컴퓨터비전 과제1 보고서

2019203021

소프트웨어학부

이승헌

1. 프로그램 동작

i) 확대할 사이즈를 입력합니다.

스크린샷, 텍스트, 컴퓨터, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

ii)원본과 확대된 사진이 보입니다. 이미지에 포커스를 두고 아무 키를 입력하면 콘솔에서 각도 입력이 가능해집니다.

텍스트, 스크린샷, 인간의 얼굴, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

iii)각도를 입력하면 원본 사진과 회전된 사진이 보이게 됩니다.

텍스트, 스크린샷, 여성, 흑백이(가) 표시된 사진

자동 생성된 설명

2. 소스 코드

#include <math.h>

#include <opencv2/opencv.hpp>

using namespace cv;

using namespace std;

int main()

{

Mat img\_in;

Mat img\_out;

int output\_size=0;

//HW1-1 시작

//출력할 이미지의 크기를 입력받음

while ((output\_size<128 ||output\_size>1024)) {

cout << "output size:";

cin >> output\_size;

if (!cin) {

cin.clear();

cin.ignore(INT\_MAX, '\n');

}

}

// image 읽고 gray로 바꾸기

img\_in = imread("Lena\_256x256.png");

cv::cvtColor(img\_in, img\_in, cv::COLOR\_RGB2GRAY);

cv::imshow("source img", img\_in);

unsigned char \*pData;

pData = (unsigned char \*)img\_in.data;

unsigned char\* outData = (unsigned char\* )malloc(output\_size \* output\_size);

//변수 초기화

for (int i = 0; i < output\_size \* output\_size; i++) {

outData[i] = 0;

}

double zoomSize = (double)output\_size / (double)256;

///////////////////// 처리하기 ///////////////////

/\*

\*

\*단순 확대

for (int i = 0; i < output\_size; i++) {

int expected\_y = round(i \* zoomSize);

for (int j = 0; j < output\_size; j++) {

int expected\_x = round(j \* zoomSize);

outData[expected\_y \* output\_size + expected\_x] = pData[i \* 256 + j];

}

}

\*/

//bilinear interpolation 처리하는 코드

for (int i = 0; i < output\_size; i++) {

for (int j = 0; j < output\_size; j++) {

//p1의 실제좌표에 매칭될 것을 구한다고 가정하면 된다.

int p1\_x = (int)(j / zoomSize); //확대된만큼 나누기해서 원래 좌표를 찾는다. 매핑할 때 round하므로 축소할 때도 round.

int p1\_y = (int)(i / zoomSize);

int p2\_x = p1\_x+1;

int p2\_y = p1\_y;

int p3\_x = p1\_x;

int p3\_y = p1\_y+1;

int p4\_x = p1\_x+1;

int p4\_y = p1\_y+1;

double distance\_of\_x\_from\_left = ((double)j / (double)zoomSize) - (double)p1\_x; //실제 거리가 아닌 닮음(즉, 비율)을 이용해서 (실수 2차원 공간이었다면 매핑되었을 x 좌표)-(정수 2차원 공간에어서 매핑된 x 좌표)=a 임을 알아낸다.

double distance\_of\_x\_from\_right = 1 - distance\_of\_x\_from\_left; // 1-a

double distance\_of\_y\_from\_top = ((double)i / (double)zoomSize) - (double)p1\_y;//실제 거리가 아닌 닮음(즉, 비율)을 이용해서 (실수 2차원 공간이었다면 매핑되었을 y 좌표)-(정수 2차원 공간에어서 매핑된 yc 좌표)=a 임을 알아낸다.

double distance\_of\_y\_from\_bottom = 1 - distance\_of\_y\_from\_top;// 1-b

double weight\_of\_p1 = distance\_of\_x\_from\_left \* distance\_of\_y\_from\_top;//(1-a)\*b\*p1

double weight\_of\_p2 = distance\_of\_x\_from\_right \* distance\_of\_y\_from\_top;//a\*b\*p2

double weight\_of\_p3 = distance\_of\_x\_from\_left \* distance\_of\_y\_from\_bottom;//(1-a)\*(1-b)\*p3

double weight\_of\_p4 = distance\_of\_x\_from\_right \* distance\_of\_y\_from\_bottom;//a\*(1-b)\*p4

outData[output\_size \* i + j] = weight\_of\_p1 \* pData[p1\_y \* 256 + p1\_x] + weight\_of\_p2 \* pData[p2\_y \* 256 + p2\_x] + weight\_of\_p3 \* pData[p3\_y \* 256 + p3\_x] + weight\_of\_p4 \* pData[p4\_y \* 256 + p4\_x];

}

}

/\*

\* 단순 축소

else {

for (int i = 0; i < img\_in.cols; i++) {

int expected\_y = floor(i \* zoomSize);

for (int j = 0; j < img\_in.rows; j++) {

int expected\_x = floor(j \* zoomSize);

outData[expected\_y \* output\_size + expected\_x] = pData[i \* 256 + j];

}

}

}

\*/

//출력에 쓰일 Mat 생성 및 출력

img\_out=Mat(output\_size,output\_size,CV\_8UC1);

img\_out.data = outData;

cv::imshow("output image", img\_out);

cv::waitKey(0);

/////////////////////////////

//HW1-2 시작

Mat img\_in2;

Mat img\_out2;

int output\_angle= -1;

//각도 입력

while (output\_angle < 0) {

cout << "angle:";

cin >> output\_angle;

if (!cin) {

cin.clear();

cin.ignore(INT\_MAX, '\n');

}

}

//입력 받은 이미지 흑백으로 변환

img\_in2 = imread("Lena.png");

cv::cvtColor(img\_in2, img\_in2, cv::COLOR\_RGB2GRAY);

cv::imshow("input", img\_in2);

int original\_size = img\_in2.rows;

unsigned char\* original\_data= img\_in2.data;

//변수 초기화

unsigned char\* rotated\_data = (unsigned char\*)malloc(original\_size \* original\_size);

for (int i = 0; i < original\_size \* original\_size; i++) {

rotated\_data[i] = 0;

}

/\*

단순히 좌표의 이동만을 생각한다. 즉, 중간중간 비는 픽셀의 보정은 하지 않은 상태.

for (int i = 0; i < original\_size; i++) {

for (int j = 0; j < original\_size; j++) {

int x = cos(output\_angle \* 3.14 / 180) \* (j - original\_size / 2) - sin(output\_angle \* 3.14 / 180) \* (i - original\_size / 2) + original\_size / 2;

int y = sin(output\_angle \* 3.14 / 180) \* (j - original\_size / 2) +cos(output\_angle \* 3.14 / 180) \* (i - original\_size / 2) + original\_size / 2;

if (x >= 0 && y >= 0) {

rotated\_data[y \* original\_size + x] = original\_data[i \* original\_size + j];

}

}

}

\*/

////처리하기

//rotation 처리하는 코드

for (int i = 0; i < original\_size; i++) {

for (int j = 0; j < original\_size; j++) {

//이번에는 원본이 회전되었을 경우 실제 실수 좌표가 어딜지를 구한다.

int p1\_x = (int)(cos(output\_angle \* 3.14 / 180) \* (j - original\_size / 2) - sin(output\_angle \* 3.14 / 180) \* (i - original\_size / 2) + original\_size / 2); //transform을 그대로 적용한다.

int p1\_y = (int)(sin(output\_angle \* 3.14 / 180) \* (j - original\_size / 2) + cos(output\_angle \* 3.14 / 180) \* (i - original\_size / 2) + original\_size / 2);

if ((p1\_x < 0 || p1\_y < 0) || (p1\_x >= original\_size || p1\_y >= original\_size)) { //만약 transform 결과, 영역에서 벗어나는 경우 매핑하는 과정을 수행하지 않는다.

continue;

}

int p2\_x = p1\_x + 1;

int p2\_y = p1\_y;

int p3\_x = p1\_x;

int p3\_y = p1\_y + 1;

int p4\_x = p1\_x + 1;

int p4\_y = p1\_y + 1;

double distance\_of\_x\_from\_left = cos(output\_angle \* 3.14 / 180) \* (j - original\_size / 2) - sin(output\_angle \* 3.14 / 180) \* (i - original\_size / 2) + original\_size / 2 - (double)p1\_x; //실제 거리가 아닌 닮음(즉, 비율)을 이용해서 (실수 2차원 공간이었다면 매핑되었을 x 좌표)-(정수 2차원 공간에어서 매핑된 x 좌표)=a 임을 알아낸다.

double distance\_of\_x\_from\_right = 1 - distance\_of\_x\_from\_left; // 1-a

double distance\_of\_y\_from\_top = sin(output\_angle \* 3.14 / 180) \* (j - original\_size / 2) + cos(output\_angle \* 3.14 / 180) \* (i - original\_size / 2) + original\_size / 2 - (double)p1\_y;//실제 거리가 아닌 닮음(즉, 비율)을 이용해서 (실수 2차원 공간이었다면 매핑되었을 y 좌표)-(정수 2차원 공간에어서 매핑된 y 좌표)=b임을 알아낸다.

double distance\_of\_y\_from\_bottom = 1 - distance\_of\_y\_from\_top;// 1-b

double weight\_of\_p1 = distance\_of\_x\_from\_left \* distance\_of\_y\_from\_top;//(1-a)\*b\*p1

double weight\_of\_p2 = distance\_of\_x\_from\_right \* distance\_of\_y\_from\_top;//a\*b\*p2

double weight\_of\_p3 = distance\_of\_x\_from\_left \* distance\_of\_y\_from\_bottom;//(1-a)\*(1-b)\*p3

double weight\_of\_p4 = distance\_of\_x\_from\_right \* distance\_of\_y\_from\_bottom;//a\*(1-b)\*p4

rotated\_data[original\_size \* i + j] = weight\_of\_p1 \* original\_data[p1\_y \* original\_size + p1\_x] + weight\_of\_p2 \* original\_data[p2\_y \* original\_size + p2\_x] + weight\_of\_p3 \*original\_data[p3\_y \* original\_size + p3\_x] + weight\_of\_p4 \* original\_data[p4\_y \* original\_size + p4\_x];

}

}

//출력하기 위한 Mat 생성과 출력

img\_out2=Mat(original\_size, original\_size, CV\_8UC1);

img\_out2.data = rotated\_data;

cv::imshow("output", img\_out2);

cv::waitKey(0);

std::free(outData);

free(rotated\_data);

return 0;

}

3. 결과 보고

436x436

텍스트, 의류, 머리장식, 모자이(가) 표시된 사진

자동 생성된 설명

원본의 실수배여서 사선이 깨지는 현상이 잘 보입니다.

머리장식, 의류, 모자, 인간의 얼굴이(가) 표시된 사진

자동 생성된 설명

512x512

436x436에 비해 사선이 깨지는 현상이 적어보입니다.

인간의 얼굴, 의류, 모자, 패션 액세서리이(가) 표시된 사진

자동 생성된 설명

30도

인간의 얼굴, 모자, 스크린샷, 소녀이(가) 표시된 사진

자동 생성된 설명

45도

인간의 얼굴, 모자, 흑백, 포스터이(가) 표시된 사진

자동 생성된 설명

60도