전기전자심화설계 및 소프트웨어실습 〈Assignment 3〉

담당교수 : 김원준
이름 : 201810528 고려욱

1. Harris coner detect & HOG image matching

1) Source Code

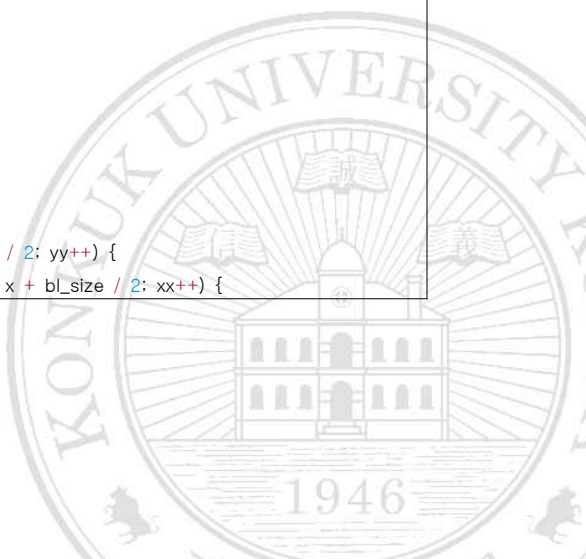
```
#pragma warning(disable:4996)
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <stdio.h>
#include <stdlib.h>
#include <cmath>
#define PI 3.141592
#define bl_size 16
using namespace cv;
void harris_corner(Mat input_r, Mat input_t)
{
		int x, y, xx, yy, z;
		int height = input_r.rows;
		int width = input_r.cols;
		Scalar c;
		Point pCenter;
		int radius = 3;
		float conv_x, conv_y;
		float dir;
		float ixix, iyiy, ixiy, det, tr;
		float k = 0.04, th = 900000;
		float min = 1000000, max = -100000;
		float* ix = (float*)calloc(height * width, sizeof(float));
		float* iy = (float*)calloc(height * width, sizeof(float));
		float* R = (float*)calloc(height * width, sizeof(float));
		float* mag = (float*)calloc(height * width, sizeof(float));
		float* o_dir = (float*)calloc(height * width, sizeof(float));
		int* cornerMap = (int*)calloc(height * width, sizeof(float));
		int* cornerMap_t = (int*)calloc(height * width, sizeof(float));
		Mat output(height, width*2, CV_8UC3);
		int b_size = 3;
		int win_size = 3;
		const int size = 9;
		float normal_pow;
		int count_r = 0, count_t = 0, max_sm = 0, max_idx = 0;
		int deg, idx_r = 0, idx_t = 0;
		float** dir_r_mag, ** dir_t_mag;
		int* pos_r, * pos_t, *idx_sm;
		int sm_count = 0;
		FILE* fp;

		int mask_x[size] = { -1,-1,-1,0,0,0,1,1,1 };
		int mask_y[size] = { -1,0,1,-1,0,1,-1,0,1 };
		//compute fx and fy for ref
		for (y = 0; y < height; y++) {
			for (x = 0; x < width; x++) {
				conv_x = 0;
				conv_y = 0;
				//inner product
```

```
                                for (yy = y - b_size / 2; yy <= y + b_size / 2; yy++) {
                                        for (xx = x - b_size / 2; xx <= x + b_size / 2; xx++) {
                                                if (yy >= 0 && yy < height && xx >= 0 && xx < width) {
                                                        conv_x += input_r.at<uchar>(yy, xx) * mask_x[(yy - (y - 1)) * b_
size + (xx - (x - 1))];
                                                        conv_y += input_r.at<uchar>(yy, xx) * mask_y[(yy - (y - 1)) * b_
size + (xx - (x - 1))];
                                                }
                                        }
                                }
                                conv_x /= 9;
                                conv_y /= 9;
                                mag[y * width + x] = sqrt(conv_x * conv_x + conv_y * conv_y);
                                dir = atan2(conv_y, conv_x);
                                dir = dir * 180.0 / PI;
                                o_dir[y * width + x] = dir;
                                ix[y * width + x] = conv_x;
                                iy[y * width + x] = conv_y;
                        }
                        output.at<Vec3b>(y, x)[0] = input_r.at<uchar>(y, x);
                        output.at<Vec3b>(y, x)[1] = input_r.at<uchar>(y, x);
                        output.at<Vec3b>(y, x)[2] = input_r.at<uchar>(y, x);
                }
                //R computation
                for (y = 0; y < height; y++) {
                        for (x = 0; x < width; x++) {
                                ixix = 0;
                                iyiy = 0;
                                ixiy = 0;
                                for (yy = y - win_size / 2; yy <= y + win_size / 2; yy++) {
                                        for (xx = x - win_size / 2; xx <= x + win_size / 2; xx++) {
                                                if (yy >= 0 && yy < height && xx >= 0 && xx < width) {
                                                        ixix += ix[yy * width + xx] * ix[yy * width + xx];
                                                        ixiy += ix[yy * width + xx] * iy[yy * width + xx];
                                                        iyiy += iy[yy * width + xx] * iy[yy * width + xx];
                                                }
                                        }
                                }
                                det = ixix * iyiy - ixiy * ixiy;
                                tr = ixix + iyiy;
                                R[y * width + x] = det - k * tr * tr;
                                //draw circle at coner
                                if (R[y * width + x] > th) {
                                        cornerMap[y * width + x] = 1;
                                        count_r++;
                                }
                                if (R[y * width + x] > max) max = R[y * width + x];
                                if (R[y * width + x] < min) min = R[y * width + x];
                        }
                }
                dir_r_mag = (float**)calloc(count_r, sizeof(float*));
                for (x = 0; x < count_r; x++) {
                        dir_r_mag[x] = (float*)calloc(9, sizeof(float));
```

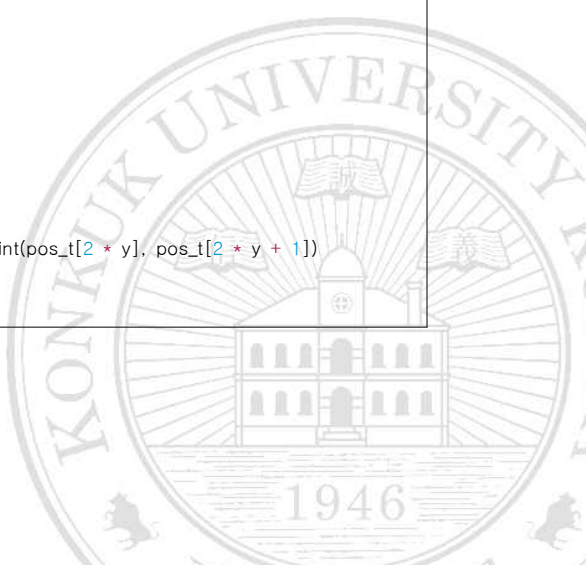
```
		}
		pos_r = (int*)calloc(count_r * 2, sizeof(float));
		fp = fopen("HOG.csv", "w");
		for (y = 0; y < height; y++) {
			for (x = 0; x < width; x++) {
				if (cornerMap[y * width + x] == 1) {
					for (yy = y - bl_size / 2; yy <= y + bl_size / 2; yy++) {
						for (xx = x - bl_size / 2; xx <= x + bl_size / 2; xx++) {
							if (yy >= 0 && yy < height && xx >= 0 && xx < width) {
								deg = o_dir[yy * width + xx] / 20.0;
								dir_r_mag[idx_r][deg] += mag[yy * width + xx];
							}
						}
					}
					normal_pow = 0.0;
					for (z = 0; z < 9; z++) {
						normal_pow += (dir_r_mag[idx_r][z] * dir_r_mag[idx_r][z]);
					}
					for (z = 0; z < 9; z++) {
						dir_r_mag[idx_r][z] = dir_r_mag[idx_r][z] / (sqrt(normal_pow) + 0.000000000
0001);
						fprintf(fp, "%d%d, %f₩n", idx_r, z, dir_r_mag[idx_r][z]);
					}
					pos_r[2 * idx_r] = x;
					pos_r[2 * idx_r+1] = y;
					pCenter.x = x;
					pCenter.y = y;
					c.val[0] = 0;
					c.val[1] = 255;
					c.val[2] = 0;
					circle(output, pCenter, radius, c, 2, 8, 0);
					idx_r++;
				}
			}
		}
		//compute fx and fy for bmp
		for (y = 0; y < height; y++) {
			for (x = 0; x < width; x++) {
				conv_x = 0;
				conv_y = 0;
				//inner product
				for (yy = y - b_size / 2; yy <= y + b_size / 2; yy++) {
					for (xx = x - b_size / 2; xx <= x + b_size / 2; xx++) {
						if (yy >= 0 && yy < height && xx >= 0 && xx < width) {
							conv_x += input_t.at<uchar>(yy, xx) * mask_x[(yy - (y - 1)) * b_
size + (xx - (x - 1))];
							conv_y += input_t.at<uchar>(yy, xx) * mask_y[(yy - (y - 1)) * b_
size + (xx - (x - 1))];
						}
					}
				}
				conv_x /= 9;
				conv_y /= 9;
```

```
                        mag[y * width + x] = sqrt(conv_x * conv_x + conv_y * conv_y);
                        dir = atan2(conv_y, conv_x);
                        dir = dir * 180.0 / PI;
                        o_dir[y * width + x] = dir;
                        ix[y * width + x] = conv_x;
                        iy[y * width + x] = conv_y;
                }
                output.at<Vec3b>(y, x+width)[0] = input_t.at<uchar>(y, x);
                output.at<Vec3b>(y, x+width)[1] = input_t.at<uchar>(y, x);
                output.at<Vec3b>(y, x+width)[2] = input_t.at<uchar>(y, x);
        }
        //R computation
        for (y = 0; y < height; y++) {
                for (x = 0; x < width; x++) {
                        ixix = 0;
                        iyiy = 0;
                        ixiy = 0;
                        for (yy = y - win_size / 2; yy <= y + win_size / 2; yy++) {
                                for (xx = x - win_size / 2; xx <= x + win_size / 2; xx++) {
                                        if (yy >= 0 && yy < height && xx >= 0 && xx < width) {
                                                ixix += ix[yy * width + xx] * ix[yy * width + xx];
                                                ixiy += ix[yy * width + xx] * iy[yy * width + xx];
                                                iyiy += iy[yy * width + xx] * iy[yy * width + xx];
                                        }
                                }
                        }
                        det = ixix * iyiy - ixiy * ixiy;
                        tr = ixix + iyiy;
                        R[y * width + x] = det - k * tr * tr;
                        //draw circle at coner
                        if (R[y * width + x] > th) {
                                cornerMap_t[y * width + x] = 1;
                                count_t++;
                        }
                        if (R[y * width + x] > max) max = R[y * width + x];
                        if (R[y * width + x] < min) min = R[y * width + x];
                }
        }
        free(R);
        free(ix);
        free(iy);
        dir_t_mag = (float**)calloc(count_t, sizeof(float*));
        for (x = 0; x < count_t; x++) {
                dir_t_mag[x] = (float*)calloc(9, sizeof(float));
        }
        pos_t = (int*)calloc(count_t * 2, sizeof(int));
        fp = fopen("HOG1.csv", "w");
        for (y = 0; y < height; y++) {
                for (x = 0; x < width; x++) {
                        if (cornerMap_t[y * width + x] == 1) {
                                for (yy = y - bl_size / 2; yy <= y + bl_size / 2; yy++) {
                                        for (xx = x - bl_size / 2; xx <= x + bl_size / 2; xx++) {
```



```
                                                if (yy >= 0 && yy < height && xx >= 0 && xx < width) {
                                                        deg = o_dir[yy * width + xx] / 20.0;
                                                        dir_t_mag[idx_t][deg] += mag[yy * width + xx];
                                                }
                                        }
                                }
                                normal_pow = 0.0;
                                for (z = 0; z < 9; z++) {
                                        normal_pow += (dir_t_mag[idx_t][z] * dir_t_mag[idx_t][z]);
                                }
                                for (z = 0; z < 9; z++) {
                                        dir_t_mag[idx_t][z] = dir_t_mag[idx_t][z] / (sqrt(normal_pow) + 0.000000000
0001);

                                        fprintf(fp, "%d%d, %f₩n", idx_t, z, dir_t_mag[idx_t][z]);
                                }
                                pos_t[2 * idx_t] = x + width;
                                pos_t[2 * idx_t + 1] = y;
                                pCenter.x = x + width;
                                pCenter.y = y;
                                c.val[0] = 0;
                                c.val[1] = 255;
                                c.val[2] = 0;
                                circle(output, pCenter, radius, c, 2, 8, 0);
                                idx_t++;
                        }
                }
        }
        imwrite("corner.bmp", output);
        free(o_dir);
        free(mag);
        fclose(fp);
        idx_sm = (int*)calloc(idx_r, sizeof(int));
        printf("%d %d₩n", idx_r, idx_t);
        for (y = 0; y < idx_t; y++) {
                max_sm = 0;
                max_idx = 0;
                for (x = 0; x < idx_r; x++) {
                        sm_count = 0;
                        for (xx = 0; xx < 9; xx++) {
                                if (abs(dir_r_mag[x][xx] - dir_t_mag[y][xx]) < 0.006) {
                                        sm_count++;
                                }
                        }
                        idx_sm[x] = sm_count;
                        if (max_sm < sm_count) {
                                max_sm = sm_count;
                                max_idx = x;
                        }
                }
                printf("%d  %d  %d₩n", max_sm, y, max_idx);
                line(output, Point(pos_r[2 * max_idx], pos_r[2 * max_idx + 1]), Point(pos_t[2 * y], pos_t[2 * y + 1])
                        , Scalar(255, 0, 0), 2, 9, 0);
        }
```



```
        imwrite("result.bmp", output);
}
void main()
{
        Mat imgref = imread("ref.bmp", CV_LOAD_IMAGE_GRAYSCALE);
        Mat imgtar = imread("tar.bmp", CV_LOAD_IMAGE_GRAYSCALE);
        harris_corner(imgref, imgtar);
}
```

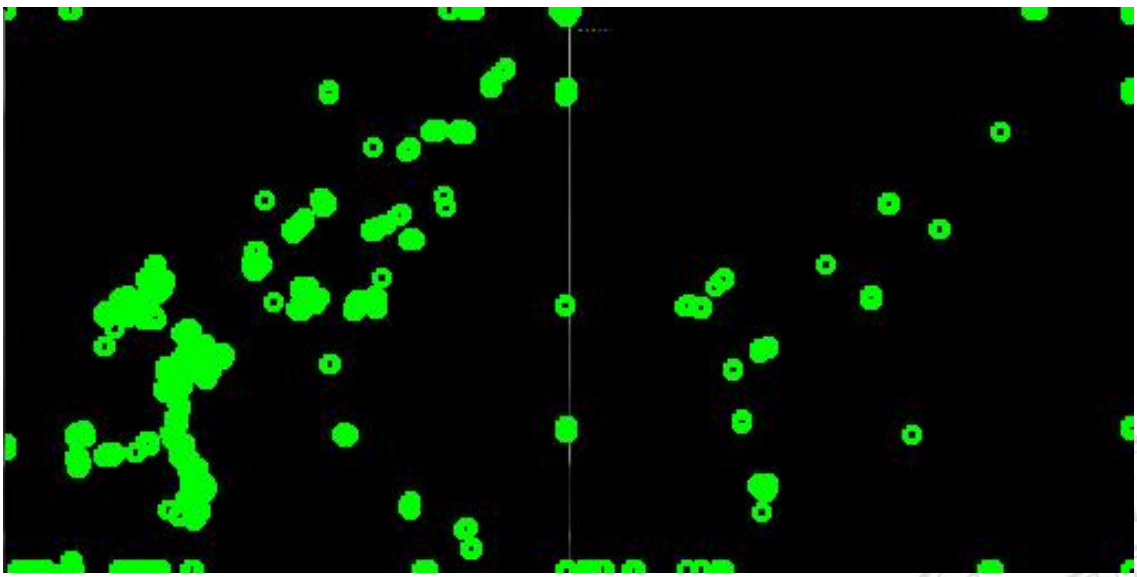
2) 결과 사진

(1) 사용 이미지

<ref.bmp>

<tar.bmp>

(2) Harris Corner detect 결과



(3) Coner 부근 HOG 결과 csv 2개

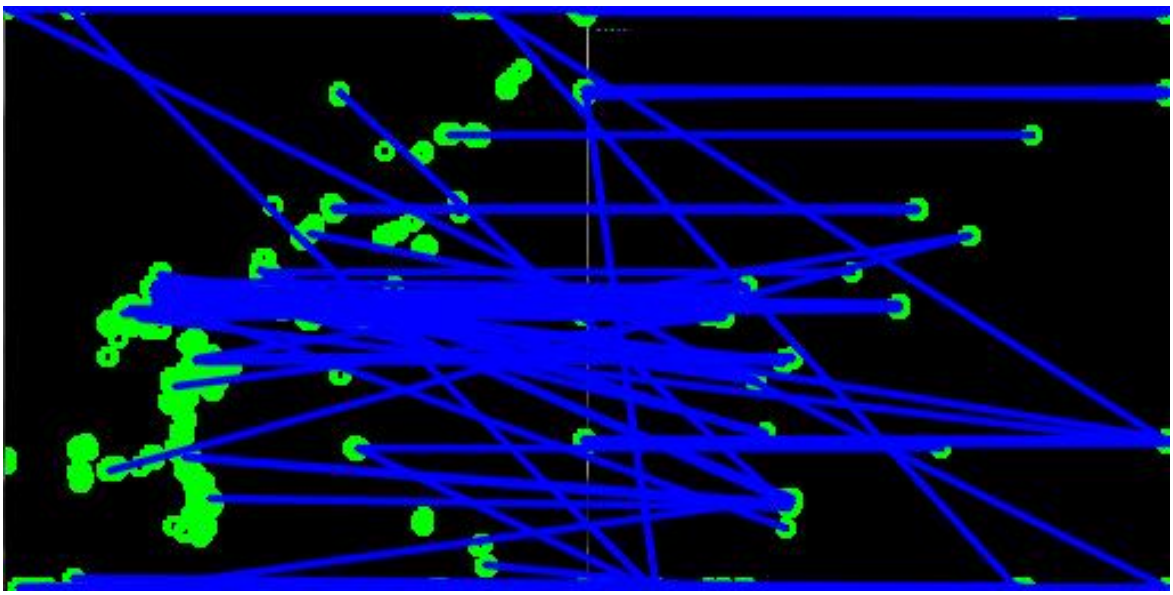
0	0.71138
1	0.001227
2	0.083791
3	0.001636
4	0.697692
5	0.003997
6	0.003621
7	0.009295
8	0.004802
10	0.750704
11	0.001153
12	0.078763
13	0.001537
14	0.65582
15	0.003757
16	0.003403
17	0.009506
18	0.004514
20	0.959181

<ref.bmp>

0	0.70986
1	0.001522
2	0.085113
3	0.001074
4	0.699075
5	0
6	0.005384
7	0.010214
8	0.003483
10	0.749712
11	0.00143
12	0.079967
13	0.001009
14	0.65681
15	0
16	0.005059
17	0.009596
18	0.004046
20	0.968824

<tar.bmp>

(4) Image Matching 사진



3) 결과 분석

이번 실습은 Harris Corner 알고리즘을 통해 코너를 검출한 후 코너 부근에 17*17 block을 주어 HOG를 계산한 후 HOG의 histogram의 유사도를 비교하여 corner점을 매칭하는 프로그램을 구현하는 것이다.

먼저, Harris Corner 구현과정에서 두 이미지 모두 코너가 잘 검출되었다. 코너 검출과정에서 thershold를 설정함에 있어 값을 높이면 검출되는 코너가 줄어들고 값을 낮추면 검출되는 코너가 많아진다. 이번 실습의 코너 검출에서 모든 범위의 threshold에서 ref가 tar보다 훨씬 많은 코너가 검출되었다. 지금 사용한 threshold는 90만으로 90만 이하의 값의 경우 ref에서 1000여개가 넘는 코너가 검출되어 메모리 상 뒤에 프로그램을 구현하는데 메모리의 부족의 이슈가 있었다. 또한 90만 이상의 값의 경우 tar의 코너가 100개 이하로 검출되어 두 이미지의 코너를 비교하는데 비교군이 너무 작은 문제가 있었다. 현재 프로그램 상 제일 최적의 임계값이 90만으로 생각되어 90만으로 설정하였다.

두번째로, HOG를 구현하는 부분에서 assignment2와 달리 이미지의 모든 부분이 아닌 코너부근을 중심으로 HOG를 사용하여야했기 때문에 코너를 검출하여 코너 display를 하는 부분에 HOG를 위한 코드를 추가하였다. dir_r_mag와 dir_t_mag를 각각 할당하여 두 이미지의 HOG값을 저장하여 후에 비교할 수 있도록 하였다. 또한, pos_r, pos_t의 배열을 할당하여 해당 HOG의 인덱스에 각 이미지의 위치를 저장 하여 이후에 선을 그어주는 작업을 진행 할 수 있도록 하였다. HOG를 구현함에 있어 nomalize를 위한 식을 사용할 때, 나눗셈의 분모가 0이 되지 않도록 하는 추가값을 0.00000000001로 주어 영향이 미미하게 줄 수 있도록 설정하였다. HOG.csv, HOG1.csv를 확인해 보면 코너의 수와 quantization 9개에 해당하는 값들이 잘 저장되어 있음을 확인 할 수 있었다.

마지막으로, 각 HOG를 비교하여 선을 그어주는 부분의 코드는 코너의 숫자가 적은 tar.bmp의 코너 하나당 ref.bmp의 코너 하나를 매칭할 수 있도록 for문을 사용하였다. 또한, sm_count와 sm_max, max_idx를 통해 각 코너의 HOG histogram을 비교할 수 있도록 진행하였다. 비교는 각 코너의 idx_r, idx_t의 deg를 비교하여 둘의 차이가 0.006이하면 sm_count가 올라가 최대 sm_count를 가지는 idx_t의 코너를 찾아 두 코너가 위치하는 지점을 연결하였다.

그러나, 결과를 보면 문제가 있음을 알 수 있다. 정확히 같은 위치의 코너들이 연결된 것이 아니라 중복되어 위치하는 코너가 있으며 전혀 다른 위치에 있는 코너와 유사도를 가져 엉뚱한 선이 그어진 경우가 많다. 먼저, 차이를 비교하는 것에 있어 어느정도의 차가 비슷하다고 볼 수 있을 지 애매하다. 0.006보다 적거나 많을 경우 각 경우마다 sm_count가 max가 되는 것이 비교하는 모든 코너마다 너무 많거나 적어 신뢰성이 떨어진다. 0.006의 경우에는 각 코너의 max_sm이 대략 4~5정도로 나타나 0.006을 사용하였다. 두번째로, 코드의 흐름 상 max_count가 동일하다면 idx_r이 적은 것이 max_idx가 되게 된다. 그 이유로 인해 많은 tar.bmp의 사진 아래에 위치한 코너들이 ref.bmp의 위쪽에 위치한 코너들과 연결된 것을 볼 수 있다.

지금 코드는 프로그램이 정상적으로 작동한다고 평가 할 수 없다. 위에서 나열한 두 가지의 문제점을 더 나은 방향으로 해결할 수 있다면, 더 성능 높은 image matching이 진행 될 것이라고 생각된다. 나아가, 메모리 동적할당을 더 세밀하게 진행하여 메모리를 아낄 수 있다면, 많은 수의 코너가 나타나더라도 충분히 비교를 할 수 있을 것이라고 생각한다.