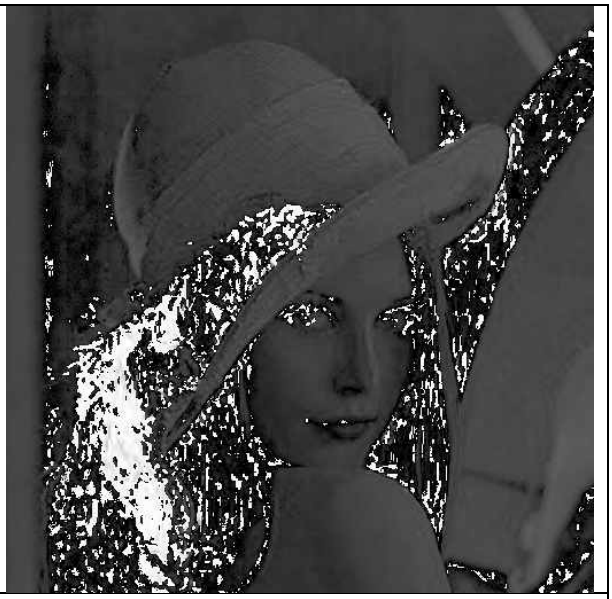
Course Title (과목명)	디지털영상처리개론 (01)
HW Number (HW 번호)	HW05
Submit Date (제출일)	2021-06-15
Grade (학년)	4th Grade
ID (학번)	20161482
Name (이름)	박준용

Color image processing

- **Color image processing 구현.**
  - ✓ **OpenCV의 Mat class method 및 입/출력 함수(imread와 imwrite)만을 사용함.**
  - ✓ **제공된 HW05.cpp 및 dft.cpp, utils.h 파일은 수정하지 않고, utils.cpp파일을 작성하여 제출함.**

(1) 구현 1

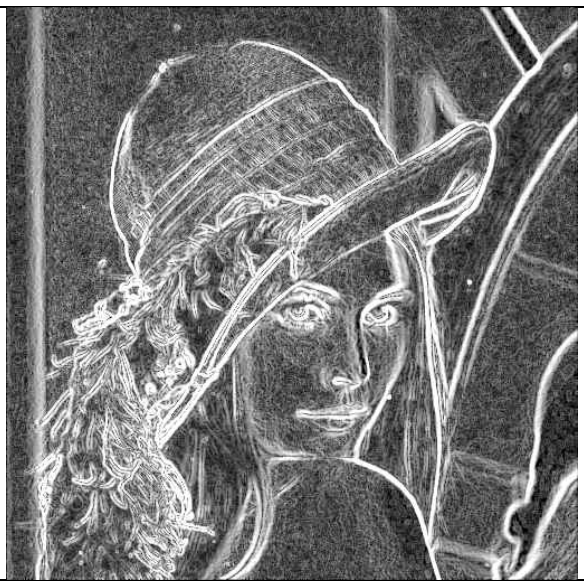
a) *.zip에 포함된 **"Lenna.tif"**를 HSI color space로 변환하고 hue, saturation, intensity를 각각 따로 출력하시오.

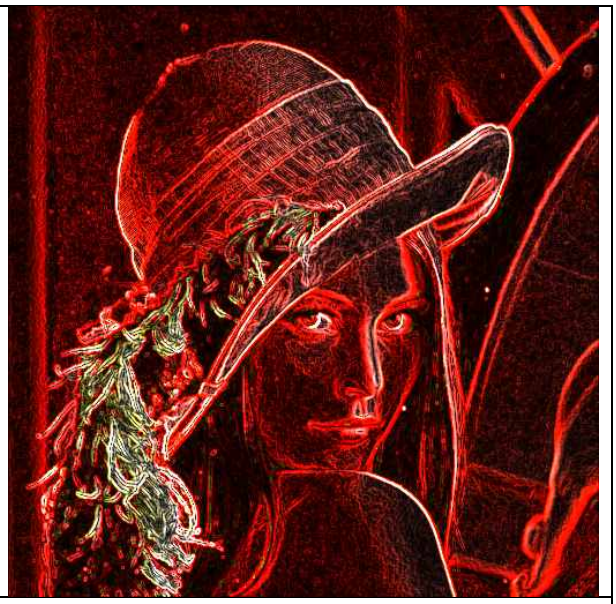
	
Original	H

	
S	I

b) RGB color space와 HSI color space에서 Sobel edge를 구하시오.

- RGB color space : R, G, B 각각에 대하여 Sobel edge를 계산
- HSI color space : Intensity에 대해서만 Sobel edge를 계산

	
Sobel in R	Sobel in G

	
Sobel in B	Sobel in HSI (Intensity)

c) 6.2)를 참고하여 작성하도록 함.

■ C/C++ : cvtBGR2HSI(), cvtHSI2BGR(), sobelFilter()

```
Mat cvtBGR2HSI(Mat src) {
	Mat dst(src.size(), src.type());

	Mat B(src.size(), src.type());
	Mat G(src.size(), src.type());
	Mat R(src.size(), src.type());

	for (int x = 0; x < src.rows; x++) {
		for (int y = 0; y < src.cols; y++) {
			B.at<float>(x, y) = src.at<Vec3f>(x, y)[0];
			G.at<float>(x, y) = src.at<Vec3f>(x, y)[1];
			R.at<float>(x, y) = src.at<Vec3f>(x, y)[2];

			if ((R.at<float>(x, y) == G.at<float>(x, y) && G.at<float>(x, y) ==
B.at<float>(x, y))) {
				dst.at<Vec3f>(x, y)[0] = 0;
			}
			else {
				float red = R.at<float>(x, y);
				float green = G.at<float>(x, y);
				float blue = B.at<float>(x, y);
				float denom = sqrt((red - green) * (red - green) + (red -
blue) * (green - blue));
				float numer = (red - green + red - blue) / 2;
				float theta = acos(numer / denom);
				theta = 180 * theta / M_PI;
				if (B.at<float>(x, y) <= G.at<float>(x, y)) {
					dst.at<Vec3f>(x, y)[0] = theta / 360;
```

```
                                }
                                else {
                                        dst.at<Vec3f>(x, y)[0] = (360 - theta) / 360;
                                }
                        }
                        dst.at<Vec3f>(x, y)[1] = 1 - 3 * min(min(R.at<float>(x, y),
G.at<float>(x, y)), B.at<float>(x, y)) / (R.at<float>(x, y) + G.at<float>(x, y) +
B.at<float>(x, y));
                        dst.at<Vec3f>(x, y)[2] = (R.at<float>(x, y) + G.at<float>(x, y) +
B.at<float>(x, y)) / 3;
                }
        }

        return dst;
}
```

- $H = \begin{cases} \theta, & \text{if } B \le G \\ 360 - \theta, & \text{if } B > G \end{cases} \; where\ \theta = \cos^{-1}\left\{ \frac{\frac{1}{2}[(R-G)+(R-B)]}{\sqrt{(R-G)^2+(R-B)(G-B)}} \right\}$
- $S = 1 - \frac{3}{(R+G+B)}[\min(R,G,B)]$
- $I = \frac{1}{3}(R+G+B)$

우선, RGB color model을 HSI color model로 변경하는 cvtBGR2HSI 함수는 위의 수식을 따라 구현하였다. 입력받은 input image source를 Vec3f를 통하여 RGB scale로 분리한 후, 각각 hue, saturation, intensity로 변경하여 result image의 Vec3f값에 적용하여 반환하도록 하였다.

```
Mat cvtHSI2BGR(Mat src) {
        Mat dst(src.size(), src.type());

        Mat H(src.size(), src.type());
        Mat S(src.size(), src.type());
        Mat I(src.size(), src.type());
        float B;
        float G;
        float R;

        for (int x = 0; x < src.rows; x++) {
                for (int y = 0; y < src.cols; y++) {
                        H.at<float>(x, y) = src.at<Vec3f>(x, y)[0];
                        S.at<float>(x, y) = src.at<Vec3f>(x, y)[1];
                        I.at<float>(x, y) = src.at<Vec3f>(x, y)[2];

                        float hue = H.at<float>(x, y);
                        float sat = S.at<float>(x, y);
                        float inten = I.at<float>(x, y);

                        if (0 <= hue && 180 * hue / M_PI <= 120) {
                                R = inten * (1 + sat * cos(hue) / cos( M_PI / 3 - hue));
                                B = inten * (1 - sat);
                                G = 3 * inten - (R + B);
                        }
```

```
                        else if (120 < 180 * hue / M_PI && 180 * hue / M_PI <= 240) {
                                G = inten * (1 + sat * cos(hue) / cos( M_PI / 3 - hue));
                                R = inten * (1 - sat);
                                B = 3 * inten - (R + B);
                        }
                        else if (240 < 180 * hue / M_PI && 180 * hue / M_PI <= 360) {
                                B = inten * (1 + sat * cos(hue) / cos( M_PI / 3 - hue));
                                G = inten * (1 - sat);
                                R = 3 * inten - (R + B);
                        }
                        dst.at<Vec3f>(x, y)[0] = B;
                        dst.at<Vec3f>(x, y)[1] = G;
                        dst.at<Vec3f>(x, y)[2] = R;
                }
        }

        return dst;
}
```

- RG sector $(0° \le H \le 120°)$
  - $B = I(1-S), R = I\left[1 + \frac{S\cos H}{\cos(60°-H)}\right], G = 3I - (R+B)$
- GB sector $(120° \le H \le 240°)$ $H = H - 120°$
  - $R = I(1-S), G = I\left[1 + \frac{S\cos H}{\cos(60°-H)}\right], B = 3I - (R+G)$
- BR sector $(240° \le H \le 360°)$ $H = H - 240°$
  - $G = I(1-S), B = I\left[1 + \frac{S\cos H}{\cos(60°-H)}\right], R = 3I - (G+B)$

HSI color model을 RGB color model로 변경하는 cvtHSI2BGR 함수 역시 위의 수식을 따라 구현하도록 하였다. 마찬가지로, 입력받은 input image source를 Vec3f를 통하여 HIS scale로 분리한 후, 각각의 sectors of interest에 대해서 R, G, B값을 정하여 result image의 Vec3f값에 적용하여 반환하도록 하였다.

```
Mat sobelFilter(Mat src, float direction) {
        Mat sobel_matrix = Mat::zeros(Size(3, 3), CV_32F);
        float* sobel = (float*)sobel_matrix.data;
        if (direction == 0) {
                sobel[0] = -1;   sobel[1] = -2;   sobel[2] = -1;
                sobel[3] = 0;    sobel[4] = 0;    sobel[5] = 0;
                sobel[6] = 1;    sobel[7] = 2;    sobel[8] = 1;
        }
        else if (direction == 1) {
                sobel[0] = -1;   sobel[1] = 0;    sobel[2] = 1;
                sobel[3] = -2;   sobel[4] = 0;    sobel[5] = 2;
                sobel[6] = -1;   sobel[7] = 0;    sobel[8] = 1;
        }

        Mat dst(src.size(), src.type());
        filter2D(src, dst, -1, sobel_matrix);
        return dst;
}
```

X – Direction Kernel

-1	0	1
-2	0	2
-1	0	1

Y – Direction Kernel

-1	-2	-1
0	0	0
1	2	1

sobelFilter() 함수의 경우 input source image와 image direction을 입력으로 받는다. Sobel filter를 구현하기 위한 kernel은 image direction에 따라 위와 같으며, 이러한 kernel을 matrix를 이용하여 구현한 다음 direction에 따라 image에 filter를 적용하여 반환할 수 있다.

4. 검토사항

1) 구현 1에 대한 결과를 분석하여 보고서에 첨부하시오.

2) RGB color space와 HSI color space의 장단점을 비교하고 색을 표현하기 위한 다른 방식에 대해서 1가지이상 조사하시오.

RGB는 색상 표현이 컴퓨터에서 처리하기 쉬우며, bit 수를 늘리는 등의 조작을 통하여 다양한 종류의 색상을 표시할 수 있지만, RGB에 대하여 색을 인식하는 수준이 사람에 따라 다르게 되며, 실생활에서 CMYK 등으로 변환하였을 때의 색조의 차이 역시 존재한다. HIS는 색상이 구현되는 구조인 색상, 채도와 그것의 intensity를 구분하여 직관적으로 나타내게 할 수 있지만, Hue에서 Red 값의 discontinuity 등이 문제로 꼽힌다.

$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.5748 \\ 1 & -0.1873 & -0.4681 \\ 1 & 1.8556 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ Cb \\ Cr \end{bmatrix}$	$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 \\ 1 & 0 & 1 \\ 1 & -1 & -1 \end{bmatrix} \cdot \begin{bmatrix} Y \\ Co \\ Cg \end{bmatrix}$
Conversion from YCbCr to RGB	Conversion from YCoCg to RGB

YCbCr 모델은 빛과 컬러 정보를 한번에 담은 포맷으로, 여기서 Y는 휘도, Cb, Cr은 각각 blue 색차와 green 색차를 나타낸다. 색을 표시하는 또 다른 방법으로는 YCoCg 이 있는데, 여기서 Y는 휘도, Co와 Cg는 각각 주황 색차와 녹색 색차 성분을 의미한다. 이 경우, Y만 전송하면 흑백 화면이 되기 때문에 흑백 화면을 지원가능하고, 3개의 color 성분 중 2개의 색을 알면 1개의 색은 자동으로 구할 수 있기 때문에 데이터 중복을 줄일 수 있다. 위와 같이 YCbCr의 경우에는 conversion이 복잡하므로 YCoCg를 사용하여 computation speed를 늘릴 수 있다. 또한, 각 색 평면 사이에 상관관계가 매우 낮아 여러 encoding 방식으로 사용된다.

3) 구현 1의 두 Sobel filtering 결과에 어떤 차이점이 있는지 비교하시오.

RGB모델과 HSI모델 모두 Sobel filtering 결과 edge 성분이 검출되었으나, RGB space에서의 filtering은 각각의 색상 channel별로 filtering이 이루어지며 특정 channel에서 edge가 검출되더라도 다른 channel에서 검출되지 않을 수 있다. 따라서 하나의 channel에 대한 결과만으로 원래 영상의 edge를 명확하게 검출하는 것이 어려울 수 있다.

그러나 HSI space에서는 RGB 모든 색들이 상관관계를 가져서 각 channel이 구성되어 색상과 명도를 나누어 표현할 수 있기 때문에, intensity 평면에서의 filtering 결과를 통하여 loss가 적은 형태로 원래 영상의 edge를 명확하게 구분해 낼 수 있다.

참조문헌

DIP_Homework5.pdf, 서강대학교 MMI Lab

OpenCV Documentation (https://docs.opencv.org/), Intel Corporation

Gonzalez, Rafael C. Woods, Richard E. - Digital image processing, 4th Edition, Pearson, 2018.