

CH. 7. 영상 처리의 기초

Section 01 관심영역

□ 관심영역 지정

- 전체 이미지에서 연산과 분석의 대상이 되는 영역만을 지정하고 떼어내는 것을 관심영역 (Region Of Interest, ROI)을 지정한다고 한다.
- 관심영역을 지정하기 위해 OpenCV C++ API에서는 별도의 관련 API를 제공하는데, Python API에는 이런 API가 없다.
- 그 이유는 NumPy 슬라이싱(slicing)을 이용하면 따로 API가 없어도 더 편하게 작업할 수 있기 때문이다.
- 전체 이미지가 `img`라는 변수에 있을 때, 관심있는 영역의 좌표가 `x, y`이고 영역의 폭이 `w`, 높이가 `h`라고 하면 이것을 이용하여 관심영역을 지정하는 코드는 다음과 같다.

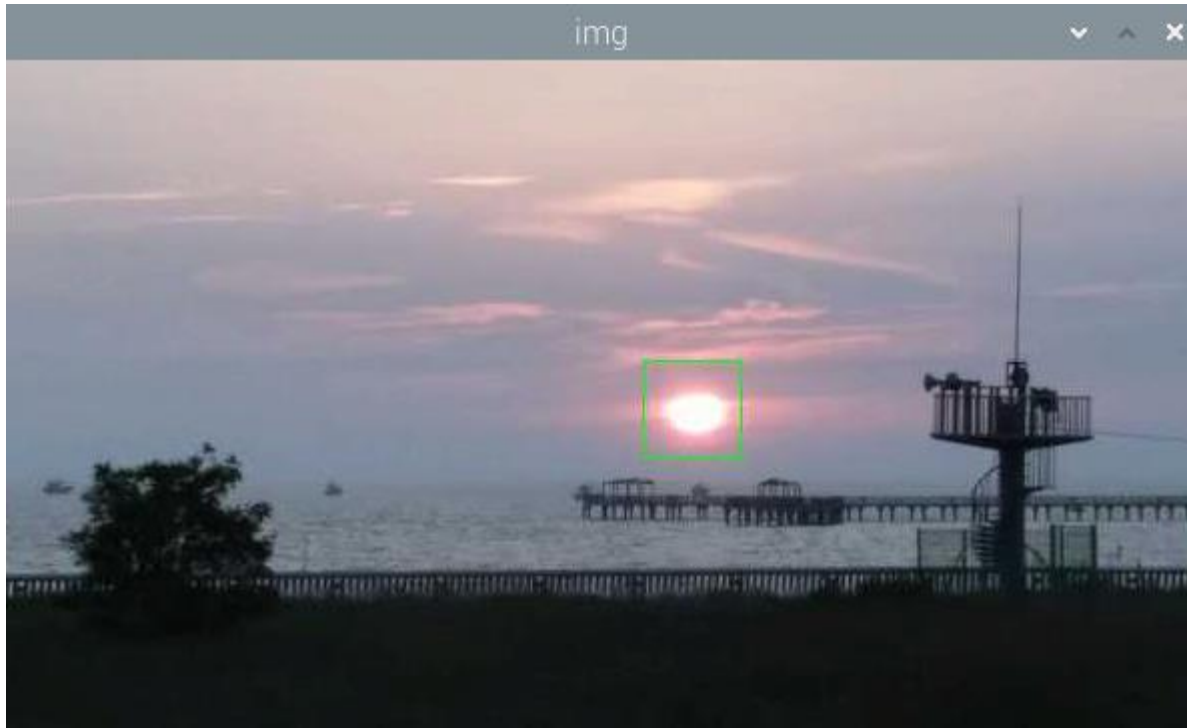
```
roi = img[y:y+h, x:x+w]
```

- 위의 코드는 `img`의 `y`행에서부터 `y+h`행까지, `x`열에서 `x+w`열까지를 슬라이싱한 것이다.

Section 01 관심영역

□ 관심영역 지정

- 다음 그림은 부두의 일몰인데, 일몰 중인 태양을 관심영역으로 지정하고 사각형으로 표시했다.



Section 01 관심영역

□ 관심영역 지정

- 앞의 이미지에서 태양 영역의 시작 좌표는 x :320, y:150이고, 태양 영역의 크기는 50 x 50 이다.
- 앞의 이미지를 나타내는 코드는 다음과 같다.

```
1  import cv2
2
3
4  img = cv2.imread('./img/sunset.jpg')
5
6  x = 320; y = 150; w = 50; h = 50
7
8  roi = img[y:y+h, x:x+w]
```

Section 01 관심영역

□ 관심영역 지정

- 앞의 이미지에서 태양 영역의 시작 좌표는 x :320, y:150이고, 태양 영역의 크기는 50 x 50 이다.
- 앞의 이미지를 나타내는 코드는 다음과 같다.

```
9      print(roi.shape)
10     cv2.rectangle(roi, (0, 0), (h-1, w-1), (0, 255, 0))
11     cv2.imshow("img", img)
12
13     cv2.waitKey(0)
14     cv2.destroyAllWindows()
```

Section 01 관심영역

□ 관심영역 지정

- 다음처럼 간단한 코드를 추가하면 지정한 관심영역을 원본 이미지에 추가해서 태양이 두 개로 보이게 하거나 지정한 관심영역만 새 창에 표시할 수 있다.

```
1 import cv2
2 import numpy as np
3
4 img = cv2.imread('./img/sunset.jpg')
5
6 x = 320; y = 150; w = 50; h = 50
7 roi = img[y:y+h, x:x+w]
8 img2 = roi.copy()
9
```

Section 01 관심영역

□ 관심영역 지정

- 다음처럼 간단한 코드를 추가하면 지정한 관심영역을 원본 이미지에 추가해서 태양이 두 개로 보이게 하거나 지정한 관심영역만 새 창에 표시할 수 있다.

```
10  img[y:y+h, x+w:x+w+w] = roi
11  cv2.rectangle(roi, (x, y), (x+w+w, y+h), (0, 255, 0))
12
13  cv2.imshow("img", img)
14  cv2.imshow("roi", img2)
15
16  cv2.waitKey(0)
17  cv2.destroyAllWindows()
```

Section 01 관심영역

□ 마우스로 관심영역 지정

- 다음 예제는 마우스 이벤트를 처리를 적용해서 마우스로 관심영역을 지정하고 잘라낸 부분만 새 창에 표시하고 파일로 저장하는 예제이다.

```
1 import cv2
2 import numpy as np
3
4 isDragging = False
5 x0, y0, w, h = -1, -1, -1, -1
6 blue, red = (255, 0, 0), (0, 0, 255)
7
8 def onMouse(event, x, y, flags, param):
9     global isDragging, x0, y0, img
10    if event == cv2.EVENT_LBUTTONDOWN:
11        isDragging = True
12        x0 = x
13        y0 = y
14    elif event == cv2.EVENT_MOUSEMOVE:
15        if isDragging:
16            img_draw = img.copy()
17            cv2.rectangle(img_draw, (x0, y0), (x, y), blue, 2)
```


Section 01 관심영역

□ 마우스로 관심영역 지정

- 다음 예제는 마우스 이벤트를 처리를 적용해서 마우스로 관심영역을 지정하고 잘라낸 부분만 새 창에 표시하고 파일로 저장하는 예제이다.

```
19 elif event == cv2.EVENT_LBUTTONDOWN:  
20     if isDragging:  
21         isDragging = False  
22         w = x - x0  
23         h = y - y0  
24         print("x: %d, y: %d, w: %d, h: %d" %(x0, y0, w, h))  
25         if w > 0 and h > 0:  
26             img_draw = img.copy()  
27             cv2.rectangle(img_draw, (x0, y0), (x, y), red, 2)  
28             cv2.imshow('img', img_draw)  
29             roi = img[y0:y0+h, x0:x0+w]  
30             cv2.imshow('cropped', roi)  
31             cv2.moveWindow('cropped', 0, 0)  
32             print("cropped.")  
33         else:  
34             cv2.imshow('img', img)  
35             print("좌측 상단에서 우측 하단으로 영역을 드래그하세요.")
```

Section 01 관심영역

□ 마우스로 관심영역 지정

- 다음 예제는 마우스 이벤트를 처리를 적용해서 마우스로 관심영역을 지정하고 잘라낸 부분만 새 창에 표시하고 파일로 저장하는 예제이다.

```
36  
37     img = cv2.read('./img/sunset.jpg')  
38     cv2.imshow('img', img)  
39     cv2.setMouseCallback('img', onMouse)  
40     cv2.waitKey(0)  
41     cv2.destroyAllWindows()
```

Section 01 관심영역

□ 마우스로 관심영역 지정

- OpenCV 3는 관심영역을 지정하기 위한 새로운 함수를 제공한다. 이 함수를 사용하면 마우스 이벤트 처리를 위한 코드 없이도 마우스로 간단히 ROI를 지정할 수 있다.

```
ret=cv2.selectROI([win_name,] img[, showCrossHair=True, fromCenter=False])
```

- win_name : ROI 선택을 진행할 창의 이름, str
 - img : ROI 선택을 진행할 이미지, NumPy ndarray
 - showCrossHair : 선택 영역 중심에 십자 모양 표시 여부
 - fromCenter : 마우스 시작 지점을 영역의 중심으로 지정
 - ret : 선택한 영역 좌표와 크기 (x, y, w, h), 선택을 취소한 경우 모두 0
- cv2.selectROI() 함수의 win_name에 창 이름을 지정하고 ROI 선택에 사용할 이미지를 img에 전달하면 마우스로 영역을 선택할 수 있다.
 - 영역을 선택하고 나서 키보드의 스페이스 또는 엔터 키를 누르면 선택한 영역의 x, y 좌표와 영역의 폭과 높이를 튜플에 담아 반환한다.

Section 01 관심영역

□ 마우스로 관심영역 지정

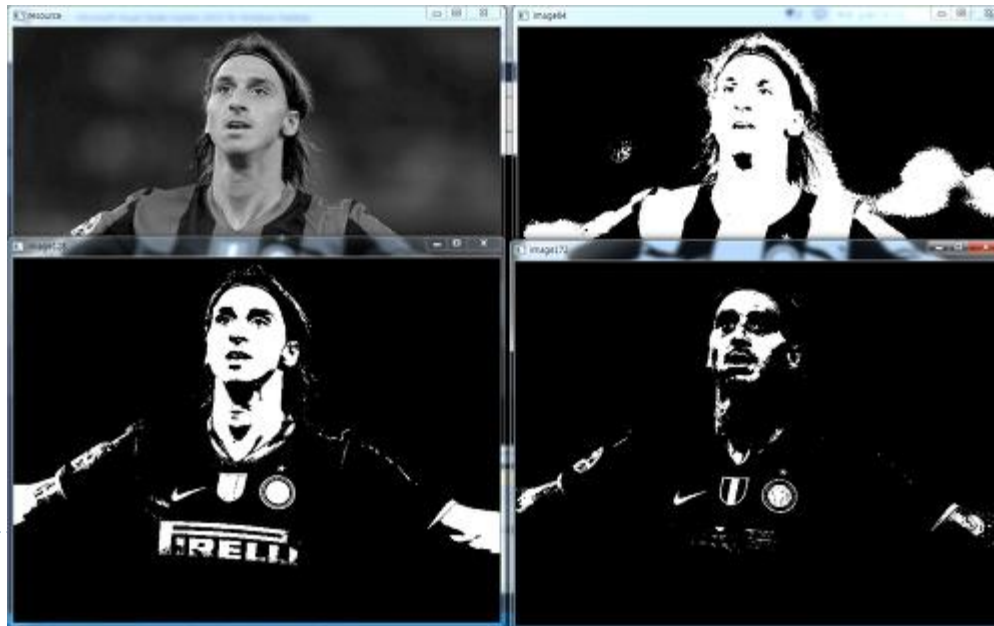
- 다음 예제는 cv2.selectROI() 함수를 이용하여 관심영역을 지정한 예제이다.

```
1  import cv2
2  import numpy as np
3
4  img = cv2.imread('./img/sunset.jpg')
5
6  x, y, w, h = cv2.selectROI('img', img, False)
7  if w and h:
8      roi = img[y:y+h, x:x+w]
9      cv2.imshow('cropped', roi)
10     cv2.moveWindow('cropped', 0, 0)
11     cv2.imwrite('./cropped2.jpg', roi)
12
13     cv2.imshow('img', img)
14     cv2.waitKey(0)
15     cv2.destroyAllWindows()
```

Section 02 컬러 스페이스

□ 디지털 영상의 종류

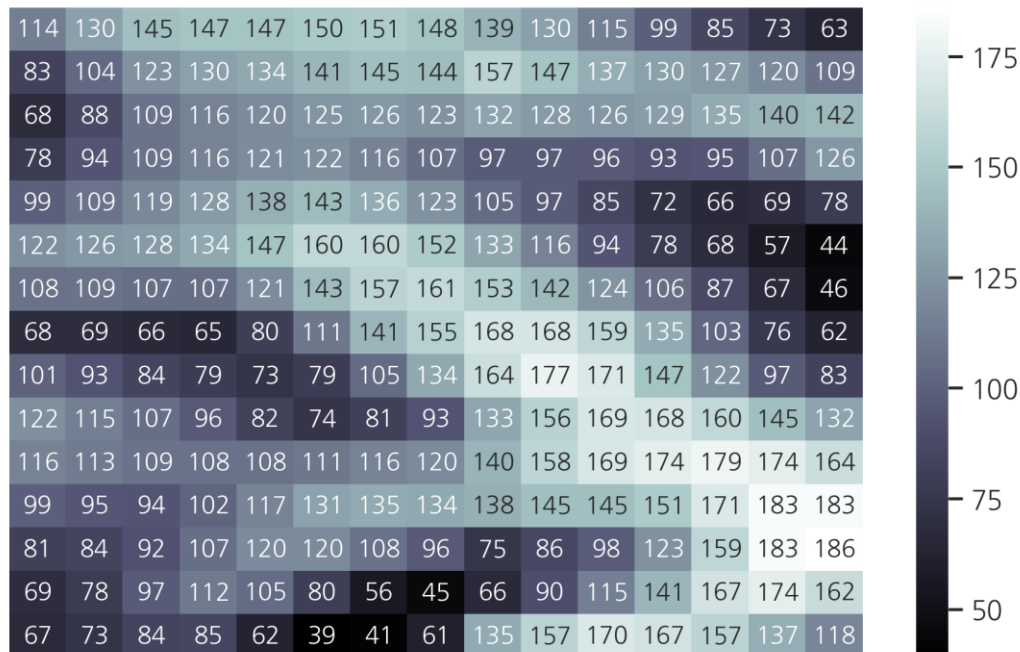
- 디지털화된 이미지는 픽셀(pixel, 화소)이라는 단위가 여러 개 모여서 그림을 표현한다.
- 한 개의 픽셀을 두 가지 값으로만 표현한 이미지를 바이너리 (binary, 이진) 이미지라고 한다.
- 영상 작업에서는 피사체의 색상과 명암 정보는 필요 없고 오직 피사체의 모양 정보만 필요할 때 이런 이미지를 사용한다.



Section 02 컬러 스페이스

□ 디지털 영상의 종류

- 그레이 스케일 이미지는 한 개의 픽셀을 0~255 의 값으로 표현한다.
- 픽셀 값의 크기로 명암을 표현하는데, 가장 작은 값인 0은 가장 어두운 검은색을 의미하고 값이 점점 커질수록 밝은 색을 의미하다가 255까지 가면 가장 밝은 흰색을 나타낸다.



Section 02 컬러 스페이스

□ 디지털 영상의 종류

- 컬러 이미지에 색상을 표현하는 방법은 무척 다양하다.
- 색상을 표현하는 방법에 따라 다르기는 하지만 흔히 컬러 이미지는 한 픽셀당 0~255의 값 3개를 조합해서 표현한다.
- 각 바이트마다 어떤 색상 표현의 역할을 맡을지를 결정하는 시스템을 컬러 스페이스(color space, 색공간)라고 한다.
- 컬러 스페이스의 종류는 RGB, HSV, YUV(YCbCr), CMYK 등 여러 가지가 있다.

RGB 이미지



R 채널



G 채널



B 채널

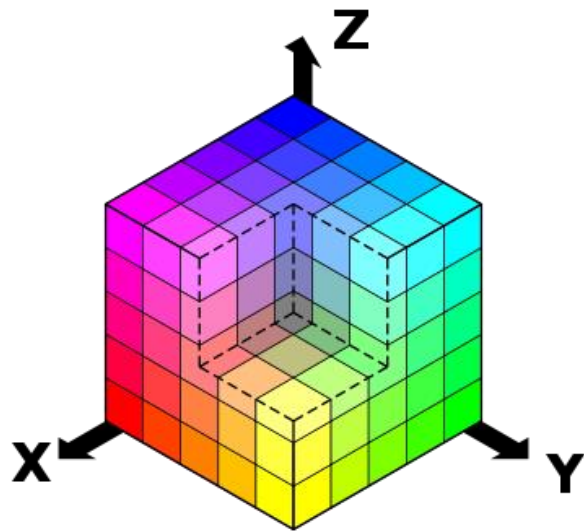


Section 02 컬러 스페이스

□ RGB, BGR, RGBA

- RGB는 빛의 원소인 빨강, 초록, 파랑 세 가지 색의 빛을 섞어서 원하는 색을 표현한다.
- 각 색상은 0~255 범위로 표현하고 값이 커질수록 해당 색상의 빛이 밝아지는 원리로 색상의 값이 모두 255 일 때 흰색으로 표현되고, 모든 색상 값이 0일 때 검은색이 표현된다.
- 세 가지 색상을 표현하므로 RGB 이미지는 3차원 배열로 표현된다.

row x column x channel



Section 02 컬러 스페이스

□ RGB, BGR, RGBA

- 영상의 크기에 해당하는 **행(row, height)**과 **열(column, width)**에 세 가지 색상을 표현하는 차원이 추가되는데, 이것을 **채널(channel)**이라고 한다.
- RGB는 3개의 채널로 색상을 표현하는 컬러 스페이스인데, OpenCV는 그 순서를 반대로 해서 BGR 순서를 사용한다.
- RGBA는 배경을 투명 처리하기 위해 알파(alpha) 채널을 추가한 것을 말한다.
- 4번째 채널의 값은 0~255로 표현할 수 있지만, 배경의 투명도를 표현하기 위해서는 0 과 255을 사용하는 경우가 많다.
- `cv2.imread()` 함수의 두 번째 인자가 `cv2.IMREAD_COLOR` 인 경우 BGR로 읽어 들이고 `cv2.IMREAD_UNCHANGED` 인 경우 대상 이미지가 알파 채널을 가지고 있다면 BGRA로 읽어 들인다.

Section 02 컬러 스페이스

□ RGB, BGR, RGBA

- 다음 예제는 배경이 투명한 OpenCV 로고 이미지를 두 가지 옵션을 지정해서 비교한다.

```
1 import cv2
2 import numpy as np
3
4 img = cv2.imread('./img/opencv_logo.png')
5 bgr = cv2.imread('./img/opencv_logo.png', cv2.IMREAD_COLOR)
6
7 bgra = cv2.imread('./img/opencv_logo.png', cv2.IMREAD_UNCHANGED)
8
9 print("default: ", img.shape, "color: ", bgr.shape, "unchanged: ", bgra.shape)
10
```

Section 02 컬러 스페이스

□ RGB, BGR, RGBA

- 다음 예제는 배경이 투명한 OpenCV 로고 이미지를 두 가지 옵션을 지정해서 비교한다.

```
11 cv2.imshow('bgr', bgr)
12 cv2.imshow('bgra', bgra)
13 cv2.imshow('alpha', bgra[:, :, 3])
14 cv2.waitKey(0)
15 cv2.destroyAllWindows()
```

Section 02 컬러 스페이스

□ 컬러 스페이스 변환

- 컬러 이미지를 그레이 스케일로 변환하는 것은 이미지 연산의 양을 줄여서 속도를 높이는 데 꼭 필요하다.
- 이때 애초에 그레이 스케일로 읽어오는 방법은 `cv2.imread(img, cv2.IMREAD_GRAYSCALE)`이다.
- 그런데 맨 처음에는 컬러 스케일로 읽어 들이고 필요에 따라 그레이 스케일이나 다른 컬러 스페이스로 변환해야 할 때도 많다.
- 그레이 스케일이나 다른 컬러 스페이스로 변환하는 방법은 변환 알고리즘을 직접 구현할 수도 있고 OpenCV에서 제공하는 `cv2.cvtColor()` 함수를 이용할 수도 있다.

Section 02 컬러 스페이스

□ 컬러 스페이스 변환

- 아래의 예제는 컬러 스케일을 그레이 스케일로 변환하는 작업을 각각 보여준다.

```
1  import cv2
2
3
4  img = cv2.imread('./img/model3.jpg')
5
6  img2 = img.astype(np.uint16)
7
8  b, g, r = cv2.split(img2)
9
10 gray1 = ((b + g + r)/3).astype(np.uint8)
11
12 gray2 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Section 02 컬러 스페이스

□ 컬러 스페이스 변환

- 아래의 예제는 컬러 스케일을 그레이 스케일로 변환하는 작업을 각각 보여준다.

```
10 cv2.imshow('original', img)
11 cv2.imshow('gray1', gray1)
12 cv2.imshow('gray2', gray2)
13
14 cv2.waitKey(0)
15 cv2.destroyAllWindows()
```

- 사실, 컬러 이미지를 그레이 스케일로 변환할 때 좀 더 정확한 명암을 얻으려면 단순히 평균 값만 계산하는 보다 좀 더 정교한 연산이 필요하다.
- 하지만, OpenCV에서 제공하는 `cv2.cvtColor(img, flag)` 함수는 이런 골치 아픈 알고리즘에서 우리를 자유롭게 해준다.

Section 02 컬러 스페이스

□ 컬러 스페이스 변환

- 다음은 cv2.cvtColor() 함수에 대한 설명이다.

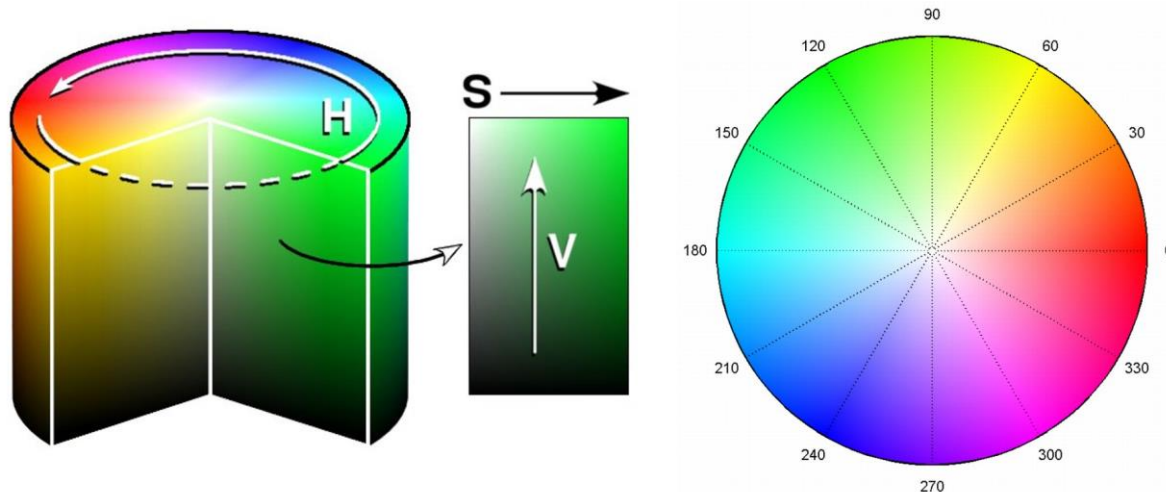
- `out = cv2.cvtColor(img, flag)`

- img : NumPy 배열, 변환할 이미지
- flag : 변환할 컬러 스페이스, cv2.COLOR_로 시작하는 이름
 - cv2.COLOR_BGR2GRAY : BGR 컬러 이미지를 그레이 스케일로 변환
 - cv2.COLOR_GRAY2BGR : 그레이 스케일 이미지를 BGR 컬러 이미지로 변환
 - cv2.COLOR_BGR2RGB : BGR 컬러 이미지를 RGB 컬러 이미지로 변환
 - cv2.COLOR_BGR2HSV : BGR 컬러 이미지를 HSV 컬러 이미지로 변환
 - cv2.COLOR_HSV2BGR : HSV 컬러 이미지를 BGR 컬러 이미지로 변환
 - cv2.COLOR_BGR2YUV : BGR 컬러 이미지를 YUV 컬러 이미지로 변환
 - cv2.COLOR_YUV2BGR : YUV 컬러 이미지를 BGR 컬러 이미지로 변환
- out : 변환한 결과 이미지(Numpy 배열)

Section 02 컬러 스페이스

□ 컬러 스페이스 변환

- HSV 포맷은 RGB와 마찬가지로 3채널로 컬러 이미지를 표시한다.
- 3채널은 각각 H(Hue, 색조), S(Saturation, 채도), V(Value, 명도)이다.
- 이때 명도를 표현하는 방법에 따라 마지막 V를 I(Intensity, 밀도)로 표기하는 HSI, 그리고 L(Lightness, 명도)로 표기하는 HSL 컬러 시스템도 있다.
- HSV를 설명하는 데 가장 흔히 사용하는 방법은 다음 그림과 같은 원통형 시스템이다.



Section 02 컬러 스페이스

□ 컬러 스페이스 변환

- H 값은 그 픽셀이 어떤 색인지를 표현한다.
- 이 그림을 요약해서 대략 색상의 범위에 맞는 H값을 표시하면 아래와 같다.
 - 빨강 : 165~180, 0~15
 - 초록 : 45~75
 - 파랑 : 90~120
- S값은 채도, 포화도, 또는 순도로 해석할 수 있는데, 해당 색상이 얼마나 순수하게 포함되어 있는지를 표현한다.
- S 값은 0~255 범위로 표현하며 255는 가장 순수한 색상을 표현한다.
- V 값은 명도로서 빛이 얼마나 밝은지 어두운지를 표현하는 값이다.
- 이 값도 범위가 0~255 이며 255 인 경우가 가장 밝은 상태이고 0(영, zero) 인 경우가 가장 어두운 상태로 검은색이 표시된다.

Section 02 컬러 스페이스

□ 컬러 스페이스 변환

- 아래의 예제는 완전한 빨강, 초록, 파랑 그리고 노랑을 BGR 포맷으로 표현해서 HSV로 변환하여 어떤 값인지를 알아보는 예제이다.

```
1  import cv2
2  import numpy as np
3
4  red_bgr = np.array([[[0, 0, 255]]], dtype=np.uint8)
5  green_bgr = np.array([[[0, 255, 0]]], dtype=np.uint8)
6  blue_bgr = np.array([[[255, 0, 0]]], dtype=np.uint8)
7  yellow_bgr = np.array([[[0, 255, 255]]], dtype=np.uint8)
8
9  red_hsv = cv2.cvtColor(red_bgr, cv2.COLOR_BGR2HSV)
10 green_hsv = cv2.cvtColor(green_bgr, cv2.COLOR_BGR2HSV)
11 blue_hsv = cv2.cvtColor(blue_bgr, cv2.COLOR_BGR2HSV)
12 yellow_hsv = cv2.cvtColor(yellow_bgr, cv2.COLOR_BGR2HSV)
13
```

Section 02 컬러 스페이스

□ 컬러 스페이스 변환

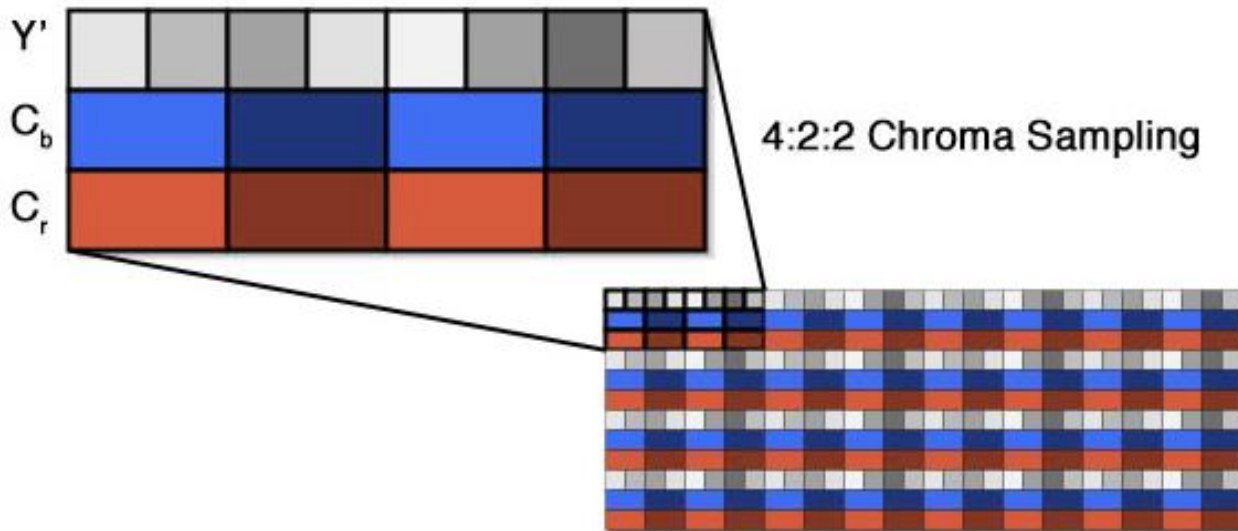
- 아래의 예제는 완전한 빨강, 초록, 파랑 그리고 노랑을 BGR 포맷으로 표현해서 HSV로 변환하여 어떤 값인지를 알아보는 예제이다.

```
14 print("red: ", red_hsv)
15 print("green: ", green_hsv)
16 print("blue: ", blue_hsv)
17 print("yellow: ", yellow_hsv)
```

Section 02 컬러 스페이스

□ YUV, YCbCr

- YUV 포맷은 사람이 색상을 인식할 때 밝기에 더 민감하고 색상은 대적으로 둔감한 점을 고려해서 만든 컬러 스페이스이다.
- Y는 밝기(Luma)를 표현하고 U(Chroma Blue, Cb)는 밝기와 파란색과의 색상 차, V(Chroma Red, Cr)는 밝기와 빨간색과의 색상 차를 표현한다.
- Y(밝기)는 많은 비트수를 할당하고 U(Cb)와 V(Cr)는 적은 비트 수를 할당해서 데이터를 압축하는 효과를 갖는다.



Section 02 컬러 스페이스

□ YUV, YCbCr

- YUV는 종종 YCbCr 포맷과 혼용되기도 하는데, 원래 YUV는 텔레비전 시스템에서 아날로그 컬러를 인코딩하는 데 사용하고, YCbCr 포맷은 MPEG이나 JPEG와 같은 디지털 컬러 정보를 인코딩하는 데 사용하였다.
- YUV는 요즘 들어 YCbCr로 인코딩된 파일을 설명하는 용도로 일반적으로 사용된다.
- YUV와 YCbCr 은 RGB 포맷에서 변환하기 위한 공식이 달라서 OpenCV는 cv2.COLOR_BGR2YUV, cv2.COLOR_BGR2YCrCb가 따로 있고 변환 결과도 미세하게 다르다.
- YUV는 밝기 정보와 컬러 정보를 분리해서 사용하므로 명암 대비(contrast)가 좋지 않은 영상을 좋게 만드는 데 대표적으로 활용된다.

Section 02 컬러 스페이스

□ YUV, YCbCr

- 예제는 완전히 어두운 값과 완전히 밝은 값 그리고 중간 값을 BGR로 표현한 후에 YUV로 변환한 3개 채널을 살펴본다.

```
1 import cv2
2 import numpy as np
3
4 dark = np.array([[[0, 0, 0]]], dtype=np.uint8)
5 middle = np.array([[[127, 127, 127]]], dtype=np.uint8)
6 bright = np.array([[[255, 0, 0]]], dtype=np.uint8)
7
8 dark_yuv = cv2.cvtColor(dark, cv2.COLOR_BGR2HSV)
9 middle_yuv = cv2.cvtColor(middle, cv2.COLOR_BGR2HSV)
10 bright_yuv = cv2.cvtColor(bright, cv2.COLOR_BGR2HSV)
11
12 print("dark: ", dark_yuv)
13 print("middle: ", middle_yuv)
14 print("bright: ", bright_yuv)
```

Section 03 이미지 임계 처리

□ 전역 임계 처리

- 임계 처리(thresholding)란 여러 점수를 커트라인을 기준으로 합격과 불합격으로 나누는 것처럼 여러 값을 경계점 기준으로 두 가지 부류로 나누는 것으로, 바이너리 이미지를 만드는 가장 대표적인 방법이다.
- 바이너리 이미지를 만들기 위해서는 컬러 이미지를 그레이 스케일로 바꾸고 각 픽셀의 값이 경계 값을 넘으면 255, 넘지 못하면 0을 지정한다.
- 이런 작업은 간단한 NumPy 연산만으로도 충분히 할 수 있지만 OpenCV는 `cv2.threshold()` 함수로 더 많은 기능을 제공한다.

Section 03 이미지 임계 처리

□ 전역 임계 처리

- 예제는 NumPy 연산과 OpenCV 함수로 각각 바이너리 이미지를 만드는 과정을 보여준다.

```
1  import cv2
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  img = cv2.imread('./img/gray_gradient.jpg', cv2.IMREAD_GRAYSCALE)
6
7  thresh_np = np.zeros_like(img)
8  thresh_np[img > 127] = 255
9
10 ret, thresh_cv = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
11 print(ret)
12
```


Section 03 이미지 임계 처리

□ 전역 임계 처리

- 예제는 NumPy 연산과 OpenCV 함수로 각각 바이너리 이미지를 만드는 과정을 보여준다.

```
13  imgs = {'Original': img, 'Numpy API': thresh_np, 'cv2.threshold': thresh_cv}
14  for i, (key, value) in enumerate(imgs.items()):
15      plt.subplot(1, 3, i + 1)
16      plt.title(key)
17      plt.imshow(value, cmap='gray')
18      plt.xticks([]); plt.yticks([])
19
20  plt.show()
```

Section 03 이미지 임계 처리

□ 전역 임계 처리

- `ret, out = cv2.threshold(img, threshold, value, type_flag)`
 - `img` : NumPy 배열, 변환할 이미지
 - `threshold` : **경계 값**
 - `value` : **경계 값 기준에 만족하는 픽셀에 적용할 값**
 - `type_flag` : **임계 처리 적용 방법 지정**
 - `cv2.THRESH_BINARY` : $px > threshold ? value : 0$, **픽셀이 경계 값을 넘으면 value를 지정하고, 넘지 못하면 0을 지정**
 - `cv2.THRESH_BINARY_INV` : $px > threshold ? 0 : value$, **`cv2.THRESH_BINARY`의 반대**
 - `cv2.THRESH_TRUNC` : $px > threshold ? value : px$, **픽셀 값이 경계 값을 넘으면 value를 지정하고, 넘지 못하면 원래의 값 유지**
 - `cv2.THRESH_TOZERO` : $px > threshold ? px : 0$, **픽셀 값이 경계 값을 넘으면 원래 값을 유지, 넘지 못하면 0을 지정**
 - `cv2.THRESH_TOZERO_INV` : $px > threshold ? 0 : px$, **`cv2.THRESH_TOZERO`의 반대**
 - `ret` : **임계 처리에 사용한 경계 값**
 - `out` : **결과 바이너리 이미지**

Section 03 이미지 임계 처리

□ 전역 임계 처리

- 아래 예제에서는 위에 나열한 몇 가지 다른 플래그를 이용한 임계 처리 사례로 보여준다.

```
1  import cv2
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  img = cv2.imread('./img/gray_gradient.jpg', cv2.IMREAD_GRAYSCALE)
6
7  _, t_bin = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
8  _, t_bininv = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV)
9  _, t_truc = cv2.threshold(img, 127, 255, cv2.THRESH_TRUNC)
10 _, t_2zr = cv2.threshold(img, 127, 255, cv2.THRESH_TOZERO)
11 _, t_2zrinv = cv2.threshold(img, 127, 255, cv2.THRESH_TOZERO_INV)
12
```

Section 03 이미지 임계 처리

□ 전역 임계 처리

- 아래 예제에서는 위에 나열한 몇 가지 다른 플래그를 이용한 임계 처리 사례로 보여준다.

```
13  imgs = {'Original': img, 'BINARY': t_bin, 'BINARY_INV': t_bininv, 'TRUNC': t_truc, 'TO  
    ZERO': t_2zr, 'TOZERO_INV': t_2zrinv}  
  
14  for i, (key, value) in enumerate(imgs.items()):  
  
15      plt.subplot(2, 3, i + 1)  
  
16      plt.title(key)  
  
17      plt.imshow(value, cmap='gray')  
  
18      plt.xticks([]); plt.yticks([])  
  
19  
  
20  plt.show()
```

Section 03 이미지 임계 처리

□ 오츠의 알고리즘

- 바이너리 이미지를 만들 때 가장 중요한 작업은 경계 값을 얼마로 정하느냐이다.
- 종이에 출력한 문서를 바이너리 이미지로 만드는 것을 예를 들면, 새하얀 종이에 검은색으로 출력된 문서의 영상이라면 굳이 임계 처리를 적용할 필요가 없다.
- 하지만, 현실은 흰색, 누런색, 회색 종이에 검은색, 파란색 등으로 인쇄된 문서가 더 많기 때문에 적절한 경계 값을 정하기 위해서는 여러 차례에 걸쳐 경계 값을 조금씩 수정해 가면서 가장 좋은 경계 값을 찾아야 한다.
- 1979년 오츠 노부유키(Nobuyuki Otsu)는 반복적인 시도 없이 한 번에 효율적으로 경계 값을 찾을 수 있는 방법을 제안했는데, 그의 이름을 따서 그것을 오츠의 이진화 알고리즘(Otsu's binarization method) 이라고 한다.

Section 03 이미지 임계 처리

□ �츠의 알고리즘

- �츠의 알고리즘은 경계 값을 임의로 정해서 픽셀들을 두 부류로 나누고 두 부류의 명암 분포를 반복해서 구한 다음 두 부류의 명암 분포를 가장 균일하게 하는 경계 값을 선택한다.
- 이것을 수식으로 표현하면 다음과 같다.
- OpenCV는 이미 구현한 �츠의 알고리즘을 사용할 수 있게 제공해 주는데, 이것을 사용하려면 앞서 설명한 `cv2.threshold()` 함수의 마지막 인자에 `cv2.THRESH_OTSU`를 추가해서 전달하기만 하면 된다.
- 그러면 원래 경계 값을 전달해야 하는 두 번째 인자 `threshold`는 무시되므로 아무 숫자나 전달해도 되고, 실행 후 결과 값으로 �츠의 알고리즘에 의해 선택된 경계 값은 반환 값 첫 번째 항목 `ret`로 받을 수 있다.

Section 03 이미지 임계 처리

□ �츠의 알고리즘

- 다음 코드는 `cv2.threshold ()` 함수에 �츠의 알고리즘을 적용하는 코드이다.

```
ret, t_img = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
```

```
1  import cv2
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  img = cv2.imread('../img/scanned_paper.jpg', cv2.IMREAD_GRAYSCALE)
6
7  _, t_130 = cv2.threshold(img, 130, 255, cv2.THRESH_BINARY)
8  t, t_otsu = cv2.threshold(img, -1, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
9  print('otsu threshold: %. t)
10
```

Section 03 이미지 임계 처리

□ �츠의 알고리즘

- 다음 코드는 `cv2.threshold()` 함수에 �츠의 알고리즘을 적용하는 코드이다.

```
ret, t_img = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
```

```
11  img = {'Original': img, 't:130': t_130, 'otsu : %d'%t : t_otsu}
12  for i, (key, value) in enumerate(imgs.items()):
13      plt.subplot(1, 3, i + 1)
14      plt.title(key)
15      plt.imshow(value, cmap='gray')
16      plt.xticks([]); plt.yticks([])
17
18  plt.show()
```


Section 03 이미지 임계 처리

□ 적응형 임계 처리

- 원본 영상에 조명이 일정하지 않거나 색이 여러 가지인 경우에는 아무리 여러 번 경계 값을 바꿔가며 시도해도 하나의 경계 값을 이미지 전체에 적용해서는 좋은 결과를 얻지 못한다.
- 이때 이미지를 여러 영역으로 나눈 다음 그 주변 픽셀 값만 가지고 계산을 해서 경계 값을 구해야 하는데, 이것을 적응형 임계 처리(adaptive threshold)라고 한다.

```
cv2.adaptiveThreshold(img, value, method, type_flag, block_size, C)
```

- img : 입력 영상
- value : 경계 값을 만족하는 픽셀에 적용할 값
- method : 경계 값 결정 방법
 - cv2.ADAPTIVE_THRESH_MEAN_C : 이웃 픽셀의 평균으로 결정
 - cv2.ADAPTIVE_THRESH_GAUSSIAN_C : 가우시안 분포에 따른 가중치의 합으로 결정
- type_flag : 임계 처리 적용 방법 지정(cv2.threshold() 함수와 동일)
- block_size : 영역으로 나눌 이웃의 크기($n \times n$), 홀수(3, 5, 7, ...)
- c : 계산된 경계 값 결과에서 가감할 상수(음수 가능)

Section 03 이미지 임계 처리

□ 적응형 임계 처리

- 다음 예제는 이미지에 적응형 임계 처리를 수행한 코드이다.

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 blk_size = 9
6 C = 5
7 img = cv2.imread('./img/sudoku.png', cv2.IMREAD_GRAYSCALE)
8
9 ret, th1 = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
10
11 th2 = cv2.threshold(img, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, blk_size, C)
12 th3 = cv2.threshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, blk_size, C)
```

Section 03 이미지 임계 처리

□ 적응형 임계 처리

- 다음 예제는 이미지에 적응형 임계 처리를 수행한 코드이다.

```
13 img = {'Original' : img, 'Global-Otsu: %d' %ret : th1, 'Adaptive-Mean' : th2, 'Adaptive-Gaussian' : th3}
14 for i, (key, value) in enumerate(imgs.items()):
15     plt.subplot(2, 2, i + 1)
16     plt.title(key)
17     plt.imshow(value, 'gray')
18     plt.xticks([]); plt.yticks([])
19
20 plt.show()
```

- 경계 값을 전체 이미지에 적용하는 것을 전역적(global) 적용이라고 하는 반면에, 이미지를 여러 구역으로 나누어 그 구역에 맞는 경계 값을 찾는 것을 지역적(local) 적용이라고 한다.
- 대부분의 이미지는 조명 차이와 그림자 때문에 지역적 적용이 필요하다.

Section 04 영상 연산

□ 영상과 영상의 연산

- 영상에 연산을 할 수 있는 방법은 NumPy의 브로드캐스팅 연산을 직접 적용하는 방법과 OpenCV에서 제공하는 네 가지 함수를 사용하는 방법이 있다.
- OpenCV에서 굳이 연산에 사용할 함수를 제공하는 이유는 영상에서의 한 픽셀이 가질 수 있는 값의 범위는 0~255 인데, 더하기나 빼기 연산을 한 결과가 255보다 클 수도 있고 0보다 작을 수도 있어서 결과 값을 0과 255로 제한할 안전 장치가 필요하기 때문이다.

- `dest=cv2.add(src1, src2 [, dest, mask , dtype))` : src1과 src2 더하기

- src1 : 입력 영상 1 또는 수
- src2 : 입력 영상 2 또는 수
- dest : 출력 영상
- mask : 0이 아닌 픽셀만 연산
- dtype : 출력 dtype

Section 04 영상 연산

□ 영상과 영상의 연산

- `dest=cv2.subtract(src1, src2 [, dest, mask, dtype])` : src1 에서 src2를 빼기
 - 모든 인자는 `cv2.add()` 함수와 동일
- `dest=cv2.multiply(src1, src2 [, dest, scale, dtype])` : src1과 src2를 곱하기
 - scale: 연산 결과에 추가 연산할 값
- `dest=cv2.divide(src1, src2[, dest, scale, dtype])` : src1을 src2로 나누기
 - 모든 인자는 `cv2.multiply()` 와 동일

Section 04 영상 연산

□ 알파 블렌딩

- 두 영상을 합성하려고 할 때 앞서 살펴본 더하기(+) 연산이나 `cv2.add()` 함수만으로는 좋은 결과를 얻을 수 없는 경우가 많다.
- 직접 더하기 연산을 하면 255를 넘는 경우 초과 값을 가지므로 영상이 거뭇거뭇하게 나타나고 `cv2.add()` 연산을 하면 대부분의 픽셀 값이 255 가까이 몰리는 현상이 일어나서 영상이 하얗게 날아간 것처럼 보인다.
- 아래의 예제와 그 결과인 그림은 이런 현상을 보여주고 있다.

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 img1 = cv2.imread('../img/wing_wall.jpg')
6 img2 = cv2.imread('../img/yate.jpg')
7
8 img3 = img1 + img2
9 img4 = cv2.add(img1, img2)
10
```

Section 04 영상 연산

□ 알파 블렌딩

- 두 영상을 합성하려고 할 때 앞서 살펴본 더하기(+) 연산이나 `cv2.add()` 함수만으로는 좋은 결과를 얻을 수 없는 경우가 많다.
- 직접 더하기 연산을 하면 255를 넘는 경우 초과 값을 가지므로 영상이 거뭇거뭇하게 나타나고 `cv2.add()` 연산을 하면 대부분의 픽셀 값이 255 가까이 몰리는 현상이 일어나서 영상이 하얗게 날아간 것처럼 보인다.
- 아래의 예제와 그 결과인 그림은 이런 현상을 보여주고 있다.

```
11 imgs = {'img1': img1, 'img2': img2, 'img1 + img2': img3, 'cv2.add(img1, img2)': img4}
12
13 for i, (k, v) in enumerate(imgs.items()):
14     plt.subplot(2, 2, i+1)
15     plt.imshow(v[:, :, :-1])
16     plt.title(k)
17     plt.xticks([]); plt.yticks([])
18 plt.show()
```

Section 04 영상 연산

□ 알파 블렌딩

- 두 영상을 합성하려면 각 픽셀의 합이 255가 되지 않게 각각의 영상에 가중치를 줘서 계산해야 한다.
- 예를 들어 두 영상이 정확히 절반씩 반영된 결과 영상을 원한다면 각 영상의 픽셀 값에 각각 50%씩 곱해서 새로운 영상을 생성하면 된다.
- 이것을 수식으로 나타내면 다음과 같고 이 때 각 영상에 적용할 가중치를 알파(alpha) 값이라고 부른다.
- 알파 값을 조정해서 7:3, 6:4, 5:5 등과 같이 배분하는 방식이다.

$$g(x) = (1 - \alpha)f_0(x) + \alpha f_1(x)$$

- $f_0(x)$: 첫 번째 이미지 픽셀 값
- $f_1(x)$: 두 번째 이미지 픽셀 값
- α : 가중치(알파)
- $g(x)$: 합성 결과 픽셀 값

Section 04 영상 연산

□ 알파 블렌딩

- `cv2.addWeight(img1, alpha, img2, beta, gamma)`
 - `img1, img2` : 합성할 두 영상
 - `alpha` : `img1`에 지정할 가중치(알파 값)
 - `beta` : `img2`에 지정할 가중치, 흔히 $(1 - \alpha)$ 적용
 - `gamma` : 연산 결과에 가감할 상수, 흔히 0(zero) 적용
- 아래 예제는 각 영상에서 50%씩의 가중치로 앞서 실습한 영상을 다시 합성하고 있다.

```
1 import cv2
2 import numpy as np
3
4 alpha = 0.5
5
6 img1 = cv2.imread('../img/wing_wall.jpg')
7 img2 = cv2.imread('../img/yate.jpg')
8
```

Section 04 영상 연산

□ 알파 블렌딩

- 아래 예제는 각 영상에서 50%씩의 가중치로 앞서 실습한 영상을 다시 합성하고 있다.

```
9   blended = img1 * alpha + img2 * (1 - alpha)
10  blended = blended.astype(np.uint8)
11  cv2.imshow('img1 * alpha + img2 * (1 - alpha)', blended)
12
13  dst = cv2.addWeighted(img1, alpha, img2, (1 - alpha), 0)
14  cv2.imshow('cv2.addWeighted', dst)
15
16  cv2.waitKey(0)
17  cv2.destroyAllWindows()
```

Section 04 영상 연산

□ 알파 블렌딩

- 아래의 예제는 남자의 얼굴과 사자의 얼굴을 알파 블렌딩하는 데 트랙바로 알파 값을 조정할 수 있게 했다.

```
1 import cv2
2 import numpy as np
3
4 win_name = 'Alpha blending'
5 trackbar_name = 'fade'
6
7 def onChange(x):
8     alpha = x/100
11     dst = cv2.addWeighted(img1, 1 - alpha, img2, alpha, 0)
12     cv2.imshow(win_name, dst)
13
14 img1 = cv2.imread('../img/man_face.jpg')
15 img2 = cv2.imread('../img/lion_face.jpg')
```

Section 04 영상 연산

□ 알파 블렌딩

- 아래의 예제는 남자의 얼굴과 사자의 얼굴을 알파 블렌딩하는 데 트랙바로 알파 값을 조정할 수 있게 했다.

```
16
17  cv2.imshow(win_name, img1)
18
19  dst = cv2.addWeighted(img1, alpha, img2, (1 - alpha), 0)
20  cv2.imshow('cv2.addWeighted', dst)
21
22  cv2.waitKey(0)
23  cv2.destroyAllWindows()
```

- 알파 블렌딩은 흔히 페이드-인/아웃(fade-in/out) 기법으로 영상이 전환되는 장면에서 자주 사용되며, '구미호'나 '늑대인간' 같은 영화의 변신 장면에서 얼굴 모핑(face morphing)이라는 기법으로 효과를 내는데, 이 기법을 구성하는 한 가지 기술이기도 하다.

Section 04 영상 연산

□ 비트와이즈 연산

- OpenCV는 두 영상의 각 픽셀에 대한 비트와이즈 (bitwise, 비트 단위) 연산 기능을 제공한다.
- 비트와이즈 연산은 두 영상을 합성할 때 특정 영역만 선택하거나 특정 영역만 제외하는 등의 선별적인 연산에 도움이 된다.
- `bitwise_and(img1, img2, mask = None)` : 각 픽셀에 대해 비트와이즈 AND 연산
- `bitwise_or(img1, img2, mask = None)` : 각 픽셀에 대해 비트와이즈 OR 연산
- `bitwise_xor(img1, img2, mask = None)` : 각 픽셀에 대해 비트와이즈 XOR 연산
- `bitwise_not(img1, mask = None)` : 각 픽셀에 대해 비트와이즈 NOT 연산
 - `img1, img2` : 연산 대상 영상, 동일한 shape
 - `mask` : 0이 아닌 픽셀만 연산, 바이너리 이미지

Section 04 영상 연산

□ 비트와이즈 연산

- 다음 예제는 하나는 좌우로, 나머지 하나는 위아래로 0과 255로 나누어 200 x 400 크기의 영상을 생성하고 비트와이즈 연산을 수행한 코드이다.

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 img = cv2.imread('./img/model3.jpg')
6
7 mask = np.zeros_like(img)
8 cv2.circle(mask, (180, 160), 100, (255, 255, 255), -1)
9
```

Section 04 영상 연산

□ 비트와이즈 연산

- 다음 예제는 하나는 좌우로, 나머지 하나는 위아래로 0과 255로 나누어 200 x 400 크기의 영상을 생성하고 비트와이즈 연산을 수행한 코드이다.

```
10 masked = cv2.bitwise_and(img, mask)
11
12 cv2.imshow('original', img)
13 cv2.imshow('mask', mask)
14 cv2.imshow('masked', masked)
15 cv2.waitKey(0)
16 cv2.destroyAllWindows()
```

Section 04 영상 연산

□ 비트와이즈 연산

- 다음 예제는 앞 예제에서 비트와이즈 연산 함수의 세 번째 인자인 mask를 이용하여 2차원 배열만으로 지정한 영역만 떼어낸 코드이다.

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 img = cv2.imread('./img/model3.jpg')
6
7 mask = np.zeros(img.shape[:2], dtype=np.uint8)
8 cv2.circle(mask, (150, 140), 100, (255), -1)
9
```


Section 04 영상 연산

□ 비트와이즈 연산

- 다음 예제는 앞 예제에서 비트와이즈 연산 함수의 세 번째 인자인 mask를 이용하여 2차원 배열만으로 지정한 영역만 떼어낸 코드이다.

```
10 masked = cv2.bitwise_and(img, img, mask=mask)
11
12 cv2.imshow('original', img)
13 cv2.imshow('mask', mask)
14 cv2.imshow('masked', masked)
15 cv2.waitKey(0)
16 cv2.destroyAllWindows()
```

Section 04 영상 연산

□ 차영상

- 영상에서 영상을 빼기 연산하면 두 영상의 차이, 즉 변화를 알 수 있는데, 이것을 차영상(image differencing) 이라고 한다.
- 산업현장에서 도면의 차이를 찾거나 전자제품의 PCB(Printable Circuit Board) 회로의 오류를 찾는 데도 사용할 수 있고, 카메라로 촬영한 영상에 실시간으로 움직임이 있는지를 알아내는 데도 유용하다.
- 차영상을 구할 때 두 영상을 무턱대고 빼기 연산하면 음수가 나올 수 있으므로 절대 값을 구해야 한다.
- 아래는 OpenCV에서 제공하는 절대 값의 차를 구하는 함수이다.
- ```
diff = cv2.absdiff(img1, img2)
```

  - img1, img2 : 입력 영상
  - diff : 두 영상의 차의 절대 값 반환

## Section 04 영상 연산

### □ 차영상

- 예제는 사람의 눈으로 찾기 힘든 두 도면의 차이를 찾아 표시한다.

```
1 import cv2
2 import numpy as np
3
4 img1 = cv2.imread('./img/robot_arm1.jpg')
5 img2 = cv2.imread('./img/robot_arm2.jpg')
6 img1_gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
7 img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
8
9 diff = cv2.absdiff(img1_gray, img2_gray)
10
11 _, diff = cv2.threshold(diff, 1, 255, cv2.THRESH_BINARY)
12 diff_red = cv2.cvtColor(diff, cv2.COLOR_GRAY2BGR)
13 diff_red[:, :, 2] = 0
14
```

## Section 04 영상 연산

### □ 차영상

- 예제는 사람의 눈으로 찾기 힘든 두 도면의 차이를 찾아 표시한다.

```
15 spot = cv2.bitwise_xor(img2, diff_red)
16
17 cv2.imshow('img1', img1)
18 cv2.imshow('img2', img2)
19 cv2.imshow('diff', diff)
20 cv2.imshow('spot', spot)
21 cv2.waitKey(0)
22 cv2.destroyAllWindows()
```

## Section 04 영상 연산

---

### □ 이미지 합성과 마스킹

- 두 개 이상의 영상에서 특정 영역끼리 합성하기 위해서는 전경이 될 영상과 배경이 될 영상에서 합성하고자 하는 영역만 떼어내는 작업과 그것을 합하는 작업으로 나눌 수 있다.
- 여기서 원하는 영역만을 떼어내는 데 꼭 필요한 것이 마스크(mask)이다.
- 배경이 투명한 알파 채널 영상을 이용해서 영상을 합성할 수 있다.
- 배경이 투명한 영상은 4개 채널 중 마지막 채널은 배경에 해당하는 영역은 0값을, 전경에 해당하는 영역은 255 값을 갖는다.
- 이것을 이용하면 손쉽게 마스크를 만들 수 있다. 마스크를 이용해서 전경과 배경을 오려내는 것은 앞서 살펴본 `cv2.bitwise_and()` 연산을 이용하면 쉽다.

## Section 04 영상 연산

### □ 이미지 합성과 마스크

- 다음 예제는 배경이 투명한 OpenCV 로고 이미지를 소녀의 사진과 합성하는 코드이다.

```
1 import cv2
2 import numpy as np
3
4 img_fg = cv2.imread('../img/opencv_logo.png', cv2.IMREAD_UNCHANGED)
5 img_bg = cv2.imread('../img/girl.jpg')
6
7 _, mask = cv2.threshold(img_fg[:, :, 3], 1, 255, cv2.THRESH_BINARY)
8 mask_inv = cv2.bitwise_not(mask)
9
10 img_fg = cv2.cvtColor(img_fg, cv2.COLOR_BGRA2BGR)
11 h, w = img_fg.shape[:2]
12 roi = img_bg[10:10+h, 10:10+w]
13
14 masked_fg = cv2.bitwise_and(img_fg, img_fg, mask=mask)
15 masked_bg = cv2.bitwise_and(roi, roi, mask=mask_inv)
16
```

## Section 04 영상 연산

### □ 이미지 합성과 마스킹

- 다음 예제는 배경이 투명한 OpenCV 로고 이미지를 소녀의 사진과 합성하는 코드이다.

```
17 added = masked_fg + masked_bg
18 img_bg[10:10+h, 10:10+w] = added
19
20 cv2.imshow('mask', mask)
21 cv2.imshow('mask_inv', mask_inv)
22 cv2.imshow('masked_fg', masked_fg)
23 cv2.imshow('masked_bg', masked_bg)
24 cv2.imshow('added', added)
25 cv2.imshow('result', img_bg)
26 cv2.waitKey()
27 cv2.destroyAllWindows()
```

## Section 04 영상 연산

### □ 이미지 합성과 마스킹

- 모양에 따라 영역을 떼어내려 는 경우도 있지만, 색상에 따라 영역을 떼어 내야 하는 경우도 있다.
- 이때는 색을 가지고 마스크를 만들어야 하는데, HSV로 변환하면 원하는 색상 범위의 것만 골라낼 수 있다.
- OpenCV는 특정 범위에 속하는지를 판단할 수 있는 함수를 아래와 같이 제공한다.
- 이것을 이용하면 특정 범위 값을 만족하는 마스크를 만들기 쉽다.
- `dst = cv2.inRange(img, from, to)` : 범위에 속하지 않은 픽셀 판단
  - `img` : 입력 영상
  - `from` : 범위의 시작 배열
  - `to` : 범위의 끝 배열
  - `dst` : `img`가 `from ~ to` 에 포함되면 255, 아니면 0을 픽셀 값으로 하는 배열



# Section 04 영상 연산

## □ 이미지 합성과 마스킹

- 예제는 컬러 큐브에서 색상별로 추출하는 예제이다.

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 img = cv2.imread("../img/cube.jpg")
6 hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
7
8 blue1 = np.array([90, 50, 50])
9 blue2 = np.array([120, 255, 255])
10 green1 = np.array([45, 50, 50])
11 green2 = np.array([75, 255, 255])
12 red1 = np.array([0, 50, 50])
13 red2 = np.array([15, 255, 255])
14 red3 = np.array([165, 50, 50])
15 red4 = np.array([180, 255, 255])
```

# Section 04 영상 연산

## □ 이미지 합성과 마스킹

- 예제는 컬러 큐브에서 색상별로 추출하는 예제이다.

```
16 yellow1 = np.array([20, 50,50])
17 yellow2 = np.array([35, 255,255])
18
19 mask_blue = cv2.inRange(hsv, blue1, blue2)
20 mask_green = cv2.inRange(hsv, green1, green2)
21 mask_red = cv2.inRange(hsv, red1, red2)
22 mask_red2 = cv2.inRange(hsv, red3, red4)
23 mask_yellow = cv2.inRange(hsv, yellow1, yellow2)
24
25 res_blue = cv2.bitwise_and(img, img, mask=mask_blue)
26 res_green = cv2.bitwise_and(img, img, mask=mask_green)
27 res_red1 = cv2.bitwise_and(img, img, mask=mask_red)
28 res_red2 = cv2.bitwise_and(img, img, mask=mask_red2)
29 res_red = cv2.bitwise_or(res_red1, res_red2)
30 res_yellow = cv2.bitwise_and(img, img, mask=mask_yellow)
```

## Section 04 영상 연산

### □ 이미지 합성과 마스킹

- 예제는 컬러 큐브에서 색상별로 추출하는 예제이다.

```
31
32 imgs = {'original': img, 'blue':res_blue, 'green':res_green, 'red':res_red, 'yellow':res_yellow}
33 for i, (k, v) in enumerate(imgs.items()):
34 plt.subplot(2,3, i+1)
35 plt.title(k)
36 plt.imshow(v[:, :, ::-1])
37 plt.xticks([]); plt.yticks([])
38 plt.show()
```

## Section 04 영상 연산

### □ 이미지 합성과 마스크

- 예제는 컬러 큐브에서 색상별로 추출하는 예제이다.

```
31
32 imgs = {'original': img, 'blue':res_blue, 'green':res_green, 'red':res_red, 'yellow':res_yellow}
33 for i, (k, v) in enumerate(imgs.items()):
34 plt.subplot(2,3, i+1)
35 plt.title(k)
36 plt.imshow(v[:, :, ::-1])
37 plt.xticks([]); plt.yticks([])
38 plt.show()
```

- 이와 같이 색상을 이용한 마스크를 이용하는 것이 크로마키(chromakey)의 원리이다.
- 일기 예보나 영화를 촬영할 때 초록색 또는 파란색 배경을 두고 찍어서 나중에 원하는 배경과 합성하는 것을 크로마키잉(chroma keying) 이라고 하고 그 초록색 배경을 크로마키라고 하다.

## Section 04 영상 연산

### □ 이미지 합성과 마스크

- 다음 예제는 크로마키를 배경으로 한 영상에서 크로마키 색상으로 마스크를 만들어 합성하는 예제이다.

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 img1 = cv2.imread('./img/man_chromakey.jpg')
6 img2 = cv2.imread('./img/street.jpg')
7
8 height1, width1 = img1.shape[:2]
9 height2, width2 = img2.shape[:2]
10 x = (width2 - width1)//2
11 y = height2 - height1
12 w = x + width1
13 h = y + height1
14
```

## Section 04 영상 연산

### □ 이미지 합성과 마스킹

- 다음 예제는 크로마키를 배경으로 한 영상에서 크로마키 색상으로 마스크를 만들어 합성하는 예제이다.

```
15 chromakey = img1[:10, :10, :]
16 offset = 20
17
18 hsv_chroma = cv2.cvtColor(chromakey, cv2.COLOR_BGR2HSV)
19 hsv_img = cv2.cvtColor(img1, cv2.COLOR_BGR2HSV)
20
21 chroma_h = hsv_chroma[:, :, 0]
22 lower = np.array([chroma_h.min()-offset, 100, 100])
23 upper = np.array([chroma_h.max()+offset, 255, 255])
24
25 mask = cv2.inRange(hsv_img, lower, upper)
26 mask_inv = cv2.bitwise_not(mask)
27 roi = img2[y:h, x:w]
28 fg = cv2.bitwise_and(img1, img1, mask=mask_inv)
29 bg = cv2.bitwise_and(roi, roi, mask=mask)
30 img2[y:h, x:w] = fg + bg
```

## Section 04 영상 연산

### □ 이미지 합성과 마스킹

- 다음 예제는 크로마키를 배경으로 한 영상에서 크로마키 색상으로 마스크를 만들어 합성하는 예제이다.

```
31
32 cv2.imshow('chromakey', img1)
33 cv2.imshow('added', img2)
34 cv2.waitKey()
35 cv2.destroyAllWindows()
```

- 이렇게 영상 합성에는 대부분 알파 블렌딩 또는 마스킹이 필요하다.
- 하지만, 이런 작업은 블렌딩을 위한 적절한 알파 값 선택과 마스킹을 위한 모양의 좌표나 색상 값 선택에 많은 노력과 시간이 필요하다.

# Section 04 영상 연산

## □ 이미지 합성과 마스크

- `dst = cv2.seamlessClone(src, dst, mask, coords, flags[, output])`
  - src : 입력 영상, 일반적으로 전경
  - dst : 대상 영상, 일반적으로 배경
  - mask : 마스크, src에서 합성하고자 하는 영역은 255, 나머지는 0
  - coords : src가 놓여지기 원하는 dst의 좌표(중앙)
  - flags : 합성 방식
    - cv2.NORMAL\_CLONE : 입력 원본 유지
    - cv2.MIXED\_CLONE : 입력과 대상을 혼합
  - output : 합성 결과



# Section 04 영상 연산

## □ 이미지 합성과 마스크

- 다음 예제는 cv2.SeamlessClone() 함수로 사진을 합성해서 손에 꽃 문신을 한 것처럼 만든 코드이다.

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 img1 = cv2.imread("./img/drawing.jpg")
6 img2= cv2.imread("./img/my_hand.jpg")
7
8 mask = np.full_like(img1, 255)
9
10 height, width = img2.shape[:2]
11 center = (width//2, height//2)
12
13 normal = cv2.seamlessClone(img1, img2, mask, center, cv2.NORMAL_CLONE)
14 mixed = cv2.seamlessClone(img1, img2, mask, center, cv2.MIXED_CLONE)
```

## Section 04 영상 연산

### □ 이미지 합성과 마스킹

- 다음 예제는 `cv2.SeamlessClone()` 함수로 사진을 합성해서 손에 꽃 문신을 한 것처럼 만든 코드이다.

```
15
16 cv2.imshow('normal', normal)
17 cv2.imshow('mixed', mixed)
18 cv2.waitKey()
19 cv2.destroyAllWindows()
```

- 결과로 나온 그림을 보면 함수의 마지막 인자 플래그가 `cv2.NORMAL_CLONE`인 경우 꽃 그림이 선명하긴 하지만, 주변의 피부가 뭉개진 듯한 결과를 보인다.
- 반면에, `cv2.MIXED_CLONE`을 사용한 경우에는 감쪽같이 두 영상의 특징을 살려서 표현하고 있다.
- 이 함수는 이미지 합성에 꼭 필요한 알파 값이나 마스크에 대해 신경 쓰지 않아도 되서 무척 편리하다.

# Section 05 히스토그램

## □ 히스토그램 계산과 표시

- 영상 분야에서의 히스토그램은 전체 영상에서 픽셀 값이 1인 픽셀이 몇 개이고 2인 픽셀이 몇 개이고 하는 식으로 픽셀 값이 255인 픽셀이 몇 개인지까지 세는 것을 말한다.
- 그렇게 하는 이유는 전체 영상에서 픽셀들의 색상이나 명암의 분포를 파악하기 위해서이다.
- OpenCV는 영상에서 히스토그램을 계산하는 `cv2.calcHist( )` 함수를 제공한다.

```
cv2.calcHist(img, channel, mask, hist Size, ranges)
```

- `img` : 입력 영상, `[img]`처럼 리스트로 감싸서 표현
- `channel` : 처리할 채널, 리스트로 감싸서 표현
  - 1채널 : `[0]`, 2채널: `[0, 1]`, 3채널 : `[0, 1, 2]`
- `mask` : 마스크에 지정한 픽셀만 히스토그램 계산
- `histSize` : 계급(bin)의 개수, 채널 개수에 맞게 리스트로 표현
  - 1 채널 : `(256)`, 2채널: `[256, 256]`, 3채널 : `[256 , 256, 256]`
- `ranges` : 각 픽셀이 가질 수 있는 값의 범위, RGB인 경우 `[0, 256]`

# Section 05 히스토그램

## □ 히스토그램 계산과 표시

- 다음 예제는 그레이 스케일 이미지의 히스토그램을 계산해서 그리는 코드이다.

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 img = cv2.imread('./img/seoul.jpg', cv2.IMREAD_GRAYSCALE)
6 cv2.imshow('img', img)
7
8 hist = cv2.calcHist([img], [0], None, [256], [0,255])
9 plt.plot(hist)
10
11 print("hist.shape:", hist.shape) #--③ 히스토그램의 shape (256,1)
12 print("hist.sum():", hist.sum(), "img.shape:",img.shape)
13 plt.show()
```

# Section 05 히스토그램

## □ 히스토그램 계산과 표시

- 다음 예제는 컬러 스케일 이미지의 3개 채널에 대해서 1차원 히스토그램을 각각 구하고 나서 하나의 플롯에 그리는 예제이다.

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 img = cv2.imread('./img/seoul.jpg')
6 cv2.imshow('img', img)
7
8 channels = cv2.split(img)
9 colors = ('b', 'g', 'r')
10 for (ch, color) in zip (channels, colors):
11 hist = cv2.calcHist([ch], [0], None, [256], [0, 255])
12 plt.plot(hist, color = color)
13 plt.show()
```

## Section 05 히스토그램

---

### □ 노멀라이즈

- 노멀라이즈(normalize, 정규화)는 원래 기준이 서로 다른 값을 같은 기준이 되게 만드는 것을 말한다.
- 노멀라이즈는 서로 다른 기준을 하나의 절대적인 기준으로 만들기도 하지만 절대적인 기준 대신 특정 구간으로 노멀라이즈하면 특정 부분에 몰려 있는 값을 전체 영역으로 골고루 분포하게 할 수도 있다.
- 영상 분야에서는 노멀라이즈를 가지고 픽셀 값들이 0~255에 골고루 분포하지 않고 특정 영역에 몰려 있는 경우 화질을 개선하기도 하고 영상 간의 연산을 해야 하는데, 서로 조건이 다른 경우 같은 조건으로 만들기도 한다.

# Section 05 히스토그램

## □ 노멀라이즈

- OpenCV는 노멀라이즈 기능을 아래와 같은 함수로 제공한다.

- `dst = cv2.normalize(src, dst, alpha, beta, type_flag)`

- src : 노멀라이즈 이전 데이터
- dst : 노멀라이즈 이후 데이터
- alpha : 노멀라이즈 구간 1
- beta : 노멀라이즈 구간 2, 구간 노멀라이즈가 아닌 경우 사용 안함
- type\_flag : 알고리즘 선택 플래그 상수
  - cv2.NORM\_MINMAX : alpha와 beta 구간으로 노멀라이즈
  - cv2.NORM\_L1 : 전체 합으로 나누기, alpha = 노멀라이즈 전체 합
  - cv2.NORM\_L2 : 단위 벡터 (unit vector)로 노멀라이즈
  - cv2.NORM\_INF : 최대 값으로 나누기

# Section 05 히스토그램

## □ 노멀라이즈

- 아래의 예제는 부연 영상에 노멀라이즈를 적용해서 화질을 개선하는 예제이다.

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 img = cv2.imread('./img/abnormal.jpg', cv2.IMREAD_GRAYSCALE)
6
7 img_f = img.astype(np.float32)
8 img_norm = ((img_f - img_f.min()) * (255) / (img_f.max() - img_f.min()))
9 img_norm = img_norm.astype(np.uint8)
10
11 img_norm2 = cv2.normalize(img, None, 0, 255, cv2.NORM_MINMAX)
12
13 hist = cv2.calcHist([img], [0], None, [256], [0, 255])
14 hist_norm = cv2.calcHist([img_norm], [0], None, [256], [0, 255])
15 hist_norm2 = cv2.calcHist([img_norm2], [0], None, [256], [0, 255])
```



# Section 05 히스토그램

## □ 노멀라이즈

- 아래의 예제는 부연 영상에 노멀라이즈를 적용해서 화질을 개선하는 예제이다.

```
16
17 cv2.imshow('Before', img)
18 cv2.imshow('Manual', img_norm)
19 cv2.imshow('cv2.normalize()', img_norm2)
20
21 hists = {'Before' : hist, 'Manual':hist_norm, 'cv2.normalize()':hist_norm2}
22 for i, (k, v) in enumerate(hists.items()):
23 plt.subplot(1,3,i+1)
24 plt.title(k)
25 plt.plot(v)
26 plt.show()
```

## Section 05 히스토그램

### □ 이퀄라이즈

- 노멀라이즈는 분포가 한곳에 집중되어 있는 경우에는 효과적이지만 그 집중된 영역에서 멀리 떨어진 값이 있을 경우에는 효과가 없다.
- 이때에는 이퀄라이즈(equalize, 평탄화)가 필요하다.
- 이퀄라이즈는 히스토그램으로 빈도를 구해서 그것을 노멀라이즈한 후 누적값을 전체 개수로 나누어 나온 결과 값을 히스토그램 원래 픽셀 값에 매핑한다.
- 이퀄라이즈는 각각의 값이 전체 분포에 차지하는 비중에 따라 분포를 재분배하므로 명암 대비(contrast)를 개선하는 데 효과적이다.
- OpenCV에서 제공하는 이퀄라이즈 함수는 아래와 같다.
- ```
dst = cv2.equalizeHist(src [, dst])
```

 - src : 대상 이미지, 8비트 1 채널
 - dst : 결과 이미지

Section 05 히스토그램

□ 이퀄라이즈

- 예제는 어둡게 나온 사진을 그레이스케일로 바꾸어 이퀄라이즈를 적용해서 개선시키는 예제이다.

```
1  import cv2
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  img = cv2.imread('./img/yate.jpg', cv2.IMREAD_GRAYSCALE)
6  rows, cols = img.shape[:2]
7
8  hist = cv2.calcHist([img], [0], None, [256], [0, 256])
9  cdf = hist.cumsum()
10 cdf_m = np.ma.masked_equal(cdf, 0)
11 cdf_m = (cdf_m - cdf_m.min()) / (rows * cols) * 255
12 cdf = np.ma.filled(cdf_m, 0).astype('uint8')
13 print(cdf.shape)
14 img2 = cdf[img]
15
```

Section 05 히스토그램

□ 이퀄라이즈

- 예제는 어둡게 나온 사진을 그레이스케일로 바꾸어 이퀄라이즈를 적용해서 개선시키는 예제이다.

```
16 img3 = cv2.equalizeHist(img)
17
18 hist2 = cv2.calcHist([img2], [0], None, [256], [0, 256])
19 hist3 = cv2.calcHist([img3], [0], None, [256], [0, 256])
20
21 cv2.imshow('Before', img)
22 cv2.imshow('Manual', img2)
23 cv2.imshow('cv2.equalizeHist()', img3)
24 hists = {'Before':hist, 'Manual':hist2, 'cv2.equalizeHist()':hist3}
25 for i, (k, v) in enumerate(hists.items()):
26     plt.subplot(1,3,i+1)
27     plt.title(k)
28     plt.plot(v)
29 plt.show()
```

Section 05 히스토그램

□ 이퀄라이즈

- 히스토그램 이퀄라이즈는 컬러 스케일에도 적용할 수 있는데, 밝기 값을 개선하기 위해서는 3개 채널 모두를 개선해야 하는 BGR 컬러 스페이스보다는 YUV나 HSV로 변환해서 밝기 채널만을 연산해서 최종 이미지에 적용하는 것이 좋다.
- 다음 예제는 YUV 컬러 스페이스로 변경한 컬러 이미지에 대한 이퀄라이즈를 보여준다.

```
1  import cv2
2  import numpy as np
3
4  img = cv2.imread('./img/yate.jpg')
5  img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
6
7  img_yuv[:, :, 0] = cv2.equalizeHist(img_yuv[:, :, 0])
8
```

Section 05 히스토그램

□ 이퀄라이즈

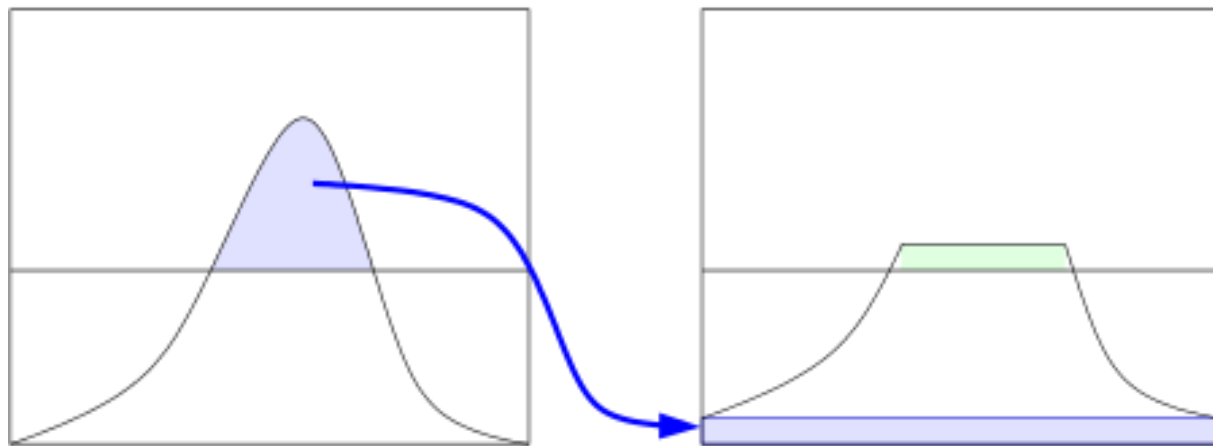
- 히스토그램 이퀄라이즈는 컬러 스케일에도 적용할 수 있는데, 밝기 값을 개선하기 위해서는 3개 채널 모두를 개선해야 하는 BGR 컬러 스페이스보다는 YUV나 HSV로 변환해서 밝기 채널만을 연산해서 최종 이미지에 적용하는 것이 좋다.
- 다음 예제는 YUV 컬러 스페이스로 변경한 컬러 이미지에 대한 이퀄라이즈를 보여준다.

```
9  img2 = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2BGR)
10
11  cv2.imshow('Before', img)
12  cv2.imshow('After', img2)
13  cv2.waitKey()
14  cv2.destroyAllWindows()
```

Section 05 히스토그램

□ CLAHE

- CLAHE(Contrast Limiting Adaptive Histogram Equalization)는 영상 전체에 이퀄라이즈를 적용했을 때 너무 밝은 부분이 날아가는 현상을 막기 위해 영상을 일정한 영역으로 나눠서 이퀄라이즈를 적용하는 것을 말한다.
- 노이즈가 증폭되는 것을 막기 위해 어느 히스토그램 계급(bin)이든 지정된 제한 값을 넘으면 그 픽셀은 다른 계급으로 배분하고나서 이퀄라이즈를 적용한다.



Section 05 히스토그램

□ CLAHE

- CLAHE를 위한 OpenCV 함수는 다음과 같다.
- `clahe = cv2.createCLAHE(clipLimit, tileGridSize)` : CLAHE 생성
 - clipLimit : Contrast 제한 경계 값, 기본 40.0
 - tileGridSize : 영역 크기, 기본 8 x 8
 - clahe : 생성된 CLAHE 객체
- `clahe.apply(src)` : CLAHE 적용
 - src : 입력 영상

Section 05 히스토그램

□ CLAHE

- 다음 예제는 CLAHE를 적용한 예제이다.

```
1 import cv2
2 import numpy as np
3
4 img = cv2.imread('./img/sky.jpg')
5 img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
6
7 img_eq = img_yuv.copy()
8 img_eq[:, :, 0] = cv2.equalizeHist(img_eq[:, :, 0])
9 img_eq = cv2.cvtColor(img_eq, cv2.COLOR_YUV2BGR)
10
```

Section 05 히스토그램

□ CLAHE

- 다음 예제는 CLAHE를 적용한 예제이다.

```
11  img_clahe = img_yuv.copy()
12  clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8,8))
13  img_clahe[:, :, 0] = clahe.apply(img_clahe[:, :, 0])
14  img_clahe = cv2.cvtColor(img_clahe, cv2.COLOR_YUV2BGR)
15
16  cv2.imshow('Before', img)
17  cv2.imshow('CLAHE', img_clahe)
18  cv2.imshow('equalizeHist', img_eq)
19  cv2.waitKey()
20  cv2.destroyAllWindows()
```

Section 05 히스토그램

□ 2D 히스토그램

- 1차원 히스토그램은 각 픽셀이 몇 개씩인지 세어서 그래프로 표현하는데, 2차원 히스토그램은 이와 같은 축이 2개이고 각각의 축이 만나는 지점의 개수를 표현한다.
- 그래서 이것을 적절히 표현하려면 지금까지 사용한 2차원 그래프가 아닌 3차원 그래프가 필요하다.
- 아래의 예제는 맑고 화창한 가을 하늘의 산을 찍은 사진을 2차원 히스토그램으로 표현한 것이다.

```
1 import cv2
2 import matplotlib.pyplot as plt
3
4 plt.style.use('classic')
5 img = cv2.imread('./img/seoul.jpg')
6
```

Section 05 히스토그램

□ 2D 히스토그램

- 아래의 예제는 맑고 화창한 가을 하늘의 산을 찍은 사진을 2차원 히스토그램으로 표현한 것이다.

```
7 plt.subplot(131)
8 hist = cv2.calcHist([img], [0,1], None, [32,32], [0,256,0,256])
9 p = plt.imshow(hist)
10 plt.title('Blue and Green')
11 plt.colorbar(p)
12
13 plt.subplot(132)
14 hist = cv2.calcHist([img], [1,2], None, [32,32], [0,256,0,256])
15 p = plt.imshow(hist)
16 plt.title('Green and Red')
17 plt.colorbar(p)
18
```

Section 05 히스토그램

□ 2D 히스토그램

- 아래의 예제는 맑고 화창한 가을 하늘의 산을 찍은 사진을 2차원 히스토그램으로 표현한 것이다.

```
19 plt.subplot(133)
20 hist = cv2.calcHist([img], [0,2], None, [32,32], [0,256,0,256])
21 p = plt.imshow(hist)
22 plt.title('Blue and Red')
23 plt.colorbar(p)
24
25 plt.show()
```

Section 05 히스토그램

□ 역투영

- 2차원 히스토그램과 HSV 컬러 스페이스를 이용하면 색상으로 특정 물체나 사물의 일부분을 배경에서 분리할 수 있다.
- 기본 원리는 물체가 있는 관심영역의 H와 V 값의 분포를 얻어낸 후 전체 영상에서 해당 분포의 픽셀 만 찾아내는 것이다.
- OpenCV는 역투영을 위해 아래와 같은 함수로 제공한다.

```
cv2.calcBackProject(img, channel, hist, ranges, scale)
```

- img : 입력 영상, [img] 처럼 리스트로 감싸서 표현
- channel : 처리할 채널, 리스트로 감싸서 표현
- 1 채널 : [0], 2채널 : [0 ,1], 3채널 : [0 ,1, 2]
- hist : 역투영에 사용할 히스토그램
- ranges : 각 픽셀이 가질 수 있는 값의 범위
- scale : 결과에 적용할 배율 계수

Section 05 히스토그램

□ 역투영

- 예제에서는 마우스로 선택한 특정 물체만 배경에서 분리해내는 모습을 보여주고 있다.

```
1 import cv2
2 import numpy as np
3
4 win_name = 'back_projection'
5 img = cv2.imread('../img/pump_horse.jpg')
6 hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
7 draw = img.copy()
8
9 def masking(bp, win_name):
10     disc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
11     cv2.filter2D(bp,-1,disc,bp)
12     _, mask = cv2.threshold(bp, 1, 255, cv2.THRESH_BINARY)
13     result = cv2.bitwise_and(img, img, mask=mask)
14     cv2.imshow(win_name, result)
15
```

Section 05 히스토그램

□ 역투영

- 예제에서는 마우스로 선택한 특정 물체만 배경에서 분리해내는 모습을 보여주고 있다.

```
16 def backProject_manual(hist_roi):
17     hist_img = cv2.calcHist([hsv_img], [0,1], None,[180,256], [0,180,0,256])
18     hist_rate = hist_roi / (hist_img + 1)
19     h,s,v = cv2.split(hsv_img)
20     bp = hist_rate[h.ravel(), s.ravel()]
21     bp = np.minimum(bp, 1)
22     bp = bp.reshape(hsv_img.shape[:2])
23     cv2.normalize(bp,bp, 0, 255, cv2.NORM_MINMAX)
24     bp = bp.astype(np.uint8)
25     masking(bp,'result_manual')
26
27 def backProject_cv(hist_roi):
28     bp = cv2.calcBackProject([hsv_img], [0, 1], hist_roi, [0, 180, 0, 256], 1)
29     masking(bp,'result_cv')
30
```


Section 05 히스토그램

□ 역투영

```
31 (x,y,w,h) = cv2.selectROI(win_name, img, False)
32 if w > 0 and h > 0:
33     roi = draw[y:y+h, x:x+w]
34     cv2.rectangle(draw, (x, y), (x+w, y+h), (0,0,255), 2)
35     hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
36     hist_roi = cv2.calcHist([hsv_roi],[0, 1], None, [180, 256], [0, 180, 0, 256] )
37     backProject_manual(hist_roi)
38     backProject_cv(hist_roi)
39
40 cv2.imshow(win_name, draw)
41 cv2.waitKey()
42 cv2.destroyAllWindows()
```

- 역투영의 장점은 알파 채널이나 크로마키 같은 보조 역할이 없어도 복잡한 모양의 사물을 분리할 수 있다는 것이다.
- 하지만 대상 사물의 색상과 비슷한 색상이 뒤섞여 있을 때는 효과가 떨어지는 단점도 있다.

Section 05 히스토그램

□ 히스토그램 비교

- 히스토그램은 영상의 픽셀 값의 분포를 갖는 정보이므로 이것을 비교하면 영상에 사용한 픽셀의 색상 비중이 얼마나 비슷한지 알 수 있다.
- 이것은 영상이 서로 얼마나 비슷한지를 알 수 있는 하나의 방법이다.
- OpenCV는 히스토그램을 비교해서 그 유사도가 얼마인지 판단해 주는 함수를 아래와 같이 제공한다.

```
cv2.compareHist( hist1, hist2, method)
```

- hist1, hist2 : **비교할 2개의 히스토그램**, 크기와 차원이 같아야 함
- method : **비교 알고리즘 선택 플래그 상수**
 - cv2.HISTCMP_CORREL : 상관관계(1: 완전 일치, - 1: 최대 불일치, 0: 무관계)
 - cv2.HISTCMP_CHISQR : 카이제곱 (0: 완전 일치, 큰 값(미정): 최대 불일치)
 - cv2.HISTCMP_INTERSECT : 교차(1: 완전 일치, 0 최대 불일치(1 로 정규화한 경우))
 - cv2.HISTCMP_BHATTACHARYYA : 바타차야 (0 : 완전 일치, 1: 최대 불일치)
 - cv2.HISTCMP_HELLINGER : HISTCMP_BHATTACHARYYA와 동일

Section 05 히스토그램

□ 히스토그램 비교

- 서로 다른 영상의 히스토그램을 같은 조건으로 비교하기 위해서는 먼저 히스토그램을 노멀라이즈해야 한다.
- 이미지가 크면 픽셀 수가 많고 당연히 히스토그램의 값도 더 커지기 때문이다.

Section 05 히스토그램

□ 히스토그램 비교

- 다음 예제는 다른 각도에서 찍은 태권브이 장난감 이미지 3개와 코주부 박사 장난감을 찍은 이미지를 비교해서 각 비교 알고리즘에 따른 결과를 보여준다.

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 img1 = cv2.imread('./img/taekwonv1.jpg')
6 img2 = cv2.imread('./img/taekwonv2.jpg')
7 img3 = cv2.imread('./img/taekwonv3.jpg')
8 img4 = cv2.imread('./img/dr_ochanomizu.jpg')
9
10 cv2.imshow('query', img1)
11 imgs = [img1, img2, img3, img4]
12 hists = []
```

Section 05 히스토그램

□ 히스토그램 비교

- 다음 예제는 다른 각도에서 찍은 태권브이 장난감 이미지 3개와 코주부 박사 장난감을 찍은 이미지를 비교해서 각 비교 알고리즘에 따른 결과를 보여준다.

```
13 for i, img in enumerate(imgs) :
14     plt.subplot(1,len(imgs),i+1)
15     plt.title('img%d'% (i+1))
16     plt.axis('off')
17     plt.imshow(img[:,::-1])
18     hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
19     hist = cv2.calcHist([hsv], [0,1], None, [180,256], [0,180,0, 256])
20     cv2.normalize(hist, hist, 0, 1, cv2.NORM_MINMAX)
21     hists.append(hist)
22
23 query = hists[0]
24 methods = {'CORREL':cv2.HISTCMP_CORREL, 'CHISQR':cv2.HISTCMP_CHISQR, 'INTERSECT':cv2.HISTCMP_INTERSECT, 'BHATTACHARYYA':cv2.HISTCMP_BHATTACHARYYA}
```

Section 05 히스토그램

□ 히스토그램 비교

- 다음 예제는 다른 각도에서 찍은 태권브이 장난감 이미지 3개와 코주부 박사 장난감을 찍은 이미지를 비교해서 각 비교 알고리즘에 따른 결과를 보여준다.

```
25 for j, (name, flag) in enumerate(methods.items()):
26     print('%-10s'%name, end='\t')
27     for i, (hist, img) in enumerate(zip(hists, imgs)):
28         ret = cv2.compareHist(query, hist, flag)
29         if flag == cv2.HISTCMP_INTERSECT:
30             ret = ret/np.sum(query)
31         print("img%d:%7.2f"% (i+1 , ret), end='\t')
32     print()
33
34 plt.show()
```

Q&A

