (output[i * width + j].edge > 0)
402                 std::cout << output[i * width + j].edge << std::endl;
403         #endif // DEBUG
404         //edge weight to bool
405         if (THRESHOLD < output[i * width + j].edge) {
406             output[i * width + j].edge = 1;

```



Figure 1: 주어진 이미지에 대한 edge detecting



Figure 2: 쉬프트된 이미지에 대한 edge detecting

```

407         }
408         else {
409             output[i * width + j].edge = 0;
410         }
411     }
412 }
413 //std::cout << max << std::endl;
414 }

```

2.2 Result

Figure 1 에서 ref. 이미지와 Target 이미지를 비교하여 보면, 이미지의 밝기가 다른것을 확인 수 있다. 이미지의 밝기는 Harris edge detecting을 적용하였을때, 민감도가 달라지는 결과를 초래한다. 이러한 경우에도 결과 이미지 Figure 1 을 보면, edge detecting되어 같은 엣지끼리 연결되는 것을 확인 할 수 있다.

또한, Figure 2, 3에서 처럼 Shift와 약간의 회전이 가해진 Target과의 비교시에도 비교적 높은 확률의 정확도를 보여주었다.



Figure 3: 쉬프트되고 회전된 이미지에 대한 edge detecting

3 discussion

이 과제의 경우, 소스 두개를 받아 각 소스의 웨이트를 비교하여야 한다. 기존의 main문에서 사용하는 방식으론 매우 코드가 복잡해질것이 뻔하니, setVision함수를 만들어 weight정보를 담고있는 pixel스트럭트를 반환하게 하였다. 또한 함수들과 상수들을 모두 헤더파일로 이동하였다.

기존 알고리즘에서 변경 한 부분이 있다. 먼저 Ref.와 Target 이미지는 서로 밝기 정보가 달라 Harris Edge detecting을 사용하면 같은 Threshold를 주었을때 서로 구해지는 엣지 갯수가 달라진다. 이를 해결하기 위해 Harris weight을 계산된뒤에 노말라이즈를 하였다. 결과적으로, 양쪽 모두 비슷한 엣지를 검출하게 되었다. 다만 노말라이즈를 사용하면 이미지의 밝기가 달라진경우엔 쉽게 대응할 수 있지만 대비가 달라진경우에 적절히 대응하지 못한다. 따라서 개선방법으로는 threshold를 사용자가 지정하는게 아니라 검출할 edge 갯수를 지정하고, 프로그램 내부에서 해당 갯수만큼의 edge가 나오는 threshold를 찾아내어 적용 하는 방식이 있겠다.

두번째로 HOG weight을 구하는 과정에서 기존 코드는 180도를 20도 간격으로 나누어 0 20도를 0, 20 40도를 1 하는식으로 분류하였는데, 이는 경계에 가까울수록 부정확한 결과를 초래한다. 해당 부분을 수정하여, 더 높은 정확도를 얻을수 있는 linear한 방식을 사용했다. 0에서 8까지로 양자화 하는것은 같지만, 구간이 아닌 0, 20, 40 등의 경계선을 양자화하고 경계선에 가까운 정도를 두어 양쪽 경계선에 영향을 주도록했다. 예를 들어 30도라면 20도와 40도의 절반이므로 HOG weight을 구할때 $0.5 \times magnitude$ 를 1번과 2번 차원에 넣는다. 이는 회전된 이미지를 감지하는데 더 높은 정확도를 지닌다.

결과적으로 약하게 회전된 두 이미지 간의 edge를 확인하고 서로 같은엣지를 확인하는데 매우 높은 정확도를 얻어내었다.