

# CHAPTER 10



## 컬러 영상 처리

OpenCV 4로 배우는 컴퓨터 비전과 머신 러닝

컴퓨터 비전 기초부터 딥러닝 활용까지!

# 10장 컬러 영상 처리

---

## 10.1 컬러 영상 다루기

## 10.2 컬러 영상 처리 기법

## 10.1 컬러 영상 다루기

---

# 1. 컬러 영상 다루기

## ➤ 컬러 영상의 픽셀 값 참조

- OpenCV에서 영상 파일을 불러와서 Mat 객체를 생성할 때에는 imread() 함수를 사용함
- 이때 imread() 함수의 두 번째 인자를 IMREAD\_COLOR로 설정하면 영상을 3채널 컬러 영상 형식으로 불러옴
- 예를 들어 butterfly.jpg 나비 영상을 3채널 컬러 영상 형식으로 불러오려면 다음과 같이 코드를 작성함

```
Mat img = imread("butterfly.jpg", IMREAD_COLOR);
```

- 일반적으로 컬러 영상은 흔히 RGB라고 부르는 빨간색(R), 녹색(G), 파란색(B) 색상 성분의 조합으로 픽셀 값을 표현함
- OpenCV의 컬러 영상은 기본적으로 RGB 색상 순서가 아니라 BGR 색상 순서로 픽셀 값을 표현함
- imread() 함수로 영상을 3채널 컬러 영상 형식으로 불러오면 각 픽셀의 색상 값이 파란색(B), 녹색(G), 빨간색(R) 순서로 저장된 Mat 객체가 생성됨

# 1. 컬러 영상 다루기

## ➤ 컬러 영상의 픽셀 값 참조

- 컬러 영상에서 각각의 R, G, B 색상 성분은 0부터 255 사이의 값을 가질 수 있음
- 색상 성분 값이 0이면 해당 색상 성분이 전혀 없는 상태임
- 255이면 해당 색상 성분이 가득 차 있음을 의미함
- OpenCV에서 각 색상 성분 값은 uchar 자료형을 사용하여 표현함
- 컬러 영상에서 하나의 픽셀은 세 개의 색상 성분을 가지고 있음
- 컬러 영상의 한 픽셀을 정확하게 표현하려면 Vec3b 자료형을 이용해야 함
- Vec3b 클래스는 크기가 3인 uchar 자료형 배열을 멤버 변수로 가지고 있는 클래스임
- Vec3b 클래스의 바이트 크기는 정확하게 3바이트임
- 실제 3채널 컬러 영상의 한 픽셀이 차지하는 바이트 수와 같음

# 1. 컬러 영상 다루기

## ➤ 컬러 영상의 픽셀 값 참조

- 컬러 영상에서 픽셀 값을 참조할 때에도 `Mat::at()` 함수를 사용함
- `Mat::at()` 함수는 템플릿으로 정의된 함수임
- 3채널 컬러 영상에 대해 `Mat::at()` 함수를 사용하려면 `Vec3b` 자료형을 명시해야 함
- 예를 들어 앞에서 butterfly.jpg 나비 영상을 저장하고 있는 `img` 객체에서 (0, 0) 위치의 픽셀 값을 참조하려면 다음과 같이 코드를 작성함

```
Vec3b& pixel = img.at<Vec3b>(0, 0);
```

# 1. 컬러 영상 다루기

## ➤ 컬러 영상의 픽셀 값 참조

- 앞의 예제 코드에서 Vec3b 참조형으로 선언된 변수 pixel은 img 영상의 (0, 0) 좌표에서 BGR 색상 정보를 가리킴
- Mat::at() 함수가 픽셀 정보를 참조 형태로 반환하기 때문에 변수 pixel값을 변경하면 img 영상의 (0, 0) 좌표 픽셀 값도 같이 변경됨
- Vec3b 클래스는 [] 연산자 재정의를 이용하여 각 멤버 변수 값에 접근할 수 있음
- pixel에 저장된 파란색(B), 녹색(G), 빨간색(R) 색상 성분 값을 각각 알고 싶다면 다음과 같이 코드를 작성함

```
uchar b1 = pixel[0];  
uchar g1 = pixel[1];  
uchar r1 = pixel[2];
```

- 이 예제 코드에서 변수 b1, g1, r1에는 각각 pixel의 파란색, 녹색, 빨간색 성분 값이 저장됨

# 1. 컬러 영상 다루기

## ➤ 컬러 영상의 픽셀 값 참조

- `Mat::ptr()` 함수를 이용하여 컬러 영상의 특정 행 시작 주소를 얻어 올 때에도 `Vec3b` 자료형을 명시하여 사용해야 함
- 예를 들어 `img` 컬러 영상에서 0번째 행 시작 픽셀 주소를 알고 싶다면 다음과 같이 코드를 작성함

```
Vec3b* ptr = img.ptr<Vec3b>(0);
```

- 포인터 변수 `ptr`에 특정 행 시작 주소를 받아 온 후에는 해당 행의 픽셀 값을 `ptr[0]`, `ptr[1]`, ..., `ptr[img.cols-1]` 형태로 접근할 수 있음
- 예를 들어 `ptr[0]`은 (0, 0) 좌표 픽셀을 가리키며, 이는 `Vec3b` 자료형에 해당함
- `ptr[0]` 코드 뒤에 다시 한 번 `[]` 연산자를 붙여서 해당 픽셀의 파란색(B), 녹색(G), 빨간색(R) 색상 성분 값을 얻을 수 있음

```
uchar b2 = ptr[0][0];
```

```
uchar g2 = ptr[0][1];
```

```
uchar r2 = ptr[0][2];
```

- 변수 `b2`, `g2`, `r2`에는 각각 (0, 0) 픽셀의 파란색, 녹색, 빨간색 성분 값이 저장됨



# 1. 컬러 영상 다루기

## ➤ 컬러 영상의 픽셀 값 참조

- 코드 10-1에 나타난 color\_inverse() 함수는 butterfly.jpg 컬러 영상의 모든 픽셀 값을 반전시켜 화면에 나타냄
- 컬러 영상을 반전하려면 B, G, R 세 개의 색상 성분 값을 각각 255에서 빼는 연산을 수행해야 함

코드 10-1 컬러 영상의 픽셀 값 반전 [ch10/ColorOp]

```

01  void color_inverse()
02  {
03      Mat src = imread("butterfly.jpg", IMREAD_COLOR);
04
05      if (src.empty()) {
06          cerr << "Image load failed!" << endl;
07          return;
08      }
09
10      Mat dst(src.rows, src.cols, src.type());
11

```

# 1. 컬러 영상 다루기

## ➤ 컬러 영상의 픽셀 값 참조

코드 10-1 컬러 영상의 픽셀 값 반전 [ch10/ColorOp]

```
12     for (int j = 0; j < src.rows; j++) {
13         for (int i = 0; i < src.cols; i++) {
14             Vec3b& p1 = src.at<Vec3b>(j, i);
15             Vec3b& p2 = dst.at<Vec3b>(j, i);
16
17             p2[0] = 255 - p1[0]; // B
18             p2[1] = 255 - p1[1]; // G
19             p2[2] = 255 - p1[2]; // R
20         }
21     }
22
23     imshow("src", src);
24     imshow("dst", dst);
25
```

# 1. 컬러 영상 다루기

## ➤ 컬러 영상의 픽셀 값 참조

코드 10-1 컬러 영상의 픽셀 값 반전 [ch10/ColorOp]

```
26     waitKey();  
27     destroyAllWindows();  
28 }
```

- 3행 butterfly.jpg 파일을 3채널 BGR 컬러 영상으로 불러와서 src에 저장합니다.
- 10행 반전된 영상을 저장할 dst 영상을 생성합니다. dst 영상의 모든 픽셀 값은 이후 for 반복문에서 설정할 것이므로 초깃값은 따로 지정하지 않았습니다.
- 14~15행 src와 dst 영상의 (i, j) 좌표 픽셀 값을 각각 p1과 p2 변수에 참조로 받아 옵니다.
- 17~19행 p1 픽셀의 세 개 색상 성분 값을 모두 반전시켜 p2 픽셀 값으로 설정합니다.

# 1. 컬러 영상 다루기

## ➤ 컬러 영상의 픽셀 값 참조

- 코드 10-1에서는 가독성을 위해서 src 영상과 dst 영상의 (i, j) 위치 픽셀 값을 각각 Vec3b 자료형 변수 p1과 p2로 참조한 후에 반전하였지만, 이 코드는 다음과 같이 간략하게 바꿔 쓸 수도 있음

```
for (int j = 0; j < src.rows; j++) {  
    for (int i = 0; i < src.cols; i++) {  
        dst.at<Vec3b>(j, i) = Vec3b(255, 255, 255) - src.at<Vec3b>(j, i);  
    }  
}
```

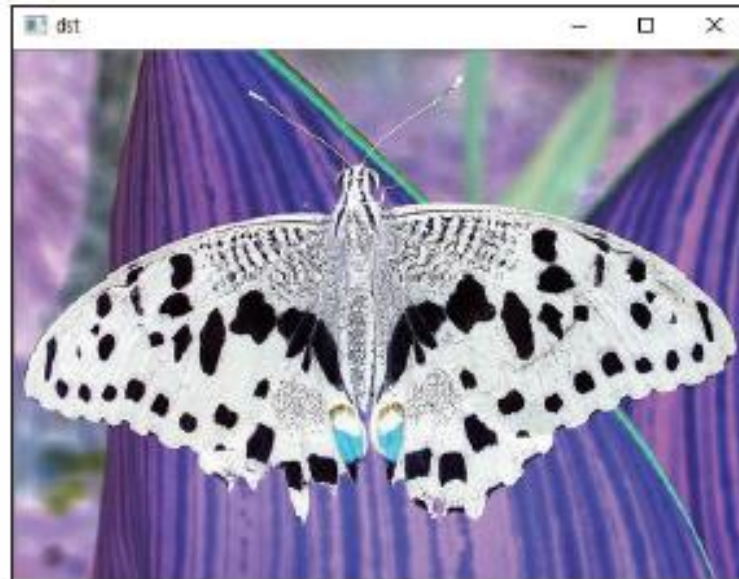
- 이 예제 코드는 B, G, R 색상 성분의 반전을 각각 따로 처리하지 않고, Vec3b 클래스에서 지원하는 - 연산자 재정의를 이용하여 한꺼번에 반전을 수행함

# 1. 컬러 영상 다루기

## ➤ 컬러 영상의 픽셀 값 참조

- 그림 10-1에서 src는 입력 영상 butterfly.jpg 파일이고, dst는 반전된 영상임
- 입력 영상 src에서 검은색 나비 날개 부분이 반전되어 흰색으로 변경된 것을 확인할 수 있음
- 입력 영상에서 배경의 녹색 나뭇잎 영역은 B, G, R 각 채널이 각각 반전되어 결과 영상에서는 보라색으로 변경됨

## ▼ 그림 10-1 컬러 영상의 반전



## 2. 색 공간 변환

### ➤ 색 공간 변환

- 빨간색, 녹색, 파란색 세 가지 색 성분의 조합으로 색을 표현하는 방식을 RGB 색 모델(color model) 또는 RGB 색 공간(color space) 표현이라고 함
- 컬러 영상 처리에서는 보통 색상 구분이 용이한 HSV, HSL 색 공간을 사용함
- 휘도 성분이 구분되어 있는 YCrCb, YUV 등 색 공간을 사용하는 것이 유리함
- OpenCV는 BGR 순서로 색상이 저장된 컬러 영상의 색 공간을 HSV, YCrCB 등 다른 색 공간으로 변환하는 인터페이스를 제공함

## 2. 색 공간 변환

### ➤ 색 공간 변환

- OpenCV에서 영상의 색 공간을 다른 색 공간으로 변환할 때에는 `cvtColor()` 함수를 사용함

```
void cvtColor(InputArray src, OutputArray dst, int code, int dstCn = 0);
```

- `src`      입력 영상. `CV_8U`, `CV_16U`, `CV_32F` 중 하나의 깊이를 사용해야 합니다.
- `dst`      결과 영상. `src`와 크기 및 깊이가 같습니다.
- `code`     색 공간 변환 코드. `ColorConversionCodes` 열거형 상수 중 하나를 지정합니다.
- `dstCn`    결과 영상의 채널 수. 0이면 자동으로 결정됩니다.

## 2. 색 공간 변환

### ➤ 색 공간 변환

- OpenCV는 표 10-1에 나타난 변환 코드 외에도 훨씬 많은 색 공간 변환을 지원함

### ▼ 표 10-1 주요 색 공간 변환 코드

ColorConversionCodes 열거형 상수	설명
COLOR_BGR2RGB 또는 COLOR_RGB2BGR	BGR 채널 순서와 RGB 채널 순서를 상호 변환합니다.
COLOR_BGR2GRAY	3채널 BGR 컬러 영상을 1채널 그레이스케일 영상으로 변환합니다.
COLOR_GRAY2BGR	1채널 그레이스케일 영상을 3채널 BGR 컬러 영상으로 변환합니다.
COLOR_BGR2XYZ	BGR 색 공간을 CIE XYZ 색 공간으로 변환합니다.
COLOR_XYZ2BGR	CIE XYZ 색 공간을 BGR 색 공간으로 변환합니다.



## 2. 색 공간 변환

### ➤ 색 공간 변환

#### ▼ 표 10-1 주요 색 공간 변환 코드

ColorConversionCodes 열거형 상수	설명
COLOR_BGR2YCrCb	BGR 색 공간을 YCrCb 색 공간으로 변환합니다.
COLOR_YCrCb2BGR	YCrCb 색 공간을 BGR 색 공간으로 변환합니다.
COLOR_BGR2HSV	BGR 색 공간을 HSV 색 공간으로 변환합니다.
COLOR_HSV2BGR	HSV 색 공간을 BGR 색 공간으로 변환합니다.
COLOR_BGR2Lab	BGR 색 공간을 CIE Lab 색 공간으로 변환합니다.
COLOR_Lab2BGR	CIE Lab 색 공간을 BGR 색 공간으로 변환합니다.

## 2. 색 공간 변환

### ➤ BGR2GRAY와 GRAY2BGR

- BGR2GRAY 색 공간 변환 코드는 BGR 컬러 영상을 그레이스케일 영상으로 변환할 때 사용함
- 컬러 영상을 그레이스케일 영상으로 변환하는 주된 이유는 연산 속도와 메모리 사용량을 줄이기 위함
- 기본적으로 컬러 영상은 그레이스케일 영상에 비해 3배 많은 메모리를 필요로 함
- 세 개의 채널에 대해 연산을 수행해야 하기 때문에 더 많은 연산 시간이 필요하게 됨
- 입력 영상에서 색상 정보의 활용도가 그리 높지 않은 경우에는 입력 영상을 그레이스케일 영상으로 변환하여 처리하는 것이 효율적임

## 2. 색 공간 변환

### ➤ BGR2GRAY와 GRAY2BGR

- BGR 3채널 컬러 영상을 그레이스케일 영상으로 변환할 때에는 다음 공식을 사용함  

$$Y = 0.299R + 0.587G + 0.114B$$
- 앞의 공식에서 R, G, B는 각각 픽셀의 빨간색, 녹색, 파란색 성분의 값을 나타냄
- Y는 해당 픽셀의 그레이스케일 성분 크기를 나타냄
- BGR2GRAY 색 공간 변환 코드에 의해 만들어지는 결과영상은 CV\_8UC1 타입으로 설정됨
- 반대로 GRAY2BGR 색 공간 변환 코드는 그레이스케일 영상을 BGR 컬러 영상으로 변환할 때 사용함
- 이 경우 결과 영상은 CV\_8UC3 타입으로 결정되고, 각 픽셀의 B, G, R 색상 성분 값은 다음과 같이 결정됨  

$$R = G = B = Y$$
- 그레이스케일 영상 위에 색깔이 있는 선 또는 글씨를 나타내기 위해 미리 그레이스케일 영상을 BGR 컬러 영상으로 변환함
- 기본적으로 그레이스케일 영상에는 색깔 있는 선 또는 글씨를 출력할 수 없기 때문임

## 2. 색 공간 변환

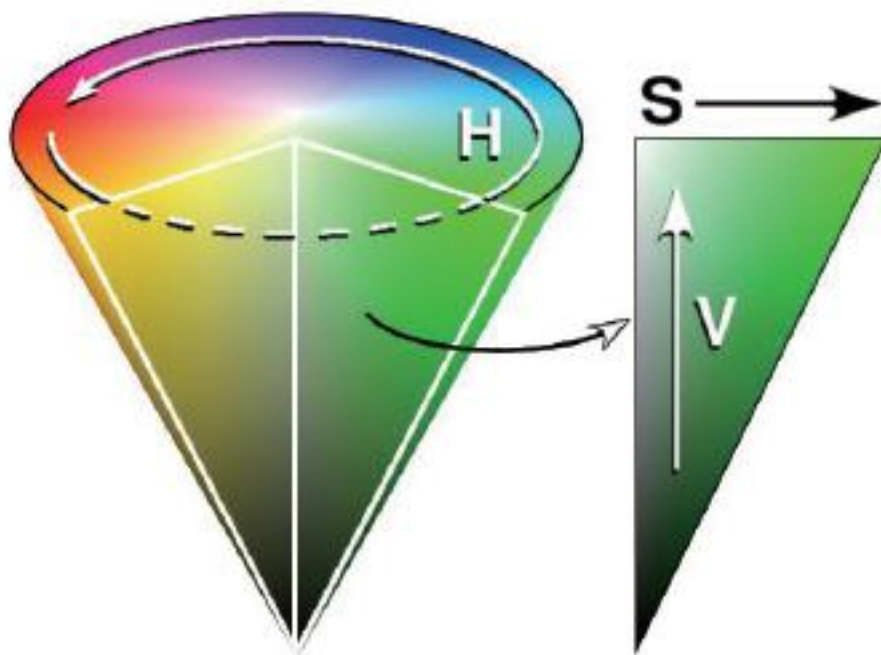
### ➤ BGR2HSV와 HSV2BGR

- HSV 색 모델은 색상(hue), 채도(saturation), 명도(value)로 색을 표현하는 방식임
- 색상은 빨간색, 노란색, 녹색과 같은 색의 종류를 의미합니다. 채도는 색의 순도를 나타냄
- 빨간색에 대하여 채도가 높으면 맑은 선홍색이고, 채도가 낮으면 탁한 빨간색으로 보이게 됨
- 명도는 빛의 세기를 나타냄
- 명도가 높으면 밝고, 명도가 낮으면 어둡게 느껴짐
- HSV 색 공간 모형에서 색상은 원뿔을 가로로 잘랐을 때 나타나는 원형에서 각도로 정의됨
- 각도가  $0^\circ$ 에 해당할 때 빨간색을 나타냄
- 각도가 증가할수록 노란색, 녹색, 하늘색, 파란색, 보라색을 거쳐 각도가  $360^\circ$ 에 가까워지면 다시 빨간색으로 표현됨
- 채도는 원뿔을 가로로 잘랐을 때 나타나는 원 모양의 중심에서 최솟값을 가짐
- 원의 중심에서 방사형으로 멀어지는 방향으로 값이 증가함
- 명도는 원뿔 아래쪽 꼭지점에서 최솟값을 갖고 원뿔의 축을 따라 올라가면서 증가함

## 2. 색 공간 변환

➤ BGR2HSV와 HSV2BGR

▼ 그림 10-2 원뿔 모양의 HSV 색 공간 모형



## 2. 색 공간 변환

### ➤ BGR2HSV와 HSV2BGR

- OpenCV에서 BGR2HSV 색 공간 변환 코드를 이용하여 8비트 BGR 영상을 HSV 영상으로 변환할 경우, H 값은 0부터 179 사이의 정수로 표현되고, S와 V는 0부터 255 사이의 정수로 표현함
- 색상 값은 보통  $0^{\circ}$ 부터  $360^{\circ}$  사이의 각도로 표현함
- uchar 자료형으로는 256 이상의 정수를 표현할 수 없기 때문에 OpenCV에서는 각도를 2로 나눈 값을 H 성분으로 저장함
- 만약 cvtColor() 함수의 입력 BGR 영상이 0에서 1 사이 값으로 정규화된 CV\_32FC3 타입의 행렬임
- H 값은 0에서 360 사이의 실수로 표현되고 S와 V는 0에서 1 사이의 실수 값으로 표현됨

## 2. 색 공간 변환

### ➤ BGR2YCrCb와 YCrCb2BGR

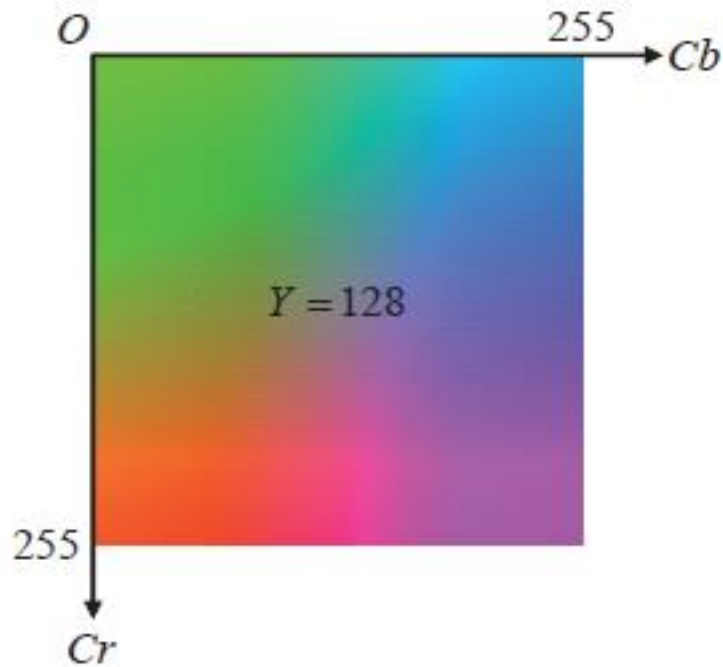
- YCrCb 색 공간에서 Y 성분은 밝기 또는 휘도(luminance) 정보를 나타냄
- Cr과 Cb 성분은 색상 또는 색차(chrominance) 정보를 나타냄
- RGB 색상 성분으로부터 Y 성분을 계산하는 공식은 그레이스케일 계산 공식과 완전히 같음
- Cr과 Cb 성분은 밝기에 대한 정보는 포함하지 않으며 오직 색상에 대한 정보만을 가지고 있음
- YCrCb 색 공간은 영상을 그레이스케일 정보와 색상 정보로 분리하여 처리할 때 유용함
- OpenCV에서 BGR2YCrCb 색 공간 변환 코드를 이용하여 8비트 BGR 영상을 YCrCb 영상으로 변환할 경우, Y, Cr, Cb 각각의 성분 값은 0부터 255 사이의 값으로 표현됨
- 만약 `cvtColor()` 함수의 입력 영상이 0에서 1 사이 값으로 정규화된 `CV_32FC3` 타입의 행렬이라면 Y, Cr, Cb 각각의 성분 값도 0에서 1 사이의 실수 값으로 표현됨

## 2. 색 공간 변환

### ➤ BGR2YCrCb와 YCrCb2BGR

- HSV 색 공간에서는 H 값만을 이용하여 색 종류를 구분할 수 있지만 YCrCb 색 공간에서는 Cr과 Cb를 함께 조합하여 색을 구분할 수 있음

### ▼ 그림 10-3 CrCb 색 분포( $Y=128$ )





## 2. 색 공간 변환

### ➤ BGR2YCrCb와 YCrCb2BGR

- 코드 10-2에 나타난 color\_grayscale() 함수는 butterfly.jpg 영상을 3채널 BGR 컬러 영상 형식으로 불러온 후 그레이스케일 영상으로 변환하여 화면에 나타냄

**코드 10-2** 컬러 영상을 그레이스케일 영상으로 변환하기 [ch10/ColorOp]

```

01  void color_grayscale()
02  {
03      Mat src = imread("butterfly.jpg");
04
05      if (src.empty()) {
06          cerr << "Image load failed!" << endl;
07          return;
08      }
09
10      Mat dst;
11      cvtColor(src, dst, COLOR_BGR2GRAY);
12

```

## 2. 색 공간 변환

### ➤ BGR2YCrCb와 YCrCb2BGR

코드 10-2 컬러 영상을 그레이스케일 영상으로 변환하기 [ch10/ColorOp]

```
13     imshow("src", src);
14     imshow("dst", dst);
15
16     waitKey();
17     destroyAllWindows();
18 }
```

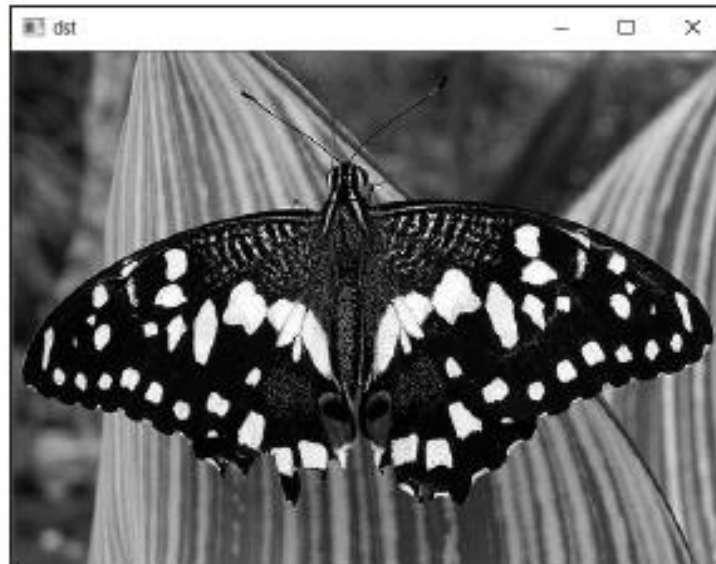
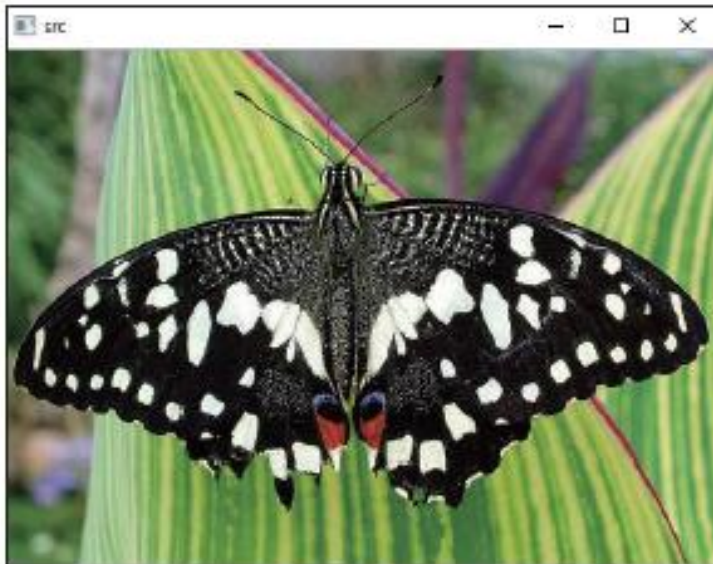
- 3행 `imshow()` 함수의 두 번째 인자를 지정하지 않으면 기본적으로 3채널 BGR 컬러 영상 형식으로 불러옵니다.
- 11행 3채널 BGR 컬러 영상 `src`를 그레이스케일 영상으로 변환하여 `dst`에 저장합니다.

## 2. 색 공간 변환

### ➤ BGR2YCrCb와 YCrCb2BGR

- 그림 10-4에서 src는 입력 영상 butterfly.jpg 파일임
- dst는 그레이스케일 형식으로 변경된 영상임
- cvtColor() 함수로 생성한 dst 영상은 imread("butterfly.jpg", IMREAD\_GRAYSCALE) 코드를 이용하여 나비 영상을 그레이스케일 영상 형식으로 불러오는 것과 완전히 같음

### ▼ 그림 10-4 컬러 영상을 그레이스케일 영상으로 변환하기 예제 실행 결과



# 3. 색상 채널 나누기

## ➤ 색상 채널 나누기

- imread() 함수로부터 생성된 컬러 영상은 하나의 픽셀이 파란색(B), 녹색(G), 빨간색(R) 세 개의 색상 정보를 가지고 있음
- OpenCV에서 컬러 영상은 보통 uchar 자료형을 사용하고 세 개의 채널을 갖는 Mat 객체로 표현함
- 컬러 영상을 다루다 보면 빨간색 성분만을 이용하거나 HSV 색 공간으로 변환한 후 H 성분만을 이용하는 경우가 종종 발생함
- 이러한 경우에는 3채널 Mat 객체를 1채널 Mat 객체 세 개로 분리해서 다루는 것이 효율적임

# 3. 색상 채널 나누기

## ➤ 색상 채널 나누기

- OpenCV에서 다채널 행렬을 1채널 행렬 여러 개로 변환할 때에는 split() 함수를 사용함

```
void split(const Mat& src, Mat* mvbegin);  
void split(InputArray src, OutputArrayOfArrays mv);
```

- **src**        입력 다채널 행렬
- **mvbegin**    분리된 1채널 행렬을 저장할 Mat 배열 주소, 영상 배열 개수는 src 영상 채널 수와 같아야 합니다.
- **mv**        분리된 1채널 행렬을 저장할 벡터

## 3. 색상 채널 나누기

### ➤ 색상 채널 나누기

- split() 함수와 반대로 1채널 행렬 여러 개를 합쳐서 다채널 행렬 하나를 생성하려면 merge() 함수를 사용함

```
void merge(const Mat* mv, size_t count, OutputArray dst);  
void merge(InputArrayOfArrays mv, OutputArray dst);
```

- **mv** 1채널 행렬을 저장하고 있는 배열 또는 벡터. 모든 행렬은 크기와 깊이가 같아야 합니다.
- **count** (mv가 Mat 타입의 배열인 경우) Mat 배열의 크기
- **dst** 출력 다채널 행렬

## 3. 색상 채널 나누기

### ➤ 색상 채널 나누기

- 코드 10-3에 나타난 color\_split() 함수는 candies.png 파일을 3채널 컬러 영상 형식으로 불러온 후 각 채널을 분리함
- 이때 분리된 각 채널은 CV\_8UC1 타입의 그레이스케일 영상이므로 imshow() 함수를 이용하여 화면에 나타낼 수 있음

코드 10-3 BGR 컬러 영상의 채널 나누기 [ch10/ColorOp]

```
01 void color_split()
02 {
03     Mat src = imread("candies.png");
04
05     if (src.empty()) {
06         cerr << "Image load failed!" << endl;
07         return;
08     }
09
```

# 3. 색상 채널 나누기

## ➤ 색상 채널 나누기

**코드 10-3** BGR 컬러 영상의 채널 나누기 [ch10/ColorOp]

```

10     vector<Mat> bgr_planes;
11     split(src, bgr_planes);
12
13     imshow("src", src);
14     imshow("B_plane", bgr_planes[0]);
15     imshow("G_plane", bgr_planes[1]);
16     imshow("R_plane", bgr_planes[2]);
17
18     waitKey();
19     destroyAllWindows();
20 }
```

- 3행 candies.png 영상을 3채널 BGR 컬러 영상 형식으로 불러옵니다.
- 10~11행 src 영상의 채널을 분할하여 bgr\_planes 벡터에 저장합니다. bgr\_planes[0]에는 파란색 색상 평면, bgr\_planes[1]에는 녹색 색상 평면, bgr\_planes[2]에는 빨간색 색상 평면이 저장됩니다.



# 3. 색상 채널 나누기

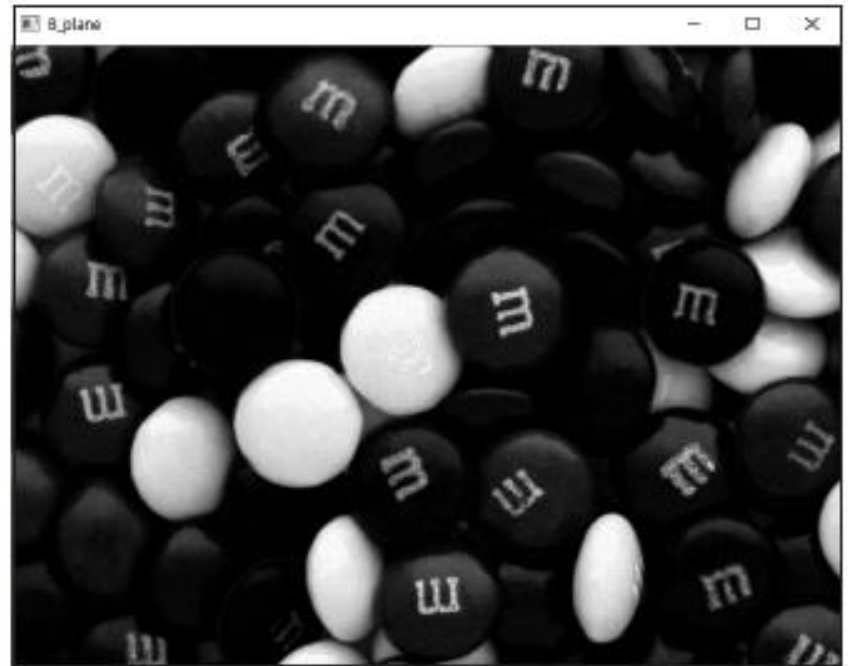
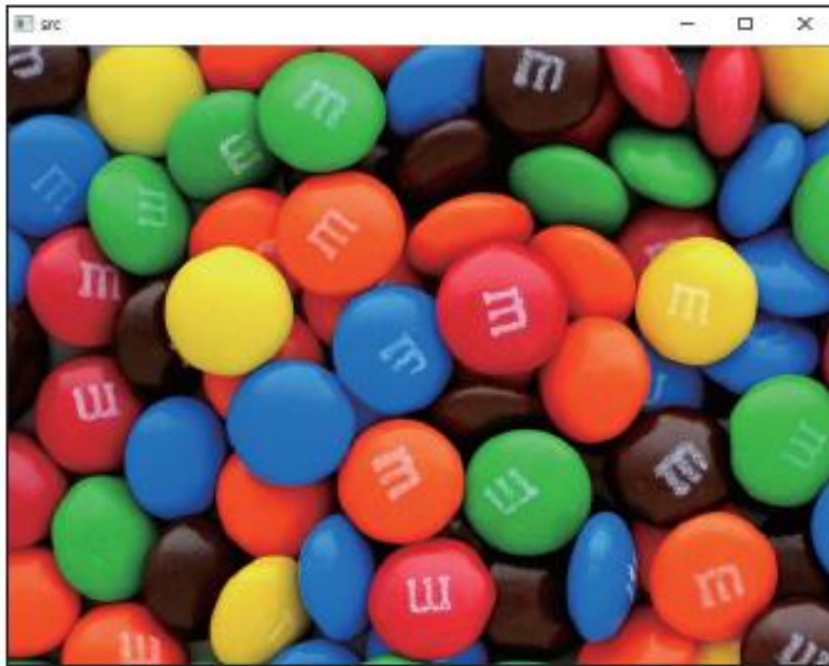
## ➤ 색상 채널 나누기

- 그림 10-5에서 src는 입력 영상 candies.png 파일이고, B\_plane은 파란색 채널, G\_plane은 녹색 채널, R\_plane은 빨간색 채널을 나타내는 영상임
- 원본 영상 src에서 파란색 초콜릿 영역은 파란색 성분 값이 크기 때문에 B\_plane 영상에서 밝은 흰색으로 표시됨
- 원본 영상 src에서 노란색 캔디 영역은 빨간색과 녹색 성분 값이 크기 때문에 R\_plane과 G\_plane 영상에서 밝게 표현됨
- 반면에 B\_plane 영상에서는 어두운 검은색으로 표현되는 것을 확인할 수 있음

# 3. 색상 채널 나누기

## ➤ 색상 채널 나누기

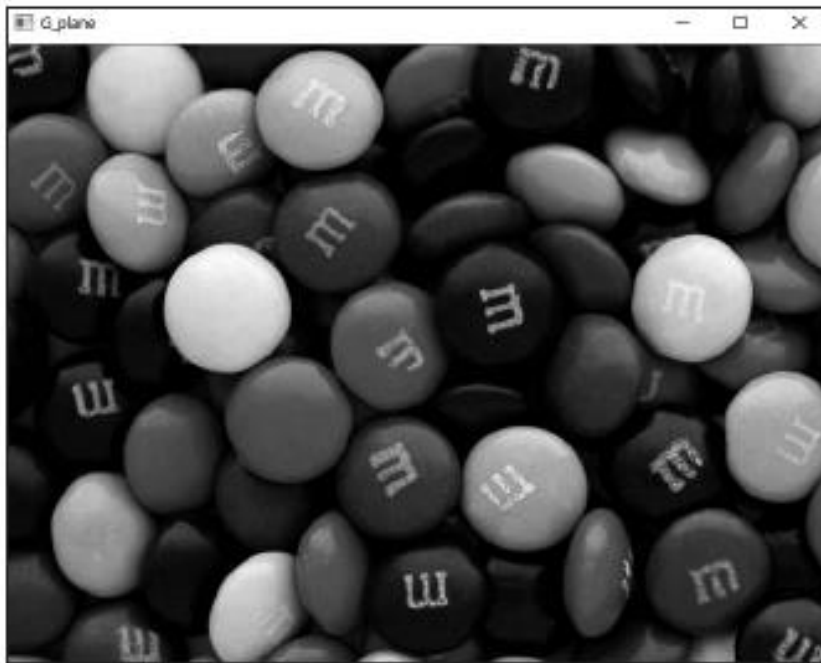
### ▼ 그림 10-5 BGR 컬러 영상의 채널 나누기 예제 실행 결과



# 3. 색상 채널 나누기

## ➤ 색상 채널 나누기

### ▼ 그림 10-5 BGR 컬러 영상의 채널 나누기 예제 실행 결과



## 10.2 컬러 영상 처리 기법

---

# 1. 컬러 영상 처리 기법

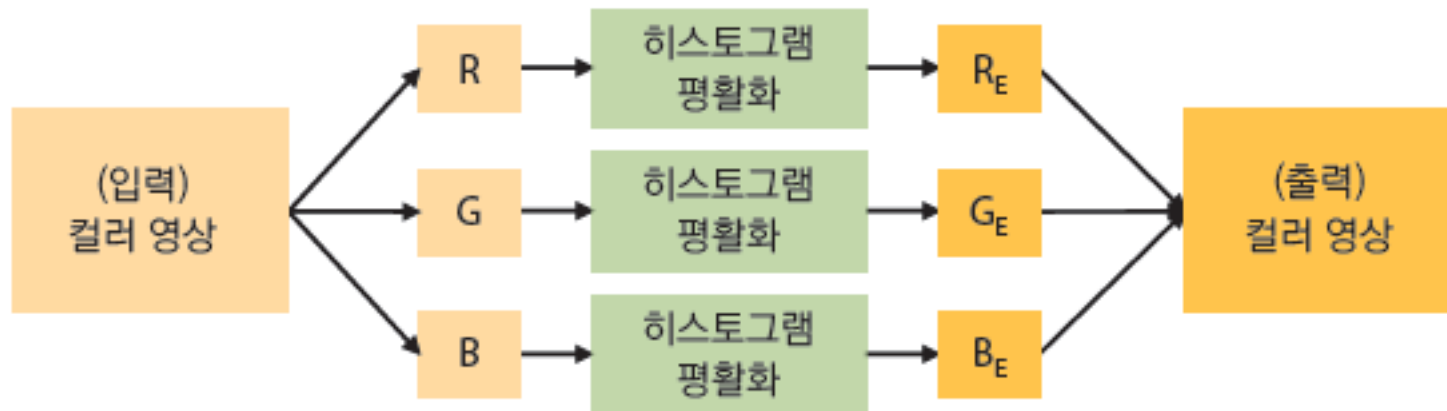
## ➤ 컬러 히스토그램 평활화

- OpenCV에서는 `equalizeHist()` 함수를 통해 히스토그램 평활화를 수행 할 수 있지만 `equalizeHist()` 함수는 그레이스케일 영상만 입력으로 받을 수 있음
- 3채널 컬러 영상에 대해 히스토그램 평활화를 수행하려면 OpenCV 함수를 조합하여 직접 구현해야 함
- 입력 영상을 R, G, B 각 채널로 나누고, 채널별로 히스토그램 평활화를 수행한 후 다시 채널을 합치는 방식임
- 이러한 방식은 R, G, B 색상 채널마다 서로 다른 형태의 명암비 변환 함수를 사용하게 됨으로써 원본 영상과 다른 색상의 결과 영상이 만들어지는 단점이 있음

# 1. 컬러 영상 처리 기법

## ➤ 컬러 히스토그램 평활화

### ▼ 그림 10-6 RGB 각 채널에 히스토그램 평활화 수행하기



# 1. 컬러 영상 처리 기법

## ➤ 컬러 히스토그램 평활화

- 그림 10-7(a)는 테스트로 사용한 입력 영상 pepper.bmp 파일이며, 이 영상은 전체적으로 녹색 성분이 많이 포함되어 있음
- 이 영상에 대해 R, G, B 각각의 색상 채널에 대해 히스토그램 평활화를 수행하고, 다시 채널을 합쳐서 만든 결과 영상이 그림 10-7(b)임
- 입력 영상에서 상대적으로 적었던 파란색 성분이 강해지면서 원본 영상의 색감과 완전히 다른 결과 영상이 만들어짐

## ▼ 그림 10-7 RGB 각 채널에 히스토그램 평활화 수행하기 결과



(a)



(b)

# 1. 컬러 영상 처리 기법

## ➤ 컬러 히스토그램 평활화

- 컬러 영상의 색감은 변경하지 않고 명암비를 높이려면 영상의 밝기 정보만을 사용해야 함
- 보통 컬러 영상에 대하여 히스토그램 평활화를 수행하려면 입력 영상을 밝기 정보와 색상 정보로 분리함
- 밝기 정보에 대해서만 히스토그램 평활화를 수행함
- 예를 들어 YCrCb 색 공간을 사용할 경우, 입력 영상을 YCrCb 색 공간으로 변환하여 Y 성분에 대해서만 히스토그램 평활화를 수행하고 Cr과 Cb 색 성분은 변경하지 않고 그대로 유지함
- 변경된 Y 채널과 Cr, Cb 채널을 다시 합치면 컬러 히스토그램 평활화 결과 영상을 얻을 수 있음

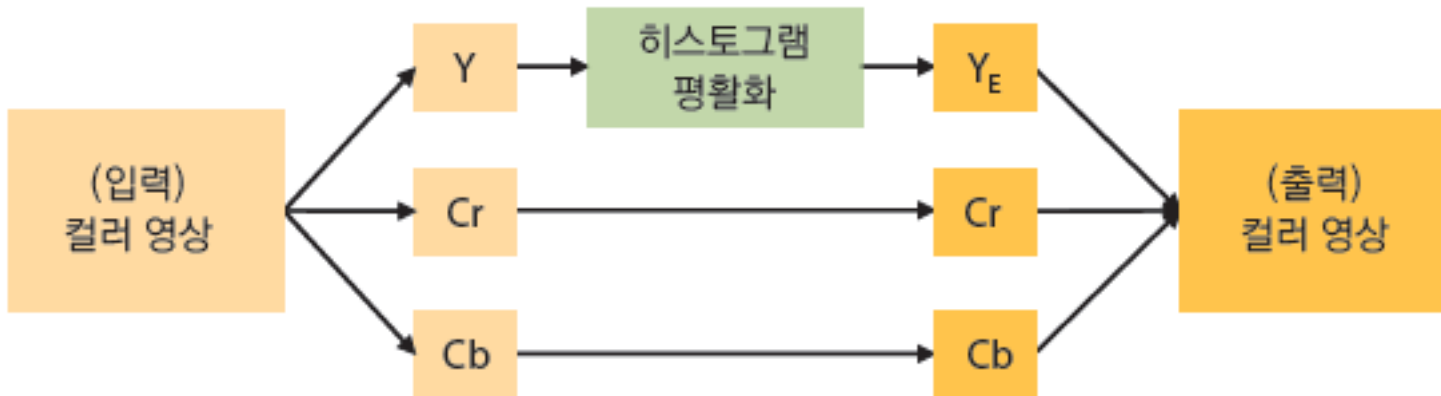


# 1. 컬러 영상 처리 기법

## ➤ 컬러 히스토그램 평활화

- 이러한 방식의 컬러 히스토그램 평활화 방법을 그림 10-8에 그림으로 나타냄
- 이 방식은 Cr과 Cb 채널의 색상 정보는 전혀 변경하지 않으므로 입력 영상의 색감이 그대로 유지되고, 오직 밝기 성분에 대해서만 명암비가 증가하게 됨

### ▼ 그림 10-8 컬러 히스토그램 평활화 방법



# 1. 컬러 영상 처리 기법

## ➤ 컬러 히스토그램 평활화

- 코드 10-4에 나타난 coloeq 예제 프로그램은 pepper.bmp 영상에 대하여 컬러 히스토그램 평활화를 수행하고 그 결과를 화면에 나타냄

**코드 10-4** 컬러 영상의 히스토그램 평활화 예제 [ch10/coloeq]

```

01  #include "opencv2/opencv.hpp"
02  #include <iostream>
03
04  using namespace cv;
05  using namespace std;
06
07  int main(void)
08  {
09      Mat src = imread("pepper.bmp", IMREAD_COLOR);
10
11      if (src.empty()) {
12          cerr << "Image load failed!" << endl;
13          return -1;
14      }
15

```

# 1. 컬러 영상 처리 기법

## ➤ 컬러 히스토그램 평활화

코드 10-4 컬러 영상의 히스토그램 평활화 예제 [ch10/coloreq]

```
16     Mat src_ycrb;  
17     cvtColor(src, src_ycrb, COLOR_BGR2YCrCb);  
18  
19     vector<Mat> ycrb_planes;  
20     split(src_ycrb, ycrb_planes);  
21  
22     equalizeHist(ycrb_planes[0], ycrb_planes[0]); // Y channel  
23  
24     Mat dst_ycrb;  
25     merge(ycrb_planes, dst_ycrb);  
26  
27     Mat dst;  
28     cvtColor(dst_ycrb, dst, COLOR_YCrCb2BGR);  
29
```

# 1. 컬러 영상 처리 기법

## ➤ 컬러 히스토그램 평활화

코드 10-4 컬러 영상의 히스토그램 평활화 예제 [ch10/coloreq]

```

30     imshow("src", src);
31     imshow("dst", dst);
32
33     waitKey(0);
34     return 0;
35 }
```

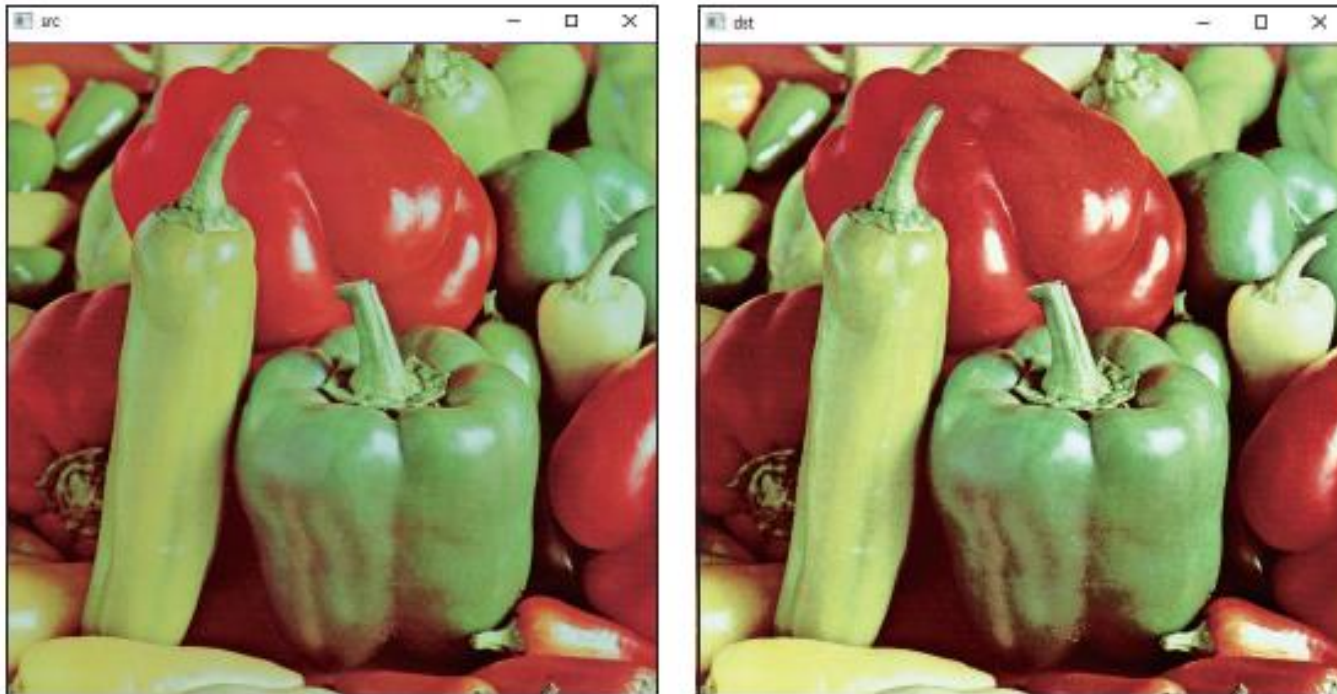
- 9행           pepper.bmp 영상을 3채널 BGR 영상으로 불러와서 src에 저장합니다.
- 16~17행      BGR 색 공간의 src 영상을 YCrCb 색 공간으로 변경하여 src\_ycrCb에 저장합니다.
- 19~20행      src\_ycrCb 영상의 채널을 분리하여 ycrCb\_planes에 저장합니다.
- 22행           Y 성분에 해당하는 ycrCb\_planes[0] 영상에 대해서만 히스토그램 평활화를 수행합니다.
- 24~25행      ycrCb\_planes 벡터에 들어 있는 세 영상을 합쳐서 dst\_ycrCb 영상을 생성합니다.
- 27~28행      dst\_ycrCb 영상의 색 공간을 BGR 색 공간으로 변환하여 dst에 저장합니다.

# 1. 컬러 영상 처리 기법

## ➤ 컬러 히스토그램 평활화

- 그림 10-9에서 src 영상은 pepper.bmp 파일이고, dst 영상은 컬러 히스토그램 평활화가 적용된 결과 영상임
- 원본 영상의 색감은 그대로 유지한 채 명암비가 높아진 것을 확인할 수 있음

## ▼ 그림 10-9 컬러 영상의 히스토그램 평활화 예제 실행 결과



## 2. 색상 범위 지정에 의한 영역 분할

### ➤ 색상 범위 지정에 의한 영역 분할

- 컬러 영상을 다루는 응용에서 자주 요구되는 기법은 특정 색상 영역을 추출하는 작업임
- 예를 들어 입력 영상에서 빨간색 픽셀을 모두 찾아내면 빨간색 객체의 위치와 크기를 알 수 있음
- 컬러 영상에서 빨간색, 파란색 등의 대표적인 색상 영역을 구분할 때에는 RGB 색 공간보다 HSV등의 색상(H) 정보가 따로 설정되어 있는 색 공간을 사용하는 것이 유리함
- 예를 들어 HSV색 공간에서 녹색은 H 값이 60 근방으로 표현되기 때문에 H 값이 60에 가까운지를 조사하여 녹색 픽셀을 찾아낼 수 있음

## 2. 색상 범위 지정에 의한 영역 분할

### ➤ 색상 범위 지정에 의한 영역 분할

- OpenCV에서 행렬의 원소 값이 특정 범위 안에 있는지 확인하려면 `inRange()` 함수를 사용할 수 있음

```
void inRange(InputArray src, InputArray lowerb,  
             InputArray upperb, OutputArray dst);
```

- `src`      입력 영상
- `lowerb`    하한 값. 주로 Mat 또는 Scalar 객체를 지정합니다.
- `upperb`    상한 값. 주로 Mat 또는 Scalar 객체를 지정합니다.
- `dst`      출력 마스크 영상. 입력 영상과 크기가 같고, 타입은 CV\_8UC1입니다.

## 2. 색상 범위 지정에 의한 영역 분할

### ➤ 색상 범위 지정에 의한 영역 분할

- `inRange()` 함수는 입력 영상 `src`의 픽셀 값이 지정한 밝기 또는 색상 범위에 포함되어 있으면 흰색임
- 그렇지 않으면 검은색으로 채워진 마스크 영상 `dst`를 반환함
- 입력 영상 `src`에는 그레이스케일 영상 같은 1채널 행렬과 컬러 영상 같은 다채널 행렬을 모두 지정할 수 있음
- 만약 그레이스케일 영상을 입력 영상으로 사용할 경우, 특정 밝기 값 범위에 있는 픽셀 영역을 추출할 수 있음
- 1채널 영상에 대해 `inRange()` 함수의 동작을 수식으로 표현하면 다음과 같음

$$dst(x, y) = \begin{cases} 255 & lowerb(x, y) \leq src(x, y) \leq upperb(x, y) \text{ 일 때} \\ 0 & \text{그 외} \end{cases}$$



## 2. 색상 범위 지정에 의한 영역 분할

### ➤ 색상 범위 지정에 의한 영역 분할

- 만약 src 영상의 채널이 두 개 이상이라면 입력 영상의 각 채널 값이 모두 지정된 범위를 만족할때 dst 영상의 픽셀 값이 255로 설정됨
- inRange() 함수의 lowerb와 upperb 인자에는 Mat 객체 또는 Scalar 객체를 지정할 수 있음
- 만약 입력 영상 src와 같은 크기의 Mat 객체를 지정할 경우, src의 모든 픽셀에 각기 다른 하한 값과 상한 값을 지정할 수 있음
- 반면에 lowerb와 upperb 인자에 Scalar 객체 또는 int, double 같은 C/C++ 기본 자료형을 지정할 경우에는 src 모든 픽셀에 동일한 하한 값과 상한 값이 적용 됨

## 2. 색상 범위 지정에 의한 영역 분할

### ➤ 색상 범위 지정에 의한 영역 분할

- 코드 10-5에 나타난 `inrange` 예제 프로그램은 영상 출력 창에 두개의 트랙바를 붙여서 사용자가 HSV 색 공간에서 색상의 하한 값과 상한 값을 설정할 수 있도록 함
- 사용자가 설정한 색상 값 범위에 해당하는 영역은 흰색, 그 외의 영역은 검은색으로 표현된 마스크 영상을 화면에 출력함

코드 10-5 `inRange()` 함수를 이용한 특정 색상 영역 분할 [ch10/inrange]

```

01  #include "opencv2/opencv.hpp"
02  #include <iostream>
03
04  using namespace cv;
05  using namespace std;
06
07  int lower_hue = 40, upper_hue = 80;
08  Mat src, src_hsv, mask;
09
10  void on_hue_changed(int, void*);
11
12  int main(int argc, char* argv[])

```

## 2. 색상 범위 지정에 의한 영역 분할

### ➤ 색상 범위 지정에 의한 영역 분할

코드 10-5 inRange() 함수를 이용한 특정 색상 영역 분할[ch10/inrange]

```
13  {
14      src = imread("candies.png", IMREAD_COLOR);
15
16      if (src.empty()) {
17          cerr << "Image load failed!" << endl;
18          return -1;
19      }
20
21      cvtColor(src, src_hsv, COLOR_BGR2HSV);
22
23      imshow("src", src);
24
```

## 2. 색상 범위 지정에 의한 영역 분할

### ➤ 색상 범위 지정에 의한 영역 분할

코드 10-5 inRange() 함수를 이용한 특정 색상 영역 분할[ch10/inrange]

```
25     namedWindow("mask");
26     createTrackbar("Lower Hue", "mask", &lower_hue, 179, on_hue_changed);
27     createTrackbar("Upper Hue", "mask", &upper_hue, 179, on_hue_changed);
28     on_hue_changed(0, 0);
29
30     waitKey(0);
31     return 0;
32 }
33
34 void on_hue_changed(int, void*)
35 {
```

## 2. 색상 범위 지정에 의한 영역 분할

### ➤ 색상 범위 지정에 의한 영역 분할

코드 10-5 inRange() 함수를 이용한 특정 색상 영역 분할 [ch10/inrange]

```
36     Scalar lowerb(lower_hue, 100, 0);
37     Scalar upperb(upper_hue, 255, 255);
38     inRange(src_hsv, lowerb, upperb, mask);
39
40     imshow("mask", mask);
41 }
```

- 7행        두 개의 트랙바 위치를 저장할 정수형 변수 lower\_hue, upper\_hue를 전역 변수로 선언합니다.
- 8행        main() 함수와 트랙바 콜백 함수 on\_hue\_changed() 함수에서 함께 사용할 Mat 객체를 전역 변수로 선언합니다.
- 14행      candies.png 파일을 불러와서 src 변수에 저장합니다.
- 21행      src 영상을 HSV 색 공간으로 변환하여 src\_hsv에 저장합니다.

## 2. 색상 범위 지정에 의한 영역 분할

### ➤ 색상 범위 지정에 의한 영역 분할

- 26~27행    색상의 하한 값과 상한 값을 조절할 수 있는 두 개의 트랙바를 생성합니다. 색상의 최댓값을 179로 설정하고, 두 트랙바의 콜백 함수를 모두 `on_hue_change()` 함수로 설정합니다.
- 28행    프로그램이 처음 실행될 때 영상이 정상적으로 출력되도록 트랙바 콜백 함수를 강제로 호출합니다.
- 36~37행    사용자가 지정한 색상의 하한 값과 상한 값을 이용하여 `lowerb`, `upperb` 객체를 생성합니다. 채도의 범위는 임의로 100부터 255로 설정하였습니다. 명도의 영향은 무시하도록 범위를 0부터 255로 설정하였습니다.
- 38행    `src_hsv` 영상에서 HSV 색 성분 범위가 `lowerb`부터 `upperb` 사이인 위치의 픽셀만 흰색으로 설정한 `mask` 영상을 생성합니다.

## 2. 색상 범위 지정에 의한 영역 분할

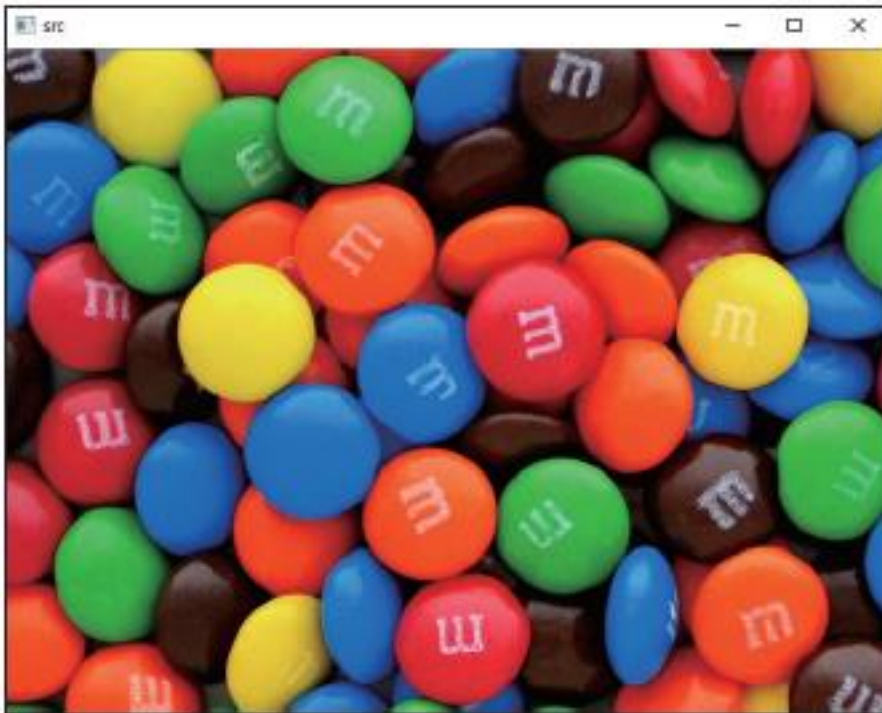
### ➤ 색상 범위 지정에 의한 영역 분할

- 그림 10-10(a)는 입력 영상인 candies.png 파일이고, 그림 10-10(b)는 입력 영상에서 녹색 초콜릿 영역만 찾은 결과임
- 그림 10-10(b)는 입력 영상에서 색상 H의 범위가 40에서 80, 채도 S의 범위가 100에서 255, 명도 V의 범위가 0에서 255 사이인 픽셀 위치를 찾은 결과임
- OpenCV의 HSV 색 공간에서 H 값이 60 근방이면 녹색을 나타냄
- 채도 값이 100보다 큰 픽셀만 찾은 것은 충분히 선명한 녹색만 찾기 위함임
- 명도 범위를 0에서 255사이로 지정한 것은 명도 값이 몇이든지 상관하지 않겠다는 의미임
- 만약 상단의 트랙바를 Lower Hue를 100, Upper Hue를 140으로 설정하면 그림 10-10(c)와 같이 파란색 초콜릿 영역만 찾을 수 있음

## 2. 색상 범위 지정에 의한 영역 분할

➤ 색상 범위 지정에 의한 영역 분할

▼ 그림 10-10 inRange( ) 함수를 이용한 특정 색상 영역 분할 예제 실행 결과



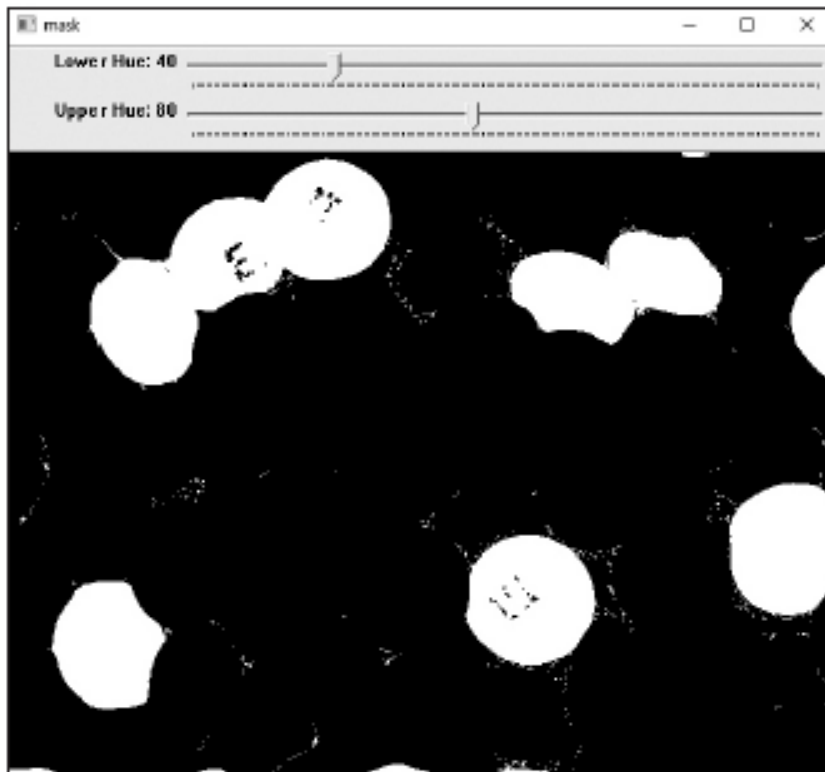
(a)



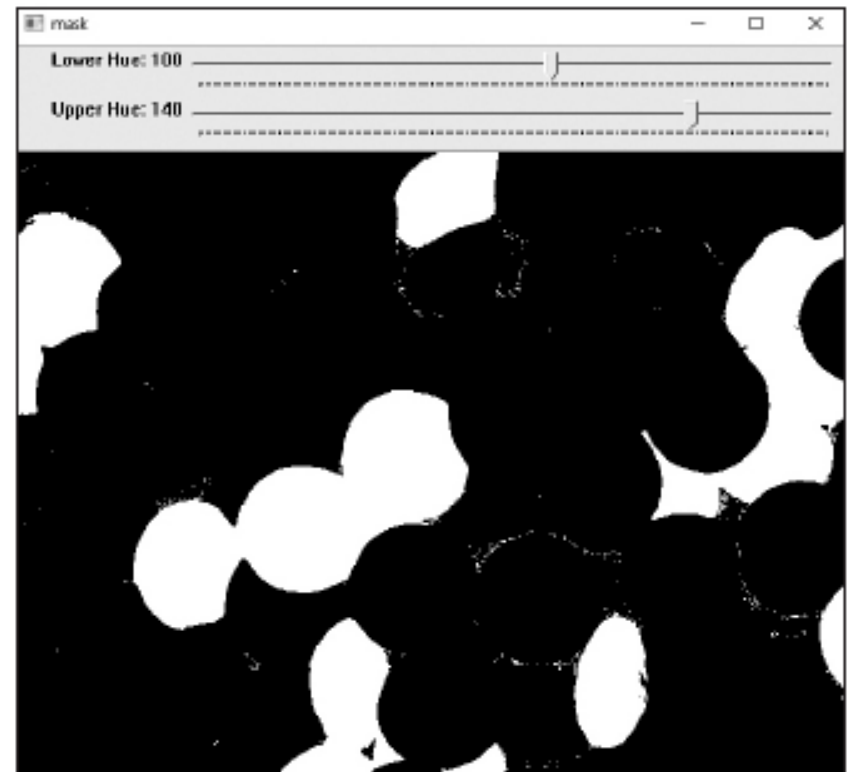
## 2. 색상 범위 지정에 의한 영역 분할

### ➤ 색상 범위 지정에 의한 영역 분할

### ▼ 그림 10-10 inRange( ) 함수를 이용한 특정 색상 영역 분할 예제 실행 결과



(b)



(c)

## 3. 히스토그램 역투영

### ➤ 히스토그램 역투영

- HSV 색 공간에서 H 값을 이용하면 간단하게 특정 색상을 골라낼 수 있어서 편리함
- 다만 이러한 방식은 보통 빨간색, 노란색, 녹색, 파란색처럼 원색에 가까운 색상을 찾기에는 효과적임
- 사람의 피부색처럼 미세한 변화가 있거나 색상 값을 수치적으로 지정하기 어려운 경우에는 적합하지 않음
- 만약 입력 영상에서 찾고자 하는 객체의 기준 영상을 미리 가지고 있다면 컬러 히스토그램 정보를 이용하여 비슷한 색상 영역을 찾을 수 있음
- 기준 영상으로부터 찾고자 하는 객체의 컬러 히스토그램을 미리 구하고 주어진 입력 영상에서 해당 히스토그램에 부합하는 영역을 찾아내는 방식임
- 이처럼 주어진 히스토그램 모델과 일치하는 픽셀을 찾아내는 기법을 히스토그램 역투영(histogram backprojection)이라고 함
- 예를 들어 피부색에 대한 색상 히스토그램을 가지고 있다면 역투영 방법을 사용하여 영상에서 피부색 영역을 검출할 수 있음

# 3. 히스토그램 역투영

## ➤ 히스토그램 역투영

- OpenCV에서 히스토그램 역투영은 `calcBackProject()` 함수를 이용하여 수행할 수 있음

```
void calcBackProject(const Mat* images, int nimages,
                    const int* channels, InputArray hist,
                    OutputArray backProject, const float** ranges,
                    double scale = 1, bool uniform = true);
```

- `images`      입력 영상의 배열 또는 입력 영상의 주소. 영상의 배열인 경우, 모든 영상의 크기와 깊이는 같아야 합니다.
- `nimages`      입력 영상 개수
- `channels`      역투영 계산 시 사용할 채널 번호 배열
- `hist`      입력 히스토그램
- `backProject`      출력 히스토그램 역투영 영상. 입력 영상과 같은 크기, 같은 깊이를 갖는 1채널 행렬입니다.
- `ranges`      각 차원의 히스토그램 빈 범위를 나타내는 배열의 배열
- `scale`      히스토그램 역투영 값에 추가적으로 곱할 값
- `uniform`      히스토그램 빈의 간격이 균등한지를 나타내는 플래그

## 3. 히스토그램 역투영

### ➤ 히스토그램 역투영

- 코드 10-6에 나타난 backproj 예제 프로그램은 기준 영상으로부터 피부색 영역에 대한 히스토그램을 추출하고, 이 히스토그램 정보를 이용하여 입력 영상에서 피부색 영역을 검출함

**코드 10-6** 히스토그램 역투영을 이용한 피부색 영역 검출 예제 [ch10/backproj]

```
01  #include "opencv2/opencv.hpp"
02  #include <iostream>
03
04  using namespace cv;
05  using namespace std;
06
07  int main()
08  {
09      // Calculate CrCb histogram from a reference image
10
```

# 3. 히스토그램 역투영

## ➤ 히스토그램 역투영

코드 10-6 히스토그램 역투영을 이용한 피부색 영역 검출 예제 [ch10/backproj]

```

11     Mat ref, ref_ycrCb, mask;
12     ref = imread("ref.png", IMREAD_COLOR);
13     mask = imread("mask.bmp", IMREAD_GRAYSCALE);
14     cvtColor(ref, ref_ycrCb, COLOR_BGR2YCrCb);
15
16     Mat hist;
17     int channels[] = { 1, 2 };
18     int cr_bins = 128; int cb_bins = 128;
19     int histSize[] = { cr_bins, cb_bins };
20     float cr_range[] = { 0, 256 };
21     float cb_range[] = { 0, 256 };
22     const float* ranges[] = { cr_range, cb_range };
23
24     calcHist(&ref_ycrCb, 1, channels, mask, hist, 2, histSize, ranges);

```

## 3. 히스토그램 역투영

### ➤ 히스토그램 역투영

코드 10-6 히스토그램 역투영을 이용한 피부색 영역 검출 예제 [ch10/backproj]

```
25
26     // Apply histogram backprojection to an input image
27
28     Mat src, src_ycrCb;
29     src = imread("kids.png", IMREAD_COLOR);
30     cvtColor(src, src_ycrCb, COLOR_BGR2YCrCb);
31
32     Mat backproj;
33     calcBackProject(&src_ycrCb, 1, channels, hist, backproj, ranges, 1, true);
34
35     imshow("src", src);
36     imshow("backproj", backproj);
```

# 3. 히스토그램 역투영

## ➤ 히스토그램 역투영

코드 10-6 히스토그램 역투영을 이용한 피부색 영역 검출 예제 [ch10/backproj]

```
37     waitKey(0);
38
39     return 0;
40 }
```

- 12행      피부색 히스토그램 정보를 추출할 기준 영상 ref.png 파일을 불러옵니다.
- 13행      기준 영상에서 피부색이 있는 위치를 흰색으로 표시한 마스크 영상 mask.bmp 파일을 불러옵니다.
- 14행      기준 영상을 YCrCb 색 공간으로 변환합니다.
- 16~24행      기준 영상에서 피부색 영역의 CrCb 2차원 히스토그램을 계산하여 hist에 저장합니다.
- 28~30행      입력 영상 kids.png 파일을 불러와 YCrCb 색 공간으로 변환합니다.
- 32~33행      앞서 구한 히스토그램 hist를 이용하여 입력 영상에서 히스토그램 역투영을 수행합니다. 역투영 결과는 backproj에 저장됩니다.

### 3. 히스토그램 역투영

#### ➤ 히스토그램 역투영

- 그림 10-11(a)는 피부색이 많이 포함된 기준 영상이고, 그림 10-11(b)는 기준 영상에서 실제로 피부색이 나타난 영역만 흰색으로 표시한 마스크 영상임
- 마스크 영상은 포토샵 같은 영상 편집 툴을 이용하여 미리 생성함
- 이 두 영상을 이용하여 기준 영상에서 피부색 영역의 CrCb 히스토그램을 구하고, 그 결과를 그레이스케일 영상 형식으로 나타낸 결과가 그림 10-11(c)임
- 그림 10-11(c)에서 밝게 나타나는 부분이 CrCb 평면에서 피부색을 표현하는 영역임

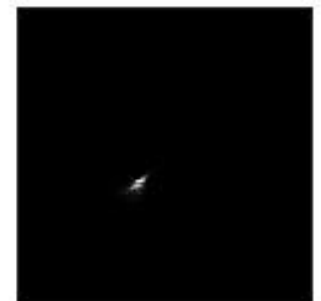
#### ▼ 그림 10-11 기준 영상과 마스크 영상을 이용하여 피부색 CrCb 히스토그램 구하기



(a)



(b)



(c)



## 3. 히스토그램 역투영

### ➤ 히스토그램 역투영

- 그림 10-12에서 src는 입력 영상인 kids.png 파일이고, backproj는 히스토그램 역투영을 통해 구한 피부색 영역임
- calcBackProject() 함수가 반환하는 backproj 영상은 CV\_8UC1 타입이므로 imshow() 함수를 이용하여 쉽게 화면에 나타낼 수 있음
- backproj 영상에서 밝은 회색 또는 흰색으로 표시된 영역은 입력 영상의 픽셀 값이 지정한 히스토그램에서 높은 빈도수로 표현됨을 의미함
- 반대로 backproj 영상에서 어두운 회색 또는 검은색으로 표시된 영역은 해당 위치의 입력 영상 픽셀 값이 지정한 히스토그램에서 빈도수가 낮거나 0임을 나타냄

### 3. 히스토그램 역투영

#### ➤ 히스토그램 역투영

#### ▼ 그림 10-12 히스토그램 역투영을 이용한 피부색 영역 검출 예제 실행 결과

