INHA UNIVERSITY

ICE4027 디지털영상처리설계

실습 7주차

보고서 작성 서약서

1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜집기하지 않겠습니다.
2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.
3. 나는 보고서의 내용을 조작하지 않겠습니다.
4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.
5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

나는 보고서 작성시 윤리에 어긋난 행동을 하지 않고 정보통신공학인으로서 나의 명예를 지킬 것을 맹세합니다.

2023년 04월 27일

학부 정보통신공학
학년 3
성명 김동한
학번 12191727

1. 개요

Homework

실습 및 과제

1. 임의의 과일 사진을 입력했을 때 해당 과일의 색을 문자로 출력하고 과일 영역을 컬러로 정확히 추출하는 코드를 구현 (BGR to HSV와 inRange() 함수는 직접 구현할 것)

2. beach.jpg에 대해 군집 간 평균 색으로 segmentation을 수행하는 k-means clustering 수행 (OpenCV 사용, 군집 수인 k에 따른 결과 확인 및 분석)

3. 임의의 과일 사진에 대해 K-means clustering로 segmentation 수행 후, 과일 영역 컬러 추출 수행 (1번 과제 결과와 비교)

1. 구현과정과 결과 분석을 반드시 포함할 것
2. 보고서에도 코드와 실험결과 사진을 첨부할 것
3. 반드시 바로 실행가능한 코드(.cpp)를 첨부할 것
4. 별도의 언급이 없으면 OpenCV가 아닌 직접 구현 함수를 쓸 것

2. 상세 설계 내용

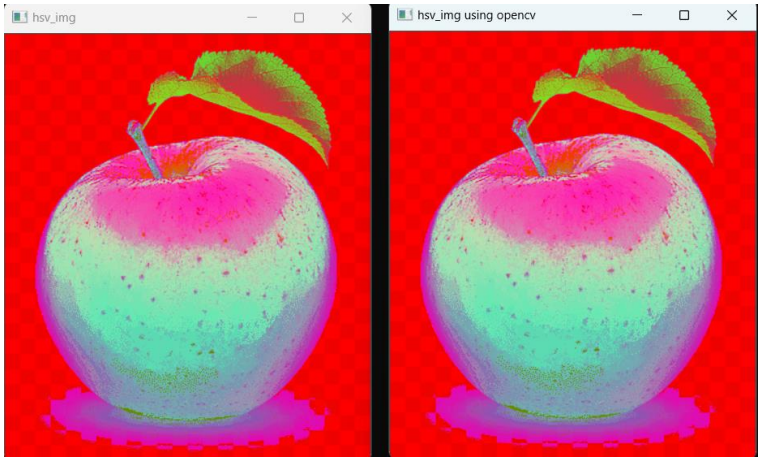
#1

MyBgr2Hsv)

```
Mat MyBgr2Hsv(Mat src_img) {
    double b, g, r, h, s, v;
    Mat dst_img(src_img.size(), src_img.type());
    for (int y = 0; y < src_img.rows; y++)
    {
        for (int x = 0; x < src_img.cols; x++)
        {
            b = (double)src_img.at<Vec3b>(y, x)[0];
            g = (double)src_img.at<Vec3b>(y, x)[1];
            r = (double)src_img.at<Vec3b>(y, x)[2];

            vector<double>vec = { r,g,b };                          //R,G,B 성분을 vector에 저장한다.
            double min = *min_element(vec.begin(), vec.end());      //vector에서 최소값을 대입
            double max = *max_element(vec.begin(), vec.end());      //vecotr에서 최댓값을 대입

            v = max;
            if (v == 0) { s = 0; }                                  //v=0 이면 s=00이다.
            else { s = (max - min) / max; }                         //s=(max-min)/max이다.

            if (max == r) { h = 0 + (g - b) / (max - min); }        //0+(G-B)/(max-min)
            else if (max == g) { h = 2 + (b - r) / (max - min); }   //2+(B-R)/(max-min)
            else { h = 4 + (r - g) / (max - min); }                 //4+(R-G)/(max-min)
            h *= 60;                                                //마지막으로 60 곱해야한다.

            if (h < 0) { h += 360; }        // h<0 이면 h+360
            h /= 2;
            s *= 255;
            h = h > 255.0 ? 255.0 : h < 0 ? 0 : h;
            s = s > 255.0 ? 255.0 : s < 0 ? 0 : s;
            v = v > 255.0 ? 255.0 : v < 0 ? 0 : v;

            dst_img.at<Vec3b>(y, x)[0] = (uchar)h;
            dst_img.at<Vec3b>(y, x)[1] = (uchar)s;
            dst_img.at<Vec3b>(y, x)[2] = (uchar)v;
        }

    }
    return dst_img;
}
```

실습 강의노트에 나와있는 수식대로 구현했다. R,G,B의 값들을 vector에 모두 저장한뒤, 최소값과 최대값을 min_element, max_element를 활용해서 할당해준뒤, 수식에 맞게끔 조건문을 활용해 구현했다.

<원본 이미지>



<좌: 직접 구현한 BGR to HSV 우: opencv 활용한 HSV img>

CvColorModels)

```
void CvColorModels(Mat bgr_img) {
    Mat gray_img, rgb_img, hsv_img, yuv_img, xyz_img;

    cvtColor(bgr_img, gray_img, cv::COLOR_BGR2GRAY);
    cvtColor(bgr_img, rgb_img, cv::COLOR_BGR2RGB);
    cvtColor(bgr_img, hsv_img, cv::COLOR_BGR2HSV);
    cvtColor(bgr_img, yuv_img, cv::COLOR_BGR2YCrCb);
    cvtColor(bgr_img, xyz_img, cv::COLOR_BGR2XYZ);

    Mat print_img;
    bgr_img.copyTo(print_img);
    cvtColor(gray_img, gray_img, cv::COLOR_GRAY2BGR);

    imshow("hsv_img using opencv", hsv_img);
    waitKey(0);
}
```

실습 강의노트에서 opencv를 활용해 HSV image로 변환하는 코드만 활용해 구현했다.

printColor)

```
void printColor(Mat src_img) {
    Mat hsv_img;
    hsv_img = MyBgr2Hsv(src_img);

    double h;

    int red = 0, orange = 0, yellow = 0, green = 0, blue = 0, purple = 0;

    for (int y = 0; y < hsv_img.rows; y++)
    {
        for (int x = 0; x < hsv_img.cols;x++) {
            h = hsv_img.at<Vec3b>(y, x)[0];

            if ((0 < h && h <= 10) || (170 <= h && h <= 180)) { red++; }
            else if (11 < h && h <= 25) { orange++; }
            else if (26 < h && h <= 35) { yellow++; }
            else if (36 < h && h <= 77) { green++; }
            else if (78 < h && h <= 99) { blue++; }
            else if (125 < h && h <= 155) { purple++; }
        }
    }
    int max_count = max(max(max(max(max(red, orange), yellow), green), blue), purple);

    if (max_count = red) { cout << "Red" << endl; }
    else if (max_count = orange) { cout << "Orange" << endl; }
    else if (max_count = yellow) { cout << "Yellow" << endl; }
    else if (max_count = green) { cout << "Green" << endl; }
    else if (max_count = blue) { cout << "Blue" << endl; }
    else if (max_count = purple) { cout << "Purple" << endl; }

}
```

printColor함수를 따로 선언해서 모든 픽셀에 접근하여 h값의 범위에 따라 red부터 purple까지의 counter를 올려준다. 그 뒤, counter가 제일 큰 값인 색깔을 출력하게끔한다. 임의의 과일을 사과로 설정했기 때문에 h의 범위중 0~10혹은 170~180의 값이 제일 많기 때문에 Red를 출력한 모습이다.

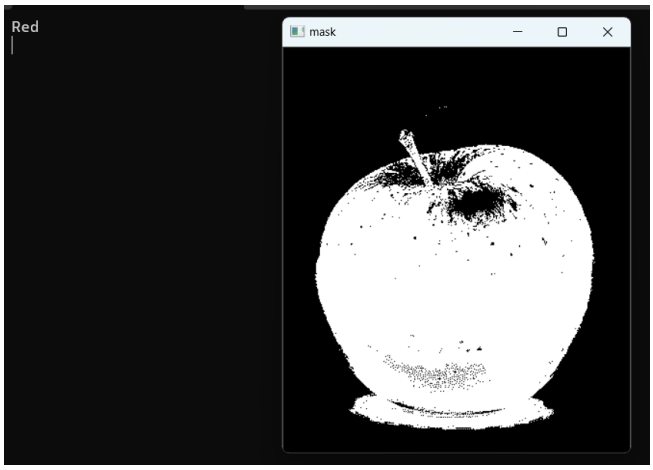
myinRange)

```
Mat myinRange(Mat hsv_img, Scalar lower, Scalar upper) {
    Mat mask = Mat::zeros(hsv_img.size(), CV_8UC3);
    //3channel 검정 배경 영상 생성
    double lower_h = lower[0];
    double lower_s = lower[1];
    double lower_v = lower[2];

    double upper_h = upper[0];
    double upper_s = upper[1];
    double upper_v = upper[2];

    double h, s, v;

    for (int y = 0; y < hsv_img.rows; y++)
    {
        for (int x = 0; x < hsv_img.cols;x++) {
            h = hsv_img.at<Vec3b>(y, x)[0];
            s = hsv_img.at<Vec3b>(y, x)[1];
            v = hsv_img.at<Vec3b>(y, x)[2];

            if (lower_h <= h && h <= upper_h
                && lower_s <= s && s <= upper_s
                && lower_v <= v && v <= upper_v) {
                mask.at<Vec3b>(y, x)[0] = 255;
                mask.at<Vec3b>(y, x)[1] = 255;
                mask.at<Vec3b>(y, x)[2] = 255;
            }
        }
    }
    return mask;
}
```

main)

```
mask = myinRange(dst_img, cv::Scalar(170, 50, 50), cv::Scalar(180, 255, 255));
```

먼저 전체 영상을 hsv_img 크기의 검은색 영상으로 생성한 다음으로, Scalar 클래스를 활용해서 h,s,v의 경계값을 main 함수에서 인자로 전달 받은뒤, 해당 경계값일 경우 흰색의 pixel값을 가지도록 화소의 밝기값을 조절한다. 위 예시는 빨간색에 해당하는 (170,50,50) (180,255,255)를 설정해두었기 때문에, mask 이미지가 이미지의 빨간 부분만 흰색으로 나오고 이외의 부분은 검은색이 나오도록 한다.



그리고 이 마스크를 원본 영상과의 bitwise연산처리를 통해서 컬러 영역만 보이도록 한다.

완벽하진 않지만, 과일의 색깔이 있는 부분만 영상에서 잘라내는 데에 성공했다.

#2

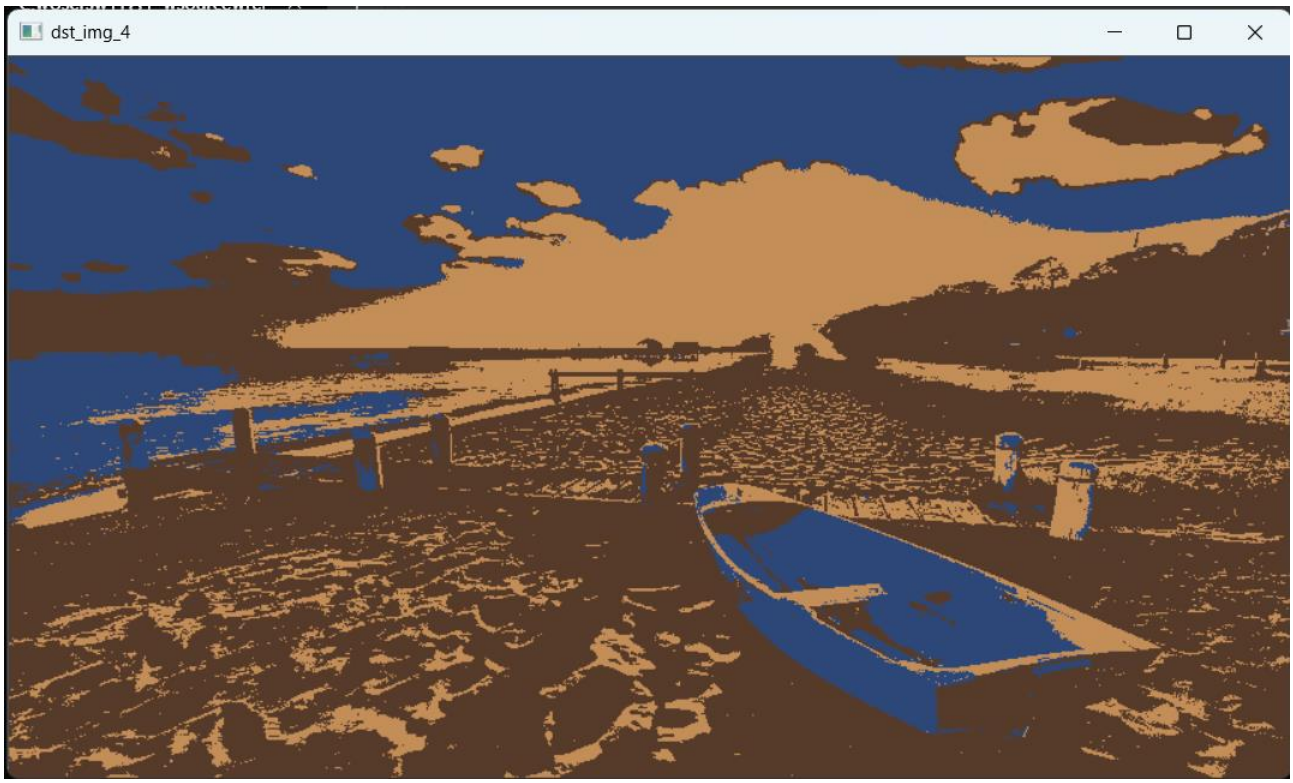
opencv를 활용해서 k-means clustering을 수행한다.

CvKMeans)

```
Mat CvKMeans(Mat src_img, int k) {
    //2차원 영상 -> 1차원 벡터
    Mat samples(src_img.rows * src_img.cols, src_img.channels(), CV_32F);
    for (int y = 0; y < src_img.rows; y++)
    {
        for (int x = 0;x < src_img.cols;x++) {
            if (src_img.channels() == 3)
            {
                for (int z = 0;z < src_img.channels();z++) {
                    samples.at<float>(y + x * src_img.rows, z) = (float)src_img.at<Vec3b>(y, x)[z];
                }
            }
            else
            {
                samples.at<float>(y + x * src_img.rows) = (float)src_img.at<uchar>(y, x);
            }
        }
    }

    // OpenCv K-means 수행

    Mat labels;
    Mat centers;
    int attempts = 5;
    kmeans(samples, k, labels, TermCriteria(CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 10000, 0.0001), attempts, KMEANS_PP_CENTERS, centers);

    Mat dst_img(src_img.size(), src_img.type());
    for (int y = 0; y < src_img.rows; y++)
    {
        for (int x = 0;x < src_img.cols;x++) {
            int cluster_idx = labels.at<int>(y + x * src_img.rows, 0);
            if (src_img.channels()==3)
            {
                for (int z = 0;z < src_img.channels();z++) {
                    dst_img.at<Vec3b>(y, x)[z] = (uchar)centers.at<float>(cluster_idx, z);
                }
            }
            else
            {
                dst_img.at<uchar>(y, x) = (uchar)centers.at<float>(cluster_idx, 0);
            }
        }
    }
    return dst_img;
}
```

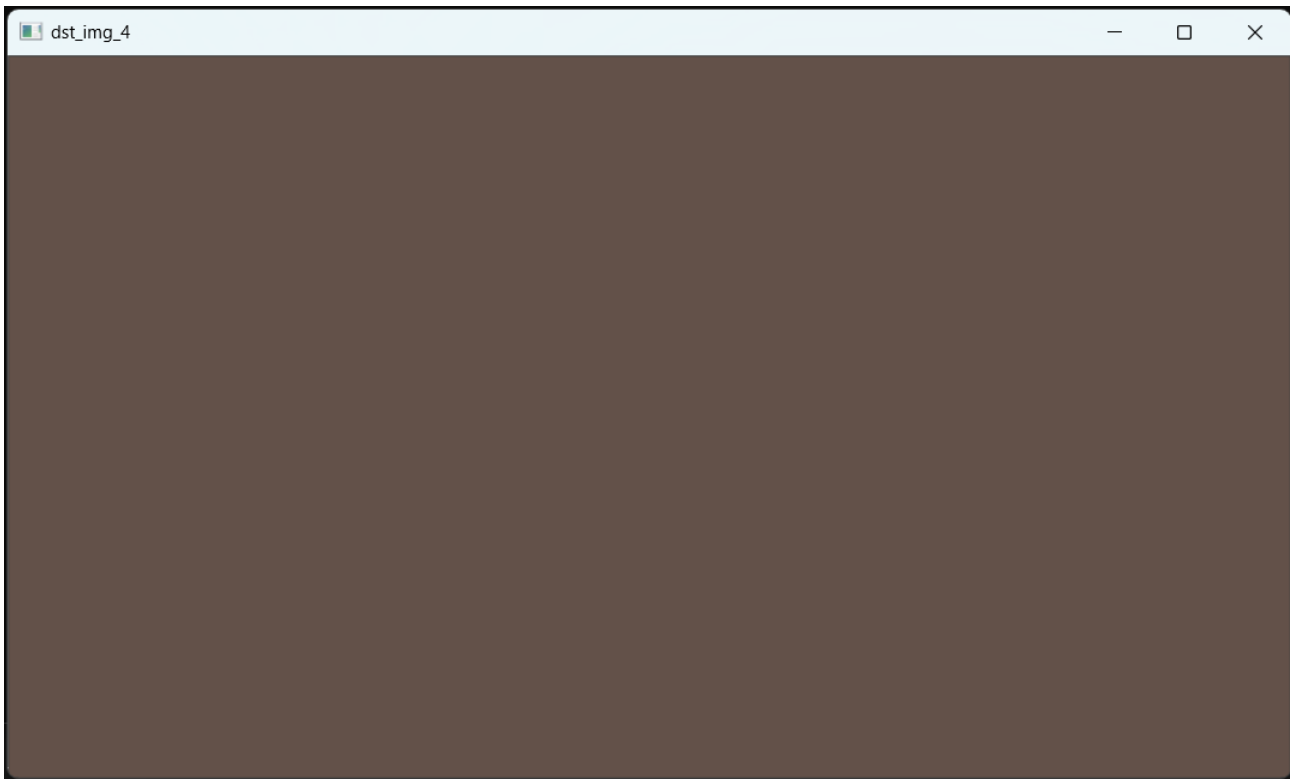
k=5

k=4



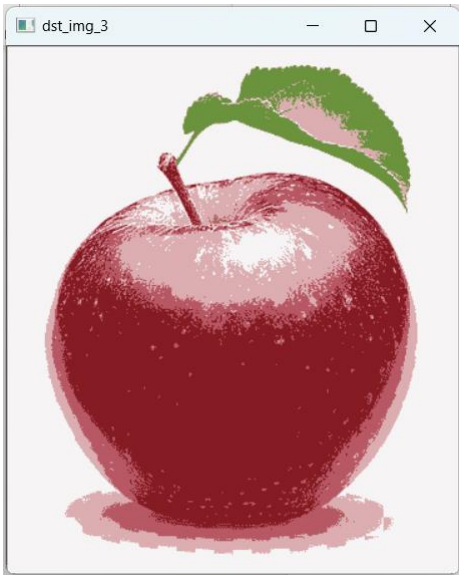
k=3

k=2



k=1

k-means clustering에서 k는 k개의 그룹으로 데이터를 그룹화하는것이다. k를 증가시킬수록 더 다양한 그룹으로 분류하는 것이다. 그래서, 더 정확하게 segmentation이 가능하다. 대신, 연산량이 많아진다. 반대로 k의 값을 너무 작게하면 segmentation이 덜 되는 것을 확인할 수 있었다.



k-means clustering 기법을 활용해 segmentation했다. 기존에 hsv 영상으로의 변환 및 이진화를 이용한 과일 영역 컬러로 출력보다 더 정확하게 segmentation 되는 것을 확인했다.