컴퓨터비전 과제 2 보고서

2019203021

소프트웨어 학부

이승헌

1. 구현 여부 요약

1. Color Slicing

1) 얼굴색이면 그대로, 아니면 Black 으로 저장 성공
(기능상으로만 봤을 때는 그렇습니다. 성능 상으로는 다소 아쉬운 점이 있습니다.)

2) 흑백 컬러 Mat 에 흑과 백으로 얼굴 영역을 매핑하기 성공

2. Smoothing 과 결합

1) smoothing 필터 적용 성공
2) 필터링된 영상과 원본 영상의 결합 성공

2. 프로그램 구동 및 결과 요약

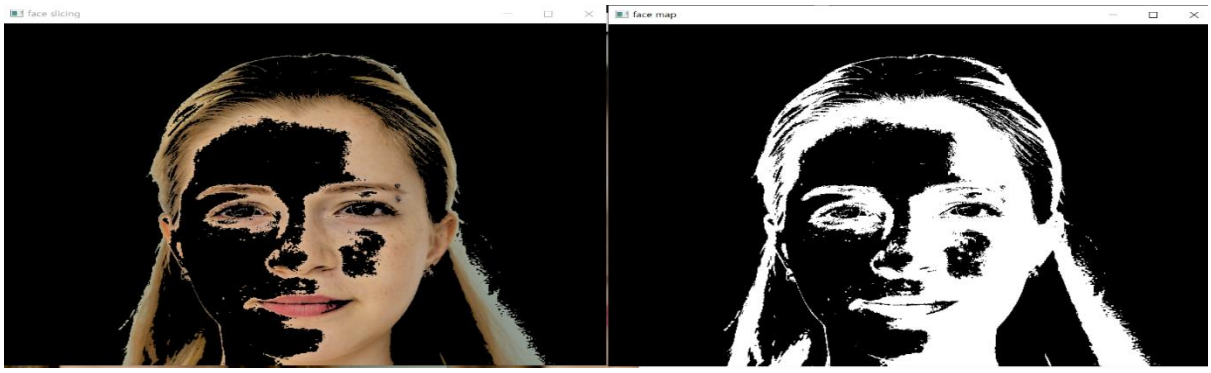
0. 진행하기에 앞서

```
const unsigned char STANDARD_RED = 175;
const unsigned char STANDARD_GREEN = 140;
const unsigned char STANDARD_BLUE = 102;
const unsigned char LOWER = 75;
const unsigned char UPPER = 45;
const unsigned char N = 4;
```

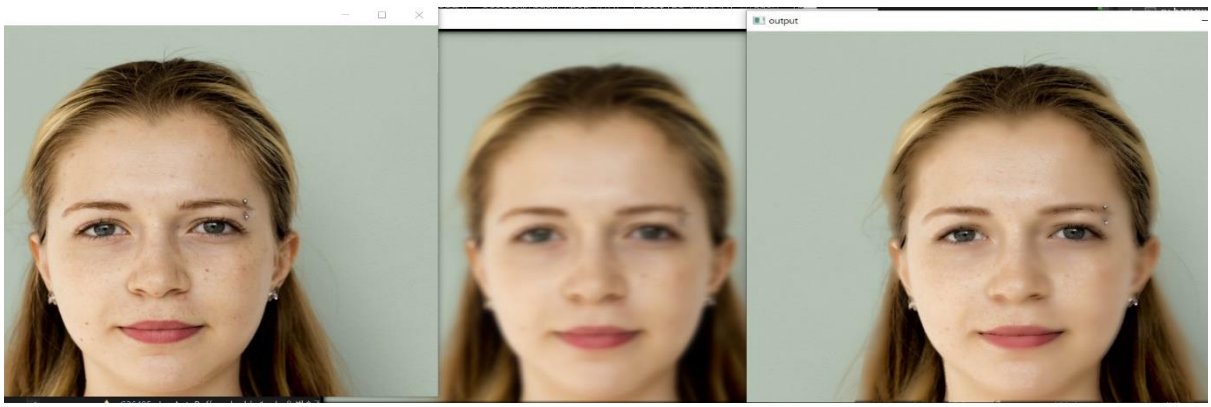
```
cv::Mat original;
original=cv::imread("sample.png");
```

메인 함수에 있는 이 변수들을 조작하면 다른 영상에 대해서도 구동 가능합니다. (기준 – LOWER)가 하한, (기준 + UPPER)가 상한이고 N 은 (2N+1)x(2N+1)필터의 크기를 설정합니다.

1. 0 번 항목에서 언급된 변수들을 알맞게 조작하고 실행하면 이미지가 무작위 위치에 5 장 나오게 됩니다.

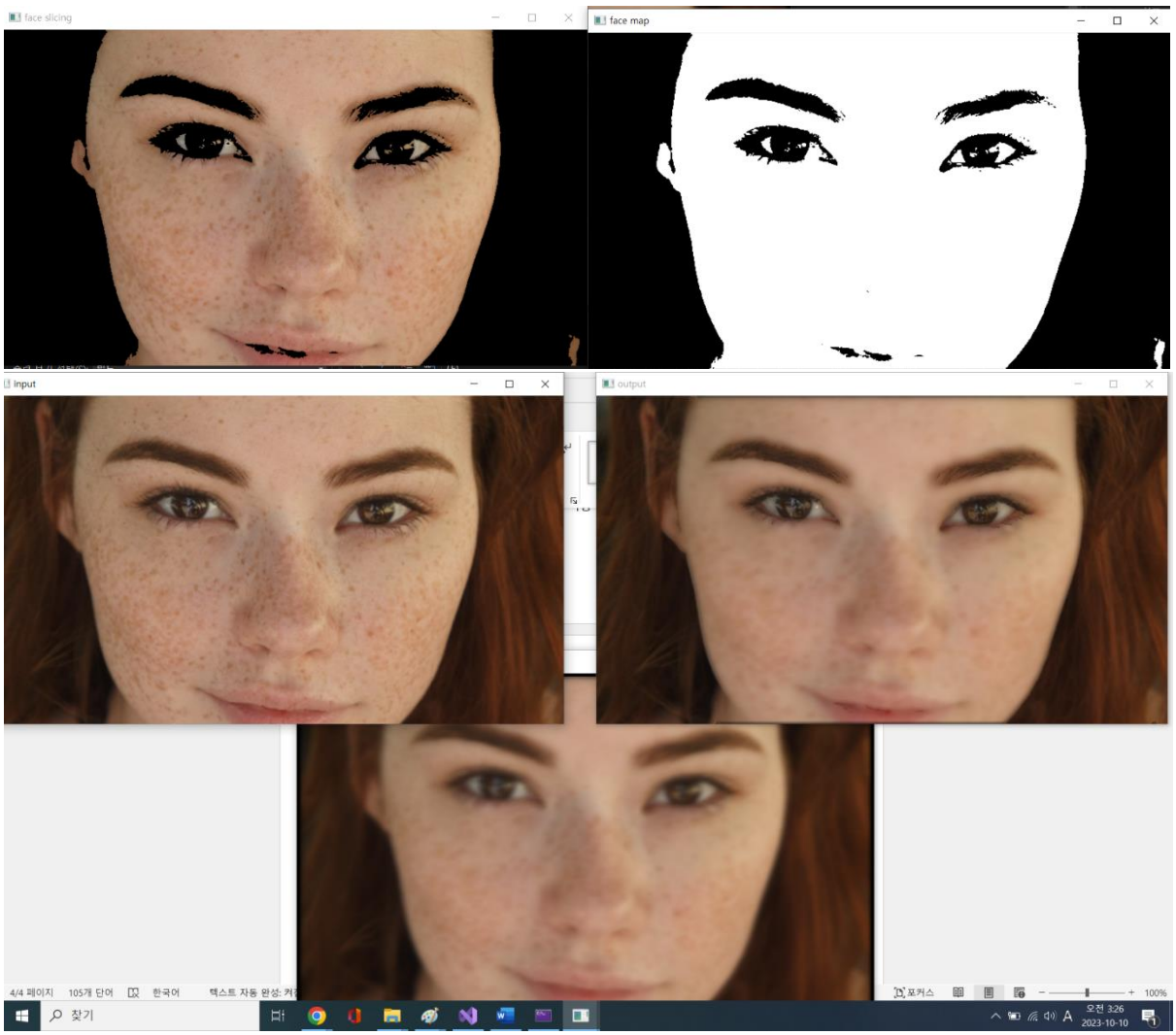


Face map



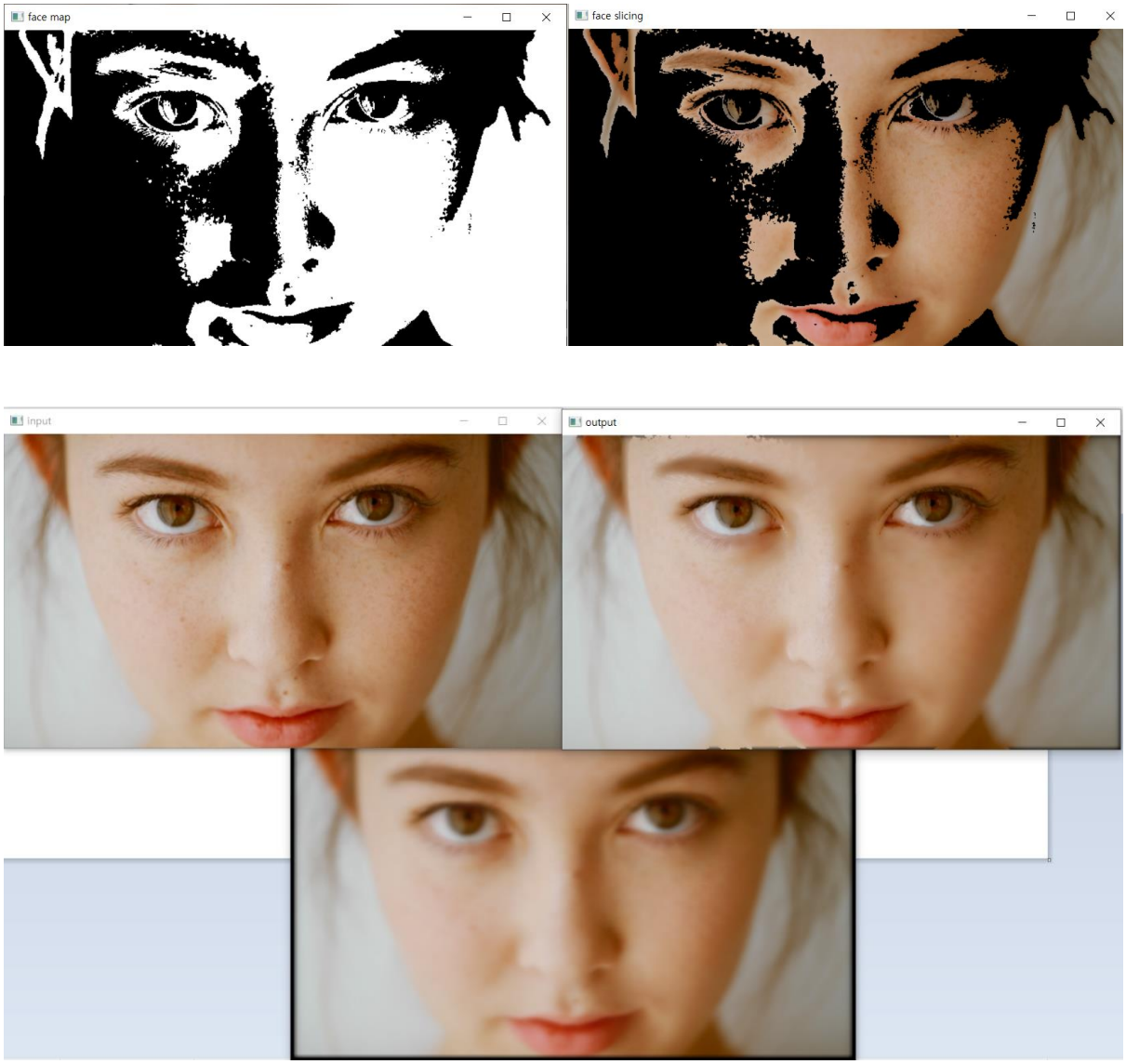
(왼쪽부터 원본, smoothing 필터, 필터 적용 결과)

영상이 빛의 영향도 비교적 적고 잡티가 몰려 있지 않아 효과가 괜찮았습니다. 다른 영상 2 개에 대한 진행 결과입니다.



(좌상단 원본, 중하단 smoothing, 우상단 결과물)

이 영상의 경우 빛의 영향이 매우 적어 얼굴을 추출하는 데에는 유리하지만, 평균을 잡더라도 잡티가 몰려 있는 경우가 많아서 미백 효과는 적었습니다.



(좌상단 원본, 중하단 smoothing, 좌상단 결과물)

이 영상의 경우 잡티가 몰려 있지 않아 미백 효과는 있었지만, 밝은 곳은 너무 밝고 어두운 곳은 너무 어두운 탓에 얼굴 슬라이싱이 제대로 동작하기 힘들었습니다.

3. 소스 코드

main.cpp

```
#include <opencv2/opencv.hpp>
#include <iostream>

int main() {

	const unsigned char STANDARD_RED = 175;
	const unsigned char STANDARD_GREEN = 140;
	const unsigned char STANDARD_BLUE = 102;
	const unsigned char LOWER = 75;
	const unsigned char UPPER = 45;
	const unsigned char N = 4;

	cv::Mat original;
	original=cv::imread("sample.png");

	cv::Mat face_slicing;
	face_slicing = original.clone();


	cv::Mat face_map = cv::Mat(face_slicing.rows, face_slicing.cols, CV_8UC1);

	cv::Mat output = cv::Mat::zeros(original.rows, original.cols, CV_8UC3);


	for (int i = 0; i < face_slicing.rows; i++) {
		for (int j = 0; j < face_slicing.cols; j++) {
			unsigned char blue=face_slicing.data[3*(i*face_slicing.cols+j)];
			unsigned char green = face_slicing.data[3 * (i*face_slicing.cols +
j)+1];
			unsigned char red = face_slicing.data[3 * (i*face_slicing.cols +
j)+2];
			if((STANDARD_RED-LOWER>red || STANDARD_RED+UPPER<red)||(STANDARD_BLUE
- LOWER > blue || STANDARD_BLUE + UPPER < blue)||(STANDARD_GREEN - LOWER > green ||
STANDARD_GREEN + UPPER < green))
			{
				face_slicing.data[3 * (i * face_slicing.cols + j)] = 0;
				face_slicing.data[3 * (i * face_slicing.cols + j) + 1] = 0;
				face_slicing.data[3 * (i * face_slicing.cols + j) + 2] = 0;

				face_map.data[i*face_slicing.cols+j]=0;
			}
			else {
				face_map.data[i * face_slicing.cols + j] = 255;
			}

		}
	}

	cv::Mat zero_padding_rgb = cv::Mat::zeros(original.rows + 2*N, original.cols + 2*N,
CV_8UC3);
```

```
	for (int i = N; i < original.rows+N; i++) {
		for (int j = N; j < original.cols+N; j++) {
			zero_padding_rgb.data[3 * ((i) * (original.cols+2*N) + j)] =
original.data[3 * ((i-N) * original.cols + (j-N))];
			zero_padding_rgb.data[3 * ((i) * (original.cols+2*N) + j) + 1] =
original.data[3 * ((i-N) * original.cols + (j-N)) + 1];
			zero_padding_rgb.data[3 * ((i) * (original.cols+2*N) + j) + 2] =
original.data[3 * ((i-N) * original.cols + (j-N)) + 2];
		}
	}

	for (int i = N; i < zero_padding_rgb.rows-N; i++) {
		for (int j = N; j < zero_padding_rgb.cols-N; j++) {
			int red_total = 0;
			int blue_total = 0;
			int green_total = 0;
			for (int a = 0; a < 2 * N + 1; a++) {
				for (int b = 0; b < 2 * N + 1; b++) {
					blue_total += zero_padding_rgb.data[3 * ((i - N + a)
* zero_padding_rgb.cols + (j - N + b))];
					green_total += zero_padding_rgb.data[3 * ((i - N +
a) * zero_padding_rgb.cols + (j - N + b)) + 1];
					red_total += zero_padding_rgb.data[3 * ((i - N + a)
* zero_padding_rgb.cols + (j - N + b)) + 2];
				}
			}
			zero_padding_rgb.data[3 * (i * zero_padding_rgb.cols + j)] =
blue_total / ((2 * N + 1) * (2 * N + 1));
			zero_padding_rgb.data[3 * (i * zero_padding_rgb.cols + j) + 1] =
green_total / ((2 * N + 1) * (2 * N + 1));
			zero_padding_rgb.data[3 * (i * zero_padding_rgb.cols + j) + 2] =
red_total/ ((2 * N + 1) * (2 * N + 1));
		}
	}
	for (int i = 0; i < original.rows; i++) {
		for (int j = 0; j < original.cols; j++) {
			if (face_map.data[i * original.cols + j] ==255) {
				output.data[3*(i*original.cols+j)] = zero_padding_rgb.data[3
* ((i + N) * zero_padding_rgb.cols + (j + N))];
				output.data[3 * (i * original.cols + j)+1] =
zero_padding_rgb.data[3 * ((i + N) * zero_padding_rgb.cols + (j + N))+1];
				output.data[3 * (i * original.cols + j)+2] =
zero_padding_rgb.data[3 * ((i + N) * zero_padding_rgb.cols + (j + N))+2];
			}
			else {
				output.data[3 * (i * original.cols + j)] = original.data[3 *
(i * original.cols + j)];
				output.data[3 * (i * original.cols + j) + 1] =
original.data[3 * (i * original.cols + j) + 1];
				output.data[3 * (i * original.cols + j) + 2] =
original.data[3 * (i * original.cols + j) + 2];
			}
		}
	}
```

```
        cv::imshow("input", original);
        cv::imshow("face slicing", face_slicing);
        cv::imshow("face map", face_map);
        cv::imshow("smoothing",zero_padding_rgb );
        cv::imshow("output",output);
        cv::waitKey(0);
        return 0;
}
```