OpenCV
에지 검출과 응용
&
컬러 영상

수학과 오서영

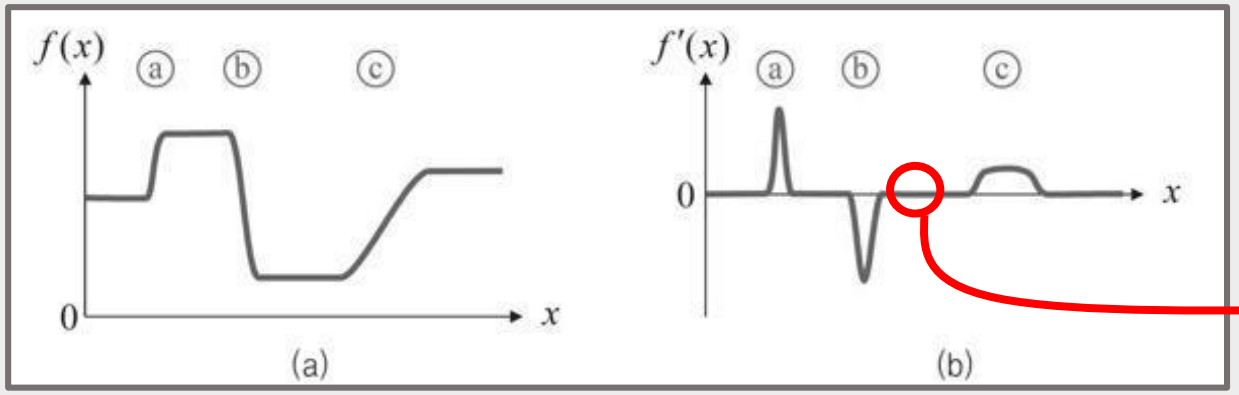
1. 미분과 그래디언트

에지(edge)
: 한쪽 방향으로 픽셀 값이 급격하게 바뀌는 부분
-> 일반적으로 객체와 배경의 경계
- 객체 판별을 위한 전처리로 에지 검출이 사용

에지 검출
: 픽셀 값의 변화율을 측정하여 변화율이 큰 픽셀을 선택
데이터의 변화율 -> 미분(derivative)

$$f' = \frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

1. 미분과 그래디언트



f′(x) = 0 <-> f의 변화 X

f(x) 값이 급격하게 바뀌는 부분을 찾기
-> f′(x) 값이 0보다 훨씬 크거나 작은 위치를 찾기

But 영상은 2차원 평면 위에 픽셀 값이 정형화되지 않은 상태
-> 미분 공식을 적용할 수 없다.

영상으로부터 미분을 계산
1. 영상이 2차원 평면에서 정의된 함수
2. 영상이 정수 단위 좌표에 픽셀이 나열되어 있는 이산함수

1. 미분과 그래디언트

1차원 이산함수에서 미분을 구하는 방법

$$\frac{dI}{dx} \cong \frac{I(x+h)-I(x)}{h}$$

전진 차분(forward difference)

$$\frac{dI}{dx} \cong \frac{I(x)-I(x-h)}{h}$$

후진 차분(backward difference)

$$\frac{dI}{dx} \cong \frac{I(x+h)-I(x-h)}{2h}$$

중앙 차분(centered difference)

2차원 이산함수에서 미분을 구하는 방법 - 영상

2차원 영상 I(x, y)를 **가로 방향**으로 미분
-> x축 방향으로의 **편미분** (partial derivative)
i.e y 좌표는 고정한 상태에서
x축 방향으로만 미분 근사를 계산

$$I_x = \frac{\partial I}{\partial x} \cong \frac{I(x+1,y)-I(x-1,y)}{2}$$

$$I_y = \frac{\partial I}{\partial y} \cong \frac{I(x,y+1)-I(x,y-1)}{2}$$

중앙 차분(centered difference)

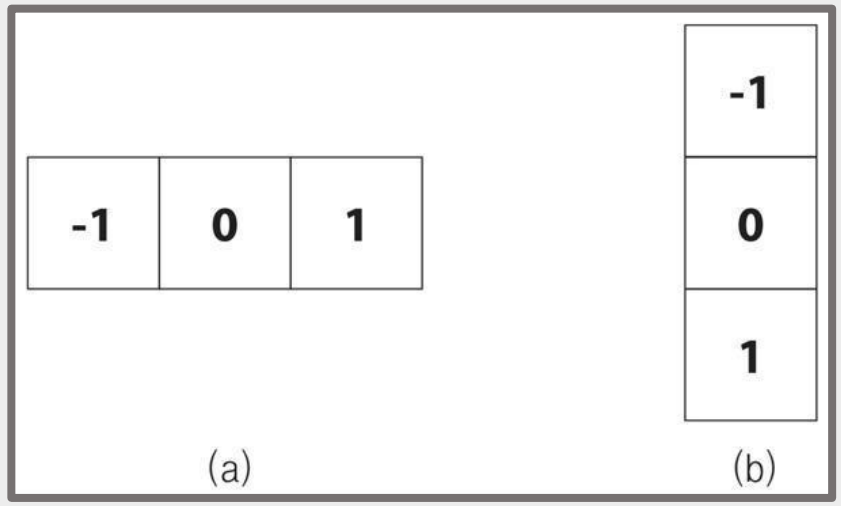
1. 미분과 그래디언트

2차원 이산함수에서 미분을 구하는 방법 - 영상

$$I_x = \frac{\partial I}{\partial x} \cong \frac{I(x+1, y) - I(x-1, y)}{2}$$

$$I_y = \frac{\partial I}{\partial y} \cong \frac{I(x, y+1) - I(x, y-1)}{2}$$

x축과 y축 방향에 대해 편미분을 수행하는 필터 마스크



1. 미분과 그래디언트

x축 방향과 y축 방향으로 각각 편미분

```python
def sobel_derivative():
    src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)

    if src is None:
        print('Image load failed!')
        return

    mx = np.array([[-1, 0, 1],
                   [-2, 0, 2],
                   [-1, 0, 1]], dtype=np.float32)
    my = np.array([[-1, -2, -1],
                   [0, 0, 0],
                   [1, 2, 1]], dtype=np.float32)

    dx = cv2.filter2D(src, -1, mx, delta=128)
    dy = cv2.filter2D(src, -1, my, delta=128)

    cv2.imshow('src', src)
    cv2.imshow('dx', dx)
    cv2.imshow('dy', dy)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

Input dx dy

회색 영역 : 입력 영상에서 픽셀 값 변화가 작은 영역
흰색 또는 검은색 : 픽셀 값이 급격하게 바뀌는 부분

1. 미분과 그래디언트

그래디언트(gradient)
: 2차원 공간에서 정의된 함수 f(x, y),
이 함수의 x축 방향 미분과 y축 방향 미분을 한꺼번에 벡터로 표현한 것

$$\nabla f = \begin{bmatrix} f_x \\ f_y \end{bmatrix} = f_x \mathbf{i} + f_y \mathbf{j}$$

그래디언트 벡터의 **방향**
: 변화 정도가 가장 큰 방향

$$\theta = \tan^{-1}\left(\frac{f_y}{f_x}\right)$$

그래디언트 벡터의 **크기**
: 변화율 세기를 나타내는 척도

$$\|\nabla f\| = \sqrt{f_x^{\,2} + f_y^{\,2}}$$

임계값(threshold)
: 에지 여부를 판단하기 위해 기준이 되는 그래디언트 값

2. 마스크 기반 에지 검출

소벨 필터(Sobel filter) 마스크
: 가장 널리 사용되고 있는 미분 마스크

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

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

x축 방향 미분 마스크
현재 행에 대한 중앙 차분 연산을 2회
이전 행과 다음 행에 대해 중앙 차분 연산을 1
-> 잡음의 영향을 줄이기 위함
-> 현재 행의 중앙 차분 근사에 더 큰 가중치를 주기 위함

2. 마스크 기반 에지 검출

소벨 필터(Sobel filter) 마스크

```python
def sobel_edge():
    src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)

    if src is None:
        print('Image load failed!')
        return

    dx = cv2.Sobel(src, cv2.CV_32F, 1, 0)
    dy = cv2.Sobel(src, cv2.CV_32F, 0, 1)

    fmag = cv2.magnitude(dx, dy)
    mag = np.uint8(np.clip(fmag, 0, 255))
    _, edge = cv2.threshold(mag, 150, 255, cv2.THRESH_BINARY)

    cv2.imshow('src', src)
    cv2.imshow('mag', mag)
    cv2.imshow('edge', edge)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

Input dx dy

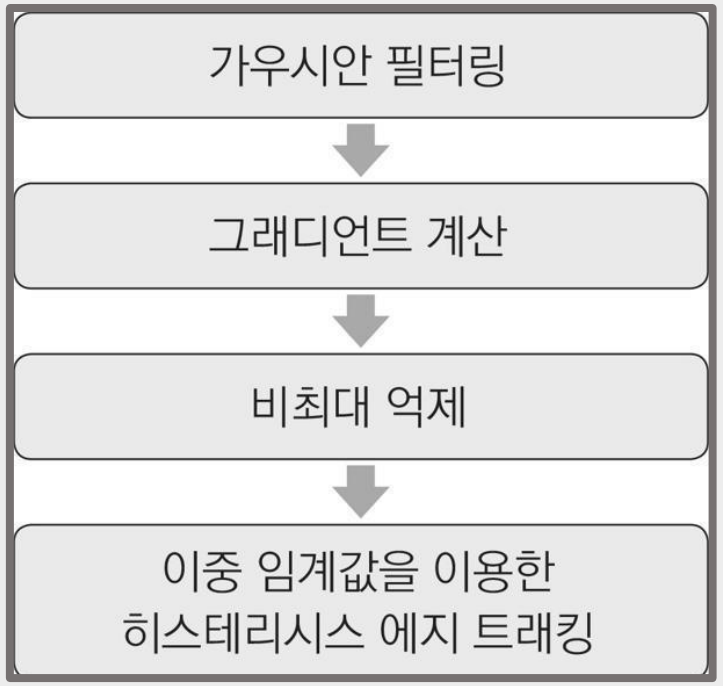
그래디언트 크기가 **150**보다 큰 픽셀은 흰색으로,
그렇지 않은 픽셀은 검은색으로 표현된 이진 영상

그래디언트 크기만을 기준으로 에지 픽셀을 검출
-> 임계값에 민감, 에지 픽셀이 두껍게 표현되는 문제점

3. 캐니 에지 검출기

캐니 에지 검출기
: 에지 검출을 최적화 문제 관점으로 접근함으로써 소벨 에지 검출 방법의 단점을 해결

1. **정확한 검출 :** 에지를 검출하지 못하거나 또는 에지가 아닌데 에지로 검출하는 확률을 최소화
2. **정확한 위치 :** 실제 에지의 중심을 찾아야 합니다.
3. **단일 에지 :** 하나의 에지는 하나의 점으로 표현되어야 합니다.



영상에 포함된 잡음을 제거 -> 에지 세기 감소

소벨 마스크를 사용

에지가 두껍게 표현되는 현상을 방지
가장 변화율이 큰 위치의 픽셀만 에지로 검색

하나의 임계값을 사용할 경우 이분법으로 결과가 판단
-> 환경 변화에 민감

3. 캐니 에지 검출기

캐니 에지 검출

```python
def canny_edge():
    src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)

    if src is None:
        print('Image load failed!')
        return

    dst1 = cv2.Canny(src, 50, 100)
    dst2 = cv2.Canny(src, 50, 150)

    cv2.imshow('src', src)
    cv2.imshow('dst1', dst1)
    cv2.imshow('dst2', dst2)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

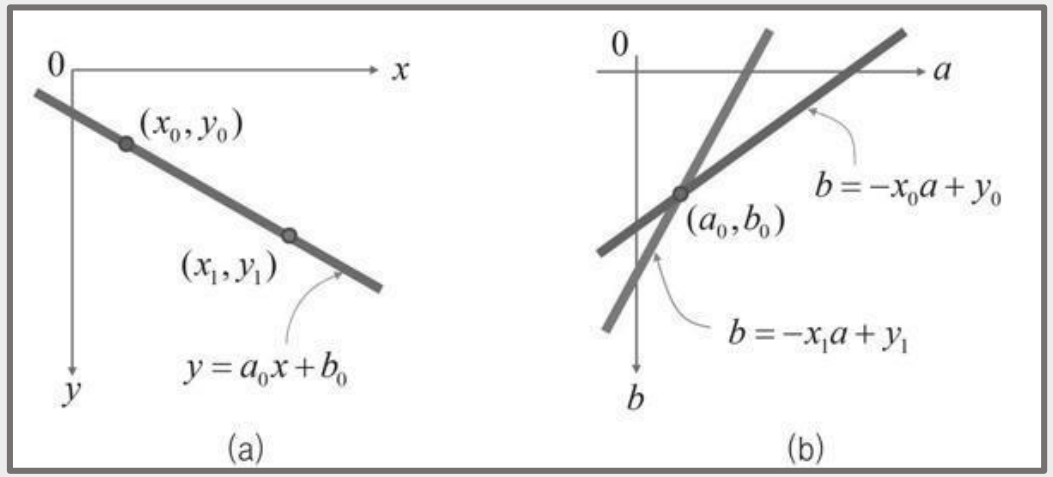
Input 100 150

높은 임계값

임계값을 낮출수록 에지로 판별되는 픽셀이 더 많아짐

4. 허프변환 직선 검출

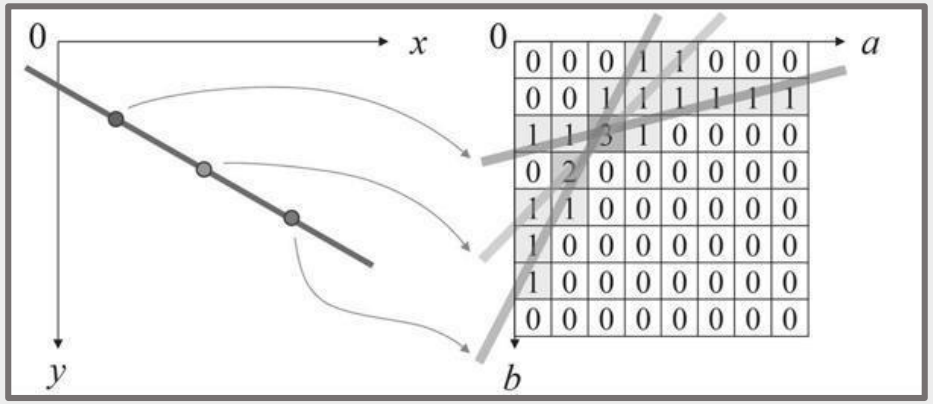
허프 변환(hough transform) 직선 검출
: 2차원 xy 좌표에서 직선의 방정식을 파라미터(parameter) 공간으로 변환하여 직선을 찾는 알고리즘
- 직선을 찾기 위한 용도



xy - 공간에서 에지로 판별된 점을 이용하여
ab - 파라미터 공간에 직선으로 표현
-> 직선이 많이 교차되는 좌표를 모두 찾기

$$y = ax + b \rightarrow b = -xa + y$$

xy-공간 직선의 방정식 **ab-공간 직선의 방정식**

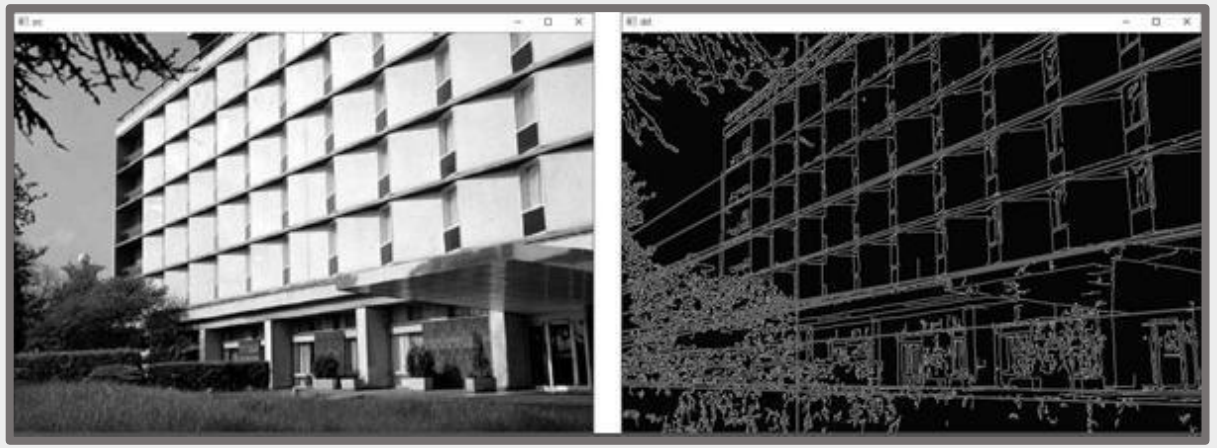


배열 위에서 직선이 지나가는 위치의 원소 값을 1씩 증가시킨 결과를 숫자로 나타냄

4. 허프변환 직선 검출

허프 변환 선분 검출

```python
def hough_lines():
    src = cv2.imread('building.jpg', cv2.IMREAD_GRAYSCALE)

    if src is None:
        print('Image load failed!')
        return

    edge = cv2.Canny(src, 50, 150)
    lines = cv2.HoughLines(edge, 1, math.pi / 180, 250)

    dst = cv2.cvtColor(edge, cv2.COLOR_GRAY2BGR)

    if lines is not None:
        for i in range(lines.shape[0]):
            rho = lines[i][0][0]
            theta = lines[i][0][1]
            cos_t = math.cos(theta)
            sin_t = math.sin(theta)
            x0, y0 = rho * cos_t, rho * sin_t
            alpha = 1000
            pt1 = (int(x0 - alpha * sin_t), int(y0 + alpha * cos_t))
            pt2 = (int(x0 + alpha * sin_t), int(y0 - alpha * cos_t))
            cv2.line(dst, pt1, pt2, (0, 0, 255), 2, cv2.LINE_AA)

    cv2.imshow('src', src)
    cv2.imshow('dst', dst)
    cv2.waitKey()
    cv2.destroyAllWindows()
```



점 두개 -> 직선 하나

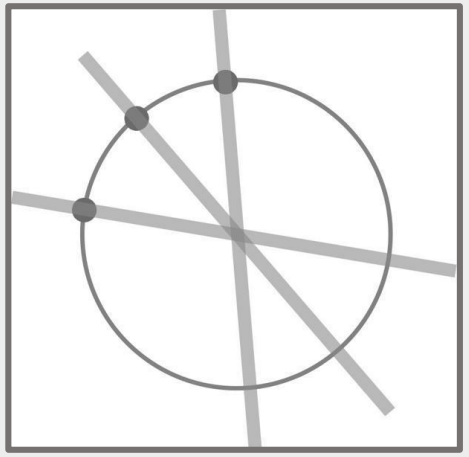
5. 허프변환 원 검출

허프 변환 원 검출
원의 방정식은 세 개의 파라미터 -> 3차원 파라미터 공간 필요 -> 많은 메모리
-> **허프 그래디언트 방법** (Hough gradient method)을 사용

1. 영상에 존재하는 모든 원의 중심 좌표를 찾기
2. 검출된 원의 중심으로부터 원에 적합한 반지름을 구하기

$$(x-a)^2+(y-b)^2=r^2$$

원의 방정식

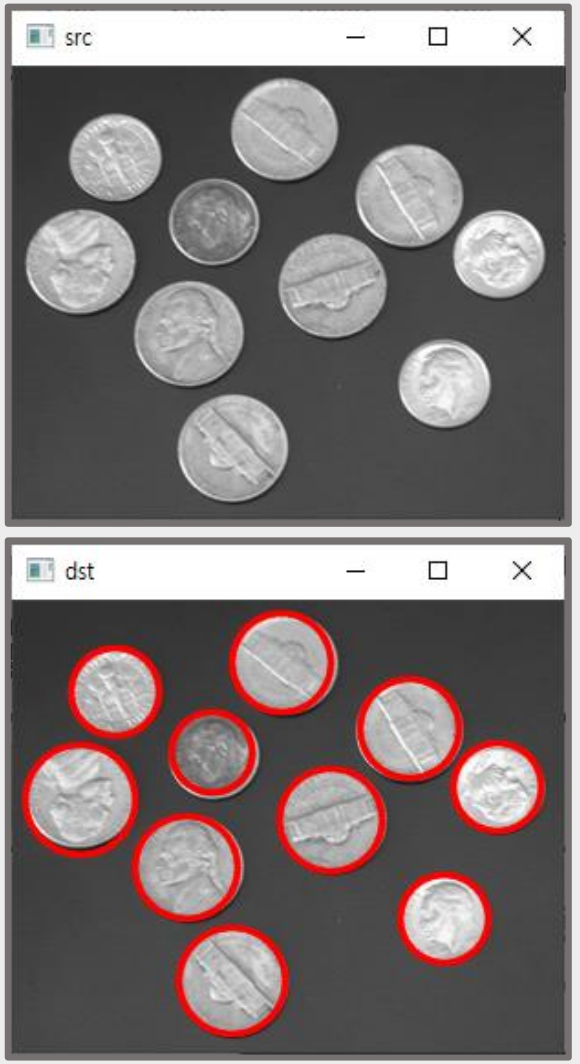


원주상의 모든 점에 대해 그래디언트 방향의 직선을 그리고, 원의 중심을 찾은 후, 다양한 반지름의 원에 대해 원주상에 충분히 많은 에지 픽셀이 존재하는지 확인하여 적절한 반지름을 선택합니다.

5. 허프변환 원 검출

허프 변환 원 검출

```
def hough_circles():
    src = cv2.imread('coins.png', cv2.IMREAD_GRAYSCALE)

    if src is None:
        print('Image load failed!')
        return

    blurred = cv2.blur(src, (3, 3))
    circles = cv2.HoughCircles(blurred, cv2.HOUGH_GRADIENT, 1, 50,
                               param1=150, param2=30)

    dst = cv2.cvtColor(src, cv2.COLOR_GRAY2BGR)

    if circles is not None:
        for i in range(circles.shape[1]):
            cx, cy, radius = circles[0][i]
            cv2.circle(dst, (cx, cy), int(radius), (0, 0, 255), 2, cv2.LINE_AA)

    cv2.imshow('src', src)
    cv2.imshow('dst', dst)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

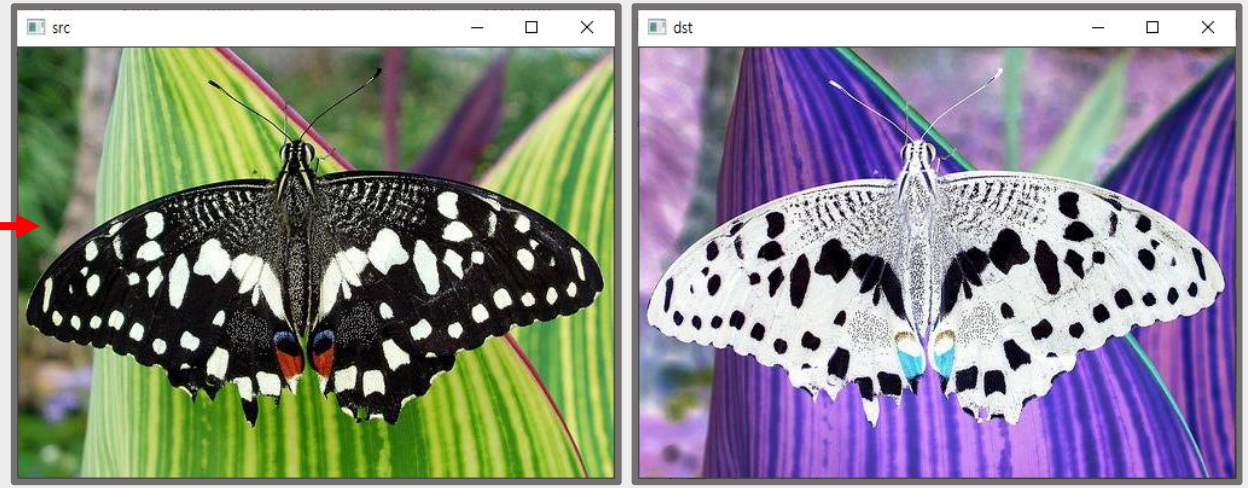


6. 컬러 영상 처리

컬러 영상의 **R, G, B** 색상 성분은 **0**부터 **255** 사이의 값을 가질 수 있다.
색상 성분 값이 0이면 해당 색상 성분이 전혀 없는 상태이고, 255이면 해당 색상 성분이 가득 차 있음을 의미

컬러 영상의 반전

```python
def color_inverse():
    src = cv2.imread('butterfly.jpg', cv2.IMREAD_COLOR)

    if src is None:
        print('Image load failed!')
        return

    dst = np.zeros(src.shape, src.dtype)

    for j in range(src.shape[0]):
        for i in range(src.shape[1]):
            p1 = src[j, i]
            p2 = dst[j, i]

            p2[0] = 255 - p1[0]
            p2[1] = 255 - p1[1]
            p2[2] = 255 - p1[2]
```



녹색 나뭇잎 영역은 B, G, R 각 채널이 각각 반전되어
결과 영상에서는 보라색으로 변경

6. 색 공간 변환

BGR2GRAY 색 공간 변환 코드
: BGR 컬러 영상을 그레이스케일 영상으로 변환할 때 사용
->연산 속도와 메모리 사용량을 줄이기 위함

$$Y = 0.299R + 0.587G + 0.114B$$

Y : 해당 픽셀의 그레이스케일 성분 크기

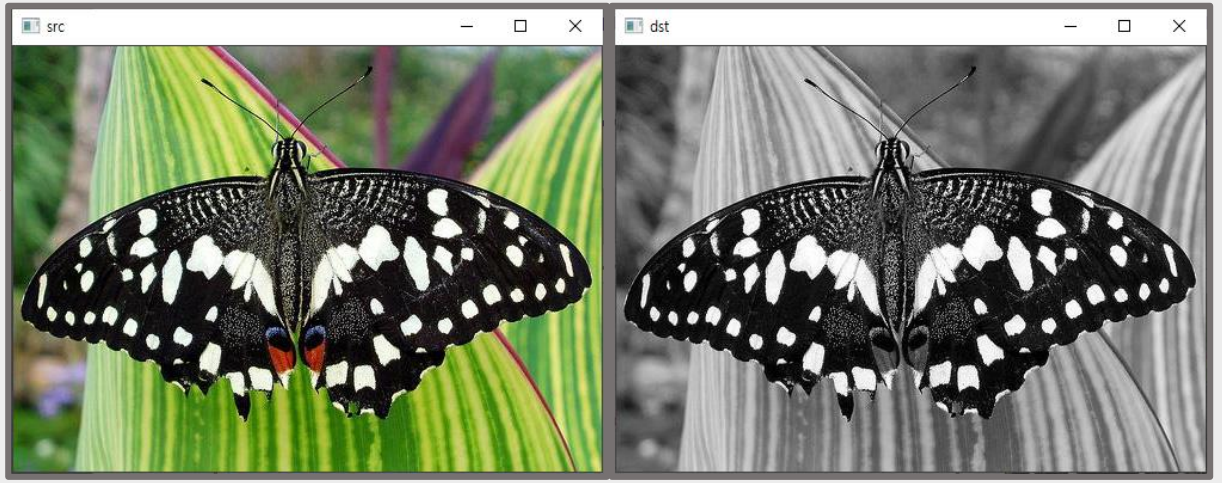
HSV 색 모델
: 색상, 채도, 명도로 색을 표현하는 방식

YCrCb 색 공간
Y 성분 : 밝기 또는 휘도(luminance) 정보
Cr과 Cb 성분 : 색상 또는 색차(chrominance) 정보
- 영상을 그레이스케일 정보와 색상 정보로 분리하여 처리할 때 유용

6. 색 공간 변환

컬러 영상을 그레이스케일 영상으로 변환하기

```
def color_grayscale():
    src = cv2.imread('butterfly.jpg', cv2.IMREAD_COLOR)

    if src is None:
        print('Image load failed!')
        return

    dst = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)

    cv2.imshow('src', src)
    cv2.imshow('dst', dst)
    cv2.waitKey()
    cv2.destroyAllWindows()
```



7. 색상 채널 나누기

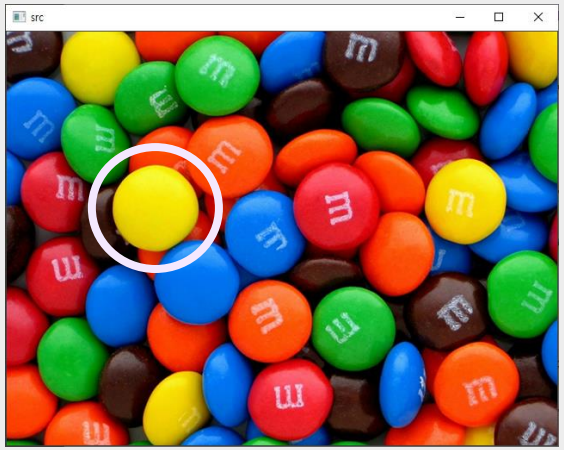
BGR 컬러 영상의 채널 나누기

```
def color_split():
    src = cv2.imread('candies.png', cv2.IMREAD_COLOR)

    if src is None:
        print('Image load failed!')
        return

    b_plane, g_plane, r_plane = cv2.split(src)
    bgr_planes = cv2.split(src)
```

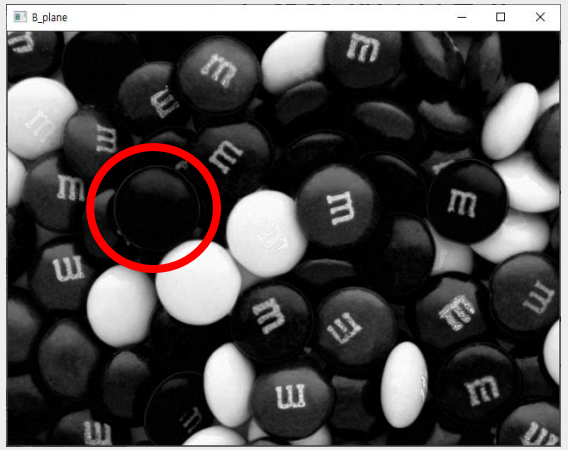
노란색 캔디 영역
: 빨간색과 **녹색** 성분 값이 큼

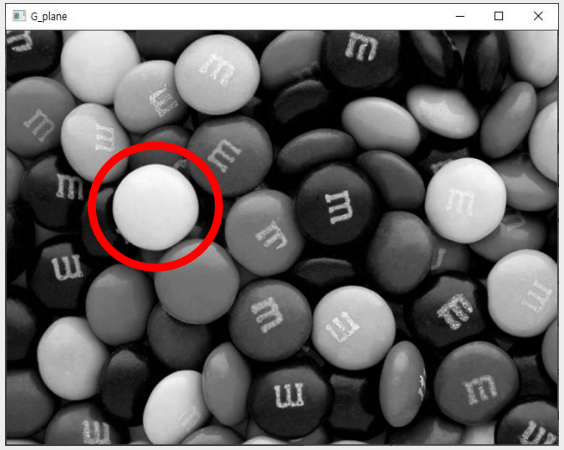
-> R_plane과 G_plane : 밝게 표현
-> B_plane : 어두운 검은색으로 표현

Input

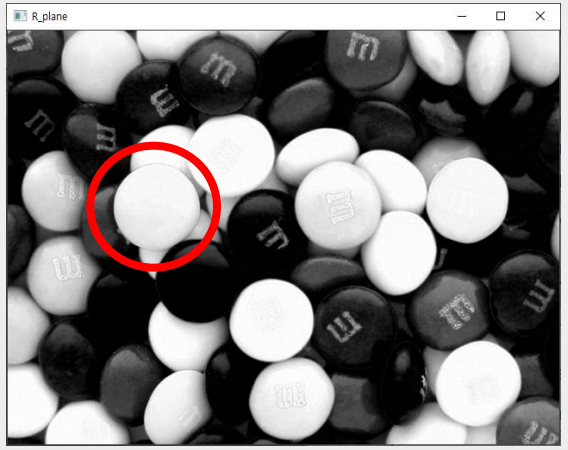


B_plane





G_plane



R_plane