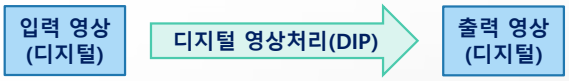
CHAPTER 06
화소처리

6.1 영상 화소의 접근
6.2 화소 밝기 변환
6.3 히스토그램
6.4 컬러 공간 변환

1

6.1 영상 화소의 접근

◆영상처리
- ❖ 2차원 데이터에 대한 행렬 연산
- ❖ 영상을 2차원행렬로 읽고 계산하고 저장하기



◆영상처리 프로그래밍을 한다는 것
- ❖ 영상이라는 2차원 데이터의 원소값을 개발자가 원하는 방향으로 변경하는 것
- ❖ 영상을 다루려면 기본적으로 영상의 화소 접근, 값 수정, 새로 만들 수 있어야 함
- ❖ 2중 for문 처리

```
##for y in range(100, 400):
##    for x in range(200, 300):
##        img[y, x] = 0


##for y in range(100, 400):
##    for x in range(200, 300):
##        img[y, x] = [255, 0, 0]     # 파랑색(blue)으로 변경
```

2

6.1 영상 화소의 접근

예제 6.1.1 행렬 원소 접근 방법 – 01.mat_access.py

```
01  import numpy as np
02
03  def mat_access1(mat):                          # 원소 직접 접근 방법
04      for i in range(mat.shape[0]):
05          for j in range(mat.shape[1]):
06              k = mat[i, j]                      # 원소 접근– mat[i][j] 방식도 가능
07              mat[i, j] = k * 2                  # 원소 할당
08
09  def mat_access2(mat):                          # item() , itemset() 함수 사용방식
10      for i in range(mat.shape[0]):
11          for j in range(mat.shape[1]):
12              k = mat.item(i, j)                 # 원소 접근
13              mat.itemset((i, j), k * 2)         # 원소 할당
14
15  mat1 = np.arange(10).reshape(2, 5)             # 0~10 사이 원소 생성
16  mat2 = np.arange(10).reshape(2, 5)
17
18  print("원소 처리 전: \n%s\n" % mat1)
19  mat_access1(mat1)
20  print("원소 처리 후: \n%s\n" % mat1)
21
22  print("원소 처리 전: \n%s\n" % mat2)
23  mat_access2(mat2)
24  print("원소 처리 후: \n%s" % mat2)
```

```
Run:  01.mat_access
  D:/source/chap06/01.mat_access.py
원소 처리 전:
[[0 1 2 3 4]
 [5 6 7 8 9]]

원소 처리 후:
[[ 0  2  4  6  8]
 [10 12 14 16 18]]

원소 처리 전:
[[0 1 2 3 4]
 [5 6 7 8 9]]

원소 처리 후:
[[ 0  2  4  6  8]
 [10 12 14 16 18]]
```

3

6.1 영상 화소의 접근

예제 6.1.2 Mat::ptr()을 통한 행렬 원소 접근 – 02.image_access.py

```
01  import numpy as np, cv2, time                  # 수행시간 계산 위해 time 모듈 임포트
02
03  def pixel_access1(image):                      # 화소 직접 접근 방법
04      image1 = np.zeros(image.shape[:2], image.dtype)
05      for i in range(image.shape[0]):
06          for j in range(image.shape[1]):
07              pixel = image[i,j]                 # 화소 접근
08              image1[i, j] = 255 - pixel         # 화소 할당
09  return image1
10
11  def pixel_access2(image):                      # item() 함수 접근 방법
12      image2 = np.zeros(image.shape[:2], image.dtype)
13      for i in range(image.shape[0]):
14          for j in range(image.shape[1]):
15              pixel = image.item(i, j)           # 화소 접근
16              image2.itemset((i, j), 255 - pixel)  # 화소 할당
17      return image2
18
19  def pixel_access3(image):                      # 룩업테이블 이용 방법
20      lut = [255 - i for i in range(256)]        # 룩업테이블 생성
21      lut = np.array(lut, np.uint8)
22      image3 = lut[image]
23      return image3
24
```

4

6.1 영상 화소의 접근

```
25  def pixel_access4(image):                          # OpenCV 함수 이용 방법
26      image4 = cv2.subtract(255, image)
27      return image4
28
29  def pixel_access5(image):                          # ndarray 산술 연산 방법
30      image5 = 255 - image
31      return image5
32
33  image = cv2.imread("images/bright.jpg", cv2.IMREAD_GRAYSCALE)
34  if image is None: raise Exception("영상파일 읽기 오류")
35
36  ## 수행시간 체크 함수                  함수명
37  def time_check(func, msg):
38      start_time = time.perf_counter()
39      ret_img = func(image)
40      elapsed = (time.perf_counter() - start_time) * 1000
41      print(msg, "수행시간 : %0.2f ms" % elapsed )
42      return ret_img
43                                  수행할 함수명
44  image1 = time_check(pixel_access1, "[방법1] 직접 접근 방식")
45  image2 = time_check(pixel_access2, "[방법2] item() 함수 방식")
46  image3 = time_check(pixel_access3, "[방법3] 룩업테이블 방식")
47  image4 = time_check(pixel_access4, "[방법4] OpenCV 함수 방식")
48  image5 = time_check(pixel_access5, "[방법5] ndarray 연산 방식")
```

```
Run: 02.image_access
C:\Python\python.exe D:/source/chap06/02.image_access.py
[방법 1] 직접 접근 방식 수행시간 : 586.93 ms
[방법 2] item() 함수 방식 수행시간 : 65.21 ms
[방법 3] 룩업 테이블 방식 수행시간 : 0.68 ms
[방법 4] OpenCV 함수 방식 수행시간 : 0.19 ms
[방법 5] ndarray 연산 방식 수행시간 : 0.16 ms
```

5

6.2 화소 밝기 변환
6.2.1 그레이 스케일(명암도) 영상

◆흑백 영상 ?

❖ 단어 자체의 의미: 검은색과 흰색의 영상, 의미 안맞음

◆그레이 스케일(gray-scale) 영상 , 명암도 영상

❖화소값은 0~255의 값을 가지는데 0은 검은색을, 255는 흰색을 의미

❖0~255 사이의 값들은 다음과 같이 진한 회색에서 연한 회색

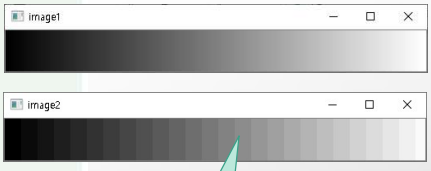
6

6.2.1 그레이 스케일(명암도) 영상

예제 6.2.1 명암도 영상 생성 – grayscale_image.py

```
01  import numpy as np
02  import cv2
03
04  image1 = np.zeros((50, 512), np.uint8)          # 50 x 512 영상 생성
05  image2 = np.zeros((50, 512), np.uint8)
06  rows, cols = image1.shape[:2]
07
08  for i in range(rows):                           # 행렬 전체 조회
09      for j in range(cols):
10          image1.itemset((i, j), j // 2)          # 화소값 점진적 증가
11          image2.itemset((i, j), j // 20*10)      # 계단 현상 증가
12
13  cv2.imshow("image1", image1)
14  cv2.imshow("image2", image2)
15  cv2.waitKey(0)
```

나눗셈에서 몫



계단 현상

7

6.2.2 영상의 화소 표현

예제 6.2.2 영상 화소값 확인 – 03.pixel_value.py

```
01  import cv2
02
03  image = cv2.imread("images/pixel.jpg", cv2.IMREAD_GRAYSCALE) # 영상 읽기
04  if image is None: raise Exception("영상파일 읽기 오류")
05
06  (x, y), (w, h) = (180, 37), (15, 10)            # 좌표는 x, y
07  roi_img = image[y:y+h, x:x+w]                   # 행렬 접근은 y, x
08
09  #print("[roi img] =\n", roi_img)                 # 행렬 원소 바로 출력 가능
10
11  print("[roi_img] =")
12  for row in roi_img:                             # 원소 순회 방식 출력
13      for p in row:
14          print("%4d" % p, end=" ")               # 순회 원소 하나씩 출력
15  print()
16
17  cv2.rectangle(image, (x, y, w, h), 255, 1)      # 관심 영역에 사각형 표시
18  cv2.imshow("image", image)
19  cv2.waitKey(0)
```

슬라이스 연산자 통한 관심 영역 지정

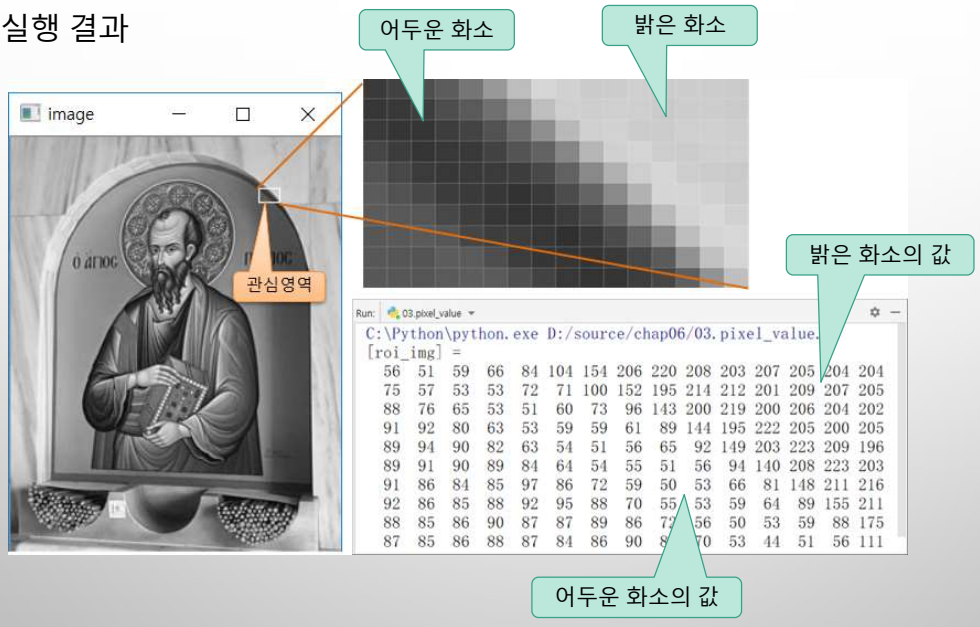
사각형 튜플

```
Run: 03.pixel_value
C:\Python\python.exe D:/source/chap06/03.pixel_value.py
Traceback (most recent call last):
  File "D:/source/chap06/03.pixel_value.py", line 4, in <module>
    if image is None: raise Exception("영상 파일 읽기 오류")
Exception: 영상 파일 읽기 오류
```

8

6.2.2 영상의 화소 표현

◆실행 결과



9

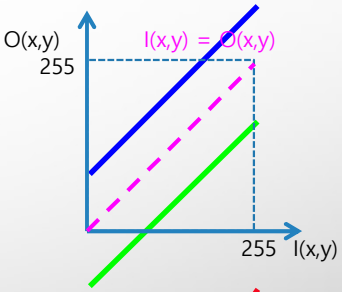
화소 처리

◆입력 영상의 크기 (W×H)와 출력 영상의 크기 (W×H)가 같음

◆입력 영상 특정 위치의 픽셀값 (`I(x,y)`)을 변경하여 출력 영상 **같은 위치**의 픽셀값 (`O(x,y)`)으로 정함

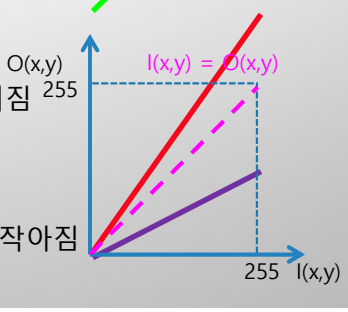
영상의 밝기 조절

- **덧셈 연산**
  - ✓ I(x,y) + α = O(x,y) ➔ 값이 커짐 ➔ 밝아짐
- **뺄셈 연산**
  - ✓ I(x,y) - α = O(x,y) ➔ 값이 작아짐 ➔ 어두워짐



영상의 명암 대비 조절

- **곱셈 연산**
  - ✓ I(x,y) * α = O(x,y) ➔ 값이 커짐, 값의 차이가 커짐 ➔ 대비가 커짐
- **나눗셈 연산**
  - ✓ I(x,y) / α = O(x,y) ➔ 값이 작아짐, 값의 차이가 작아짐 ➔ 대비가 작아짐



10

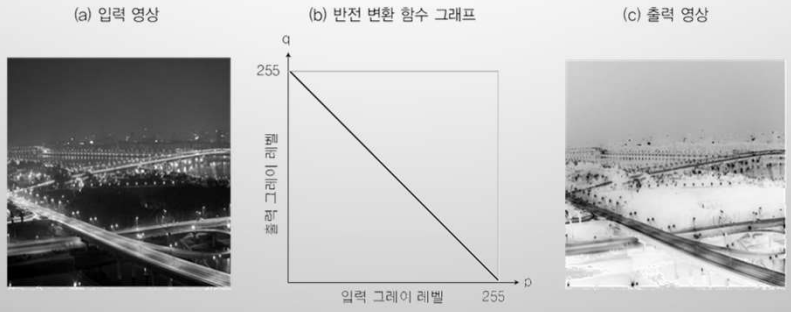
영상의 반전 변환

◆사진학적 역변환 (NEGATIVE/REVERSE TRANSFORM)

◆각 화소의 값이 영상 내에 대칭이 되는 값으로 변환

❖8비트 그레이 레벨의 영상을 반전시키면 화소 값 0번은 255번으로, 화소 값 1번은 254번으로 변환됨

◆반전 변환의 변환 함수 : **255 - I**(X,Y)



[그림 4-15] 8비트 그레이 레벨 영상에서의 반전 변환 함수 그래프와 출력 영상

11

산술연산의 문제점과 해결 방법

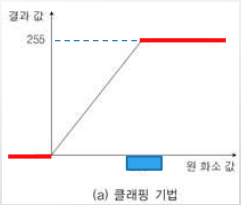
◆문제점

❖ 결과 값이 화소의 최대값과 최소값을 넘을 수 있음.

◆해결 방법

① 클래핑(CLAMPING) 기법, SATURATION 방식

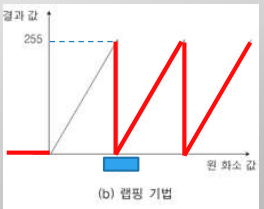
- ●연산의 결과 값이 최소값보다 작으면 그 결과 값을 최소값으로, 최대값보다 크면 결과 값을 최대값으로 하는 기법
- ●8비트 그레이 영상 : 음수는 0으로 설정하고, 255보다 큰 값은 255로 설정함.



(a) 클래핑 기법

② 랩핑(WRAPING) 기법, MODULO 방식

- ●연산의 결과 값이 최소값보다 작으면 그 결과 값을 최소값으로, 최대값보다 크면 최소값부터 최대값까지를 한 주기로 해서 이를 반복하는 기법
- ●8비트 그레이 영상 : 음수는 0으로, 255보다 큰 결과 값 256은 0으로, 257은 1로 설정한 후 이런 방식으로 주기를 계속 반복



(b) 랩핑 기법

12

6.2.3 밝기의 가감 연산

◆modulo 방식과 saturation 방식 비교

예제 6.2.3 행렬 가감 연산 통한 영상 밝기 변경 – 04.bright_dark.py

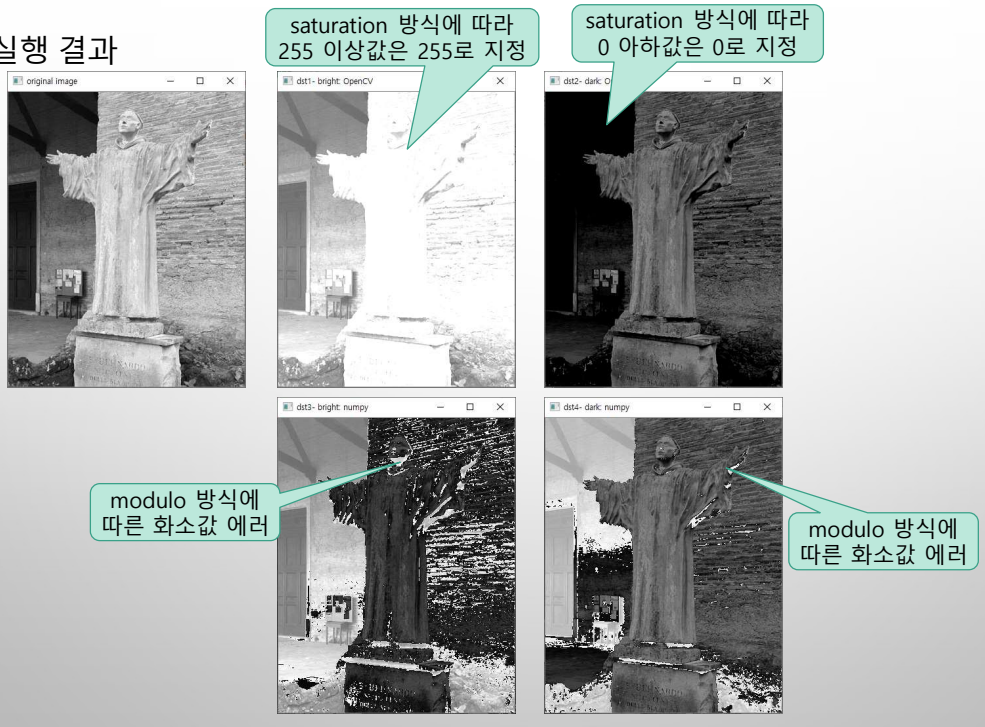
```
01  import cv2
02
03  image = cv2.imread("images/bright.jpg", cv2.IMREAD_GRAYSCALE)    # 영상 읽기
04  if image is None: raise Exception("영상파일 읽기 오류")
05
06  ## OpenCV 함수 이용(saturation 방식)
07  dst1 = cv2.add(image, 100)                                       # 영상 밝게
08  dst2 = cv2.subtract(image, 100)                                  # 영상 어둡게
09
10  ## numpy.ndarray 이용(modulo 방식)
11  dst3 = image + 100                                               # 영상 밝게
12  dst4 = image - 100                                               # 영상 어둡게
13
14  cv2.imshow("original image", image)
15  cv2.imshow("dst1- bright:OpenCV", dst1)
16  cv2.imshow("dst2- dark:OpenCV", dst2)
17  cv2.imshow("dst3- bright:numpy", dst3)
18  cv2.imshow("dst4- dark:numpy", dst4)
19  cv2.waitKey(0)
```

OpenCV와 numpy의 0 미만과 255 이상의 화소값 처리 방식이 다름에 주의
- OpenCV : 250 + 100 = 360 → 255 (saturation 방식)
- numpy : 250 + 100 = 350 % 256 → 104 (modulo 방식)

13

6.2.2 영상의 화소 표현

◆실행 결과



14

6.2.4 행렬 덧셈 및 곱셈을 이용한 영상 합성

심화예제 6.2.4 행렬 합과 곱 연산을 통한 영상 합성 – 05.image_synthesis.py

```
01  import numpy as np, cv2
02
03  image1 = cv2.imread("images/add1.jpg", cv2.IMREAD_GRAYSCALE)     # 영상 읽기
04  image2 = cv2.imread("images/add2.jpg", cv2.IMREAD_GRAYSCALE)
05  if image1 is None or image2 is None: raise Exception("영상파일 읽기 오류")
06
07  ## 영상 합성 방법
08  alpha, beta = 0.6, 0.7                                          # 곱셈 비율
09  add_img1 = cv2.add(image1, image2)                              # 두 영상 단순 더하기
10  add_img2 = cv2.add(image1 * alpha, image2 * beta)               # 두 영상 비율에 따른 더하기
11  add_img2 = np.clip(add_img2, 0, 255).astype('uint8')            # saturation 처리
12  add_img3 = cv2.addWeighted(image1, alpha, image2, beta, 0) # 두 영상 비율에 따른 더하기
13
14  titles = ['image1', 'image2', 'add_img1', 'add_img2', 'add_img3'] # 윈도우 이름
15  for t in titles: cv2.imshow(t, eval(t))                         # 영상 표시
16  cv2.waitKey(0)
```

1대1 합성 – 화소값이 255가 넘는 경우들이 생겨 밝은 값으로 나타남

1) dst(y,x) = image1(y,x) * 0.5 + image2(y,x) * 0.5 ;
2) dst(y,x) = image1(y,x) * alpha + image2(y,x) * (1-alpha)
3) dst(y,x) = image1(y,x) * alpha + image2(y,x) * beta

비율 조정하여 합성

15

6.2.5 명암 대비

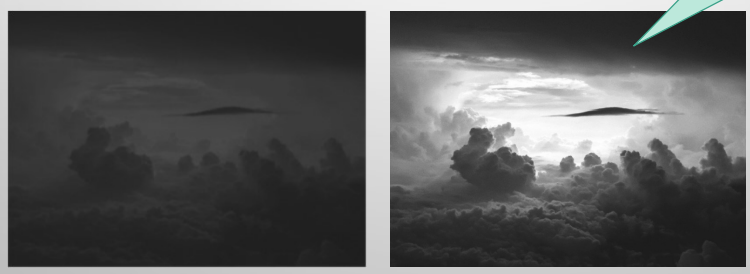
◆대비

❖ 같은 색도 인접한 색의 밝기에 따라서 다르게 보임

〈그림 6.2.2〉 밝기 대비 예시

◆대비 영상의 예

히스토그램 평활화한 영상



16

6.2.5 명암 대비

예제 6.2.5 영상 대비 변경 – 06.contrast.py

```
01  import numpy as np, cv2
02
03  image = cv2.imread("images/contrast.jpg", cv2.IMREAD_GRAYSCALE) # 영상 읽기
04  if image is None: raise Exception("영상파일 읽기 오류")
05
06  noimage = np.zeros(image.shape[:2], image.dtype)                # 더미 영상
07  avg = cv2.mean(image)[0]/2.0                                    # 영상 화소 평균의 절
08
09  dst1 = cv2.scaleAdd(image, 0.5, noimage)                        # 명암 대비 감소
10  dst2 = cv2.scaleAdd(image, 2.0, noimage)                        # 명암 대비 증가
11  dst3 = cv2.addWeighted(image, 0.5, noimage, 0, avg)             # 명암 대비 감소
12  dst4 = cv2.addWeighted(image, 2.0, noimage, 0, -avg)            # 명암 대비 증가
13
14  cv2.imshow("image", image)                                      # 영상 띄우기
15  cv2.imshow("dst1 - decrease contrast", dst1)
16  cv2.imshow("dst2 - increase contrast", dst2)
17  cv2.imshow("dst3 - decrease contrast using average", dst3)
18  cv2.imshow("dst4 - increase contrast using average", dst4)
19  cv2.waitKey(0)
```

곱셈으로 영상 대비 변경 (감소 및 증가)

영상 평균값을 활용하여 대비 변경시 화질 개선

17

6.3 히스토그램
6.3.1 히스토그램 개념

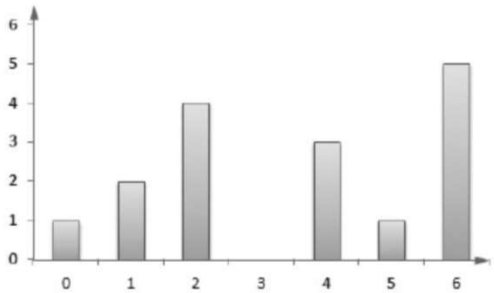
◆히스토그램

❖ "관측값의 개수를 겹치지 않는 다양한 계급으로 표시하는 것"

❖ 어떤 데이터가 얼마나 많은지를 나타내는 도수 분포표를 그래프로 나타낸 것

5	4	6	6
2	1	6	4
2	2	4	6
1	6	0	2

(a) 입력 영상



(b) 히스토그램

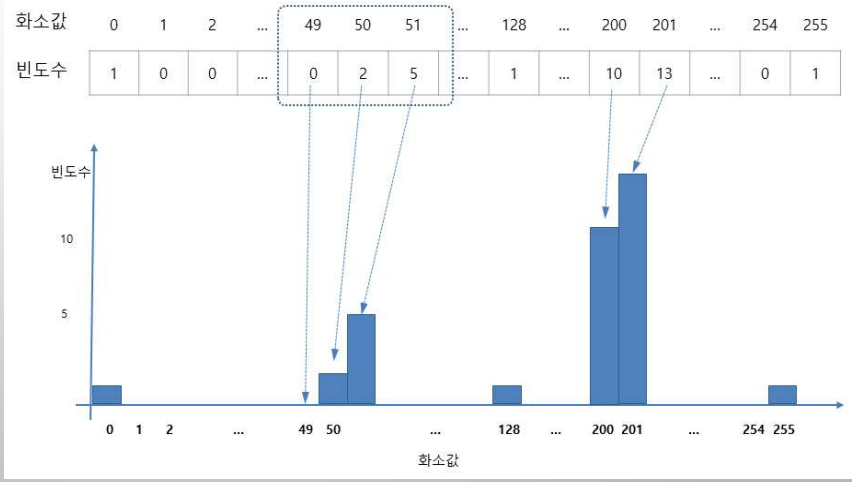
〈그림 6.3.1〉 히스토그램의 개념

18

6.3.2 히스토그램 계산

◆히스토그램 계산 및
그래프 그리기 예시

```
Run:  03.pixel_value
C:\Python\python.exe D:/source/chap06/03.pixel_value.py
[roi_img] =
  56  51  59  66  84 104 154 206 220 208 203 207 205 204 204
  75  57  53  53  72  71 100 152 195 214 212 201 209 207 205
  88  76  65  53  51  60  73  96 143 200 219 200 206 204 202
  91  92  80  63  53  59  59  61  89 144 195 222 205 200 205
  89  94  90  82  63  54  51  56  65  92 149 203 223 209 196
  89  91  90  89  84  64  54  55  51  56  94 140 208 223 203
  91  86  84  85  97  86  72  59  50  53  66  81 148 211 216
  92  86  85  88  92  95  88  70  55  53  59  64  89 155 211
  88  85  86  90  87  87  89  86  72  56  50  53  59  88 175
  87  85  86  88  87  84  86  90  86  70  53  44  51  56 111
```



19

6.3.2 히스토그램 계산

◆단일 채널 히스토그램 구현 함수

예제 6.3.1 영상 히스토그램 계산 – 08.calc_histogrm.opencv.py(일부)

```
01  import numpy as np, cv2
02
03  def calc_histo(image, histSize, ranges=[0, 256] ):      # 행렬 원소의 1차원 히스토그램 계산
04      hist = np.zeros((histSize, 1), np.float32)          # 히스토그램 누적 행렬
05      gap = ranges[1] / histSize                          # 계급 간격
06
07      for row in image:                                   # 2차원 행렬 순회 방식
08          for pix in row:
09              idx = int(pix/gap)
10              hist[idx] += 1
11      return hist
```

20

6.3.3 OpenCV 함수 활용

함수 및 인수 구조		
cv2.calcHist (images , channels , mask , histSize , ranges [, hist [, accumulate]]) → ret ■설명: 행렬의 원속값의 빈도를 계산한다.		
인수 설명	■images	원본 배열들 – CV_8U 혹은 CV_32F 형으로 크기가 같아야 함
	■channels	히스토그램 계산에 사용되는 차원 목록
	■mask	특정 영역만 계산하기 위한 마스크 행렬 – 입력 영상과 같은 크기의 8비트 배열
	■histSize	각 차원의 히스토그램 배열 크기 – 계급(bin)의 개수
	■ranges	각 차원의 히스토그램의 범위
	■accumulate	누적 플래그– 여러 배열에서 단일 히스토그램을 구할 때 사용

21

6.3.3 OpenCV 함수 활용

예제 6.3.1 영상 히스토그램 계산 – 08.calc_histogram_opencv.py

```
01  import numpy as np, import cv2
02
03  def calc_histo(image, histSize, ranges=[0, 256]): …          # 소스 내용 생략
12
13  image = cv2.imread("images/pixel_test.jpg", cv2.IMREAD_GRAYSCALE)        # 영상 읽기
14  if image is None: raise Exception("영상파일 읽기 오류")
15
16  histSize, ranges = [32], [0, 256]                    # 히스토그램 간격수, 값 범위
17  gap = ranges[1]/histSize[0]                          # 계급 간격
18  ranges_gap = np.arange(0, ranges[1]+1, gap)          # 넘파이 계급범위&간격
19  hist1 = calc_histo(image, histSize, ranges)          # User 함수
20  hist2 = cv2.calcHist([image], [0], None, histSize, ranges)      # OpenCV 함수
21  hist3, bins = np.histogram(image, ranges_gap )
22
23  print("User 함수: \n", hist1.flatten())
24  print("OpenCV 함수: \n", hist2.flatten())
25  print("numpy 함수: \n", hist3)
```

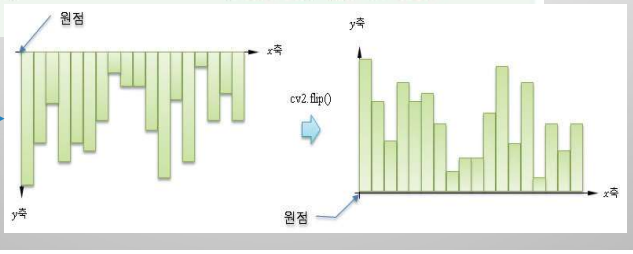

```
Run: 08.calc_histogram_opencv
C:\Python\python.exe D:/source/chap06/08.calc_histogram_opencv.py
User 함수:
 [  97.  247.  563. 1001. 1401. 1575. 1724. 1951. 2853. 3939. 3250. 2549.
 2467. 2507. 2402. 2418. 2727. 3203. 3410. 3161. 2985. 2590. 3384. 4312.
 4764. 3489. 2802. 2238. 1127.  628.  199.   37.]
OpenCV 함수:
 [  97.  247.  563. 1001. 1401. 1575. 1724. 1951. 2853. 3939. 3250. 2549.
 2467. 2507. 2402. 2418. 2727. 3203. 3410. 3161. 2985. 2590. 3384. 4312.
 4764. 3489. 2802. 2238. 1127.  628.  199.   37.]
numpy 함수:
 [  97  247  563 1001 1401 1575 1724 1951 2853 3939 3250 2549 2467 2507
 2402 2418 2727 3203 3410 3161 2985 2590 3384 4312 4764 3489 2802 2238
 1127  628  199   37]
```

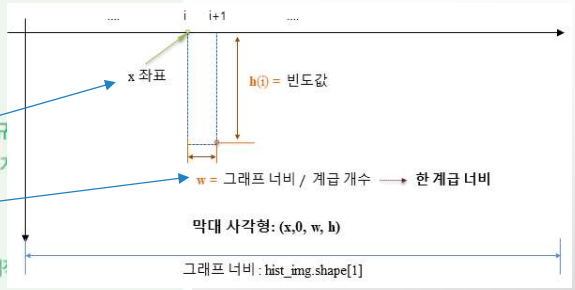
22

6.3.3 OpenCV 함수 활용

예제 6.3.3 히스토그램 그래프 그리기 – 09.draw_histogram.py

```
01  import numpy as np, cv2
02
03  def draw_histo(hist, shape=(200, 256)):
04      hist_img = np.full(shape, 255, np.uint8)
05      cv2.normalize(hist, hist, 0, shape[0], cv2.NORM_MINMAX)   # 정규
06      gap = hist_img.shape[1]/hist.shape[0]                     # 한 
07
08      for i, h in enumerate(hist):
09          x = int(round(i * gap))                               # 막대 사각형 사
10          w = int(round(gap))
11          cv2.rectangle(hist_img, (x, 0, w, int(h)), 0, cv2.FILLED)
12
13      return cv2.flip(hist_img, 0)                              # 영상 상하 뒤집기 후 반환
14
```



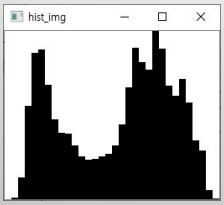


23

6.3.3 OpenCV 함수 활용

```
15  image = cv2.imread("images/pixel.jpg", cv2.IMREAD_GRAYSCALE)  # 영상 읽기
16  if image is None: raise Exception("영상파일 읽기 오류")
17
18  hist = cv2.calcHist([image], [0], None, [32], [0, 256])
19  hist_img = draw_histo(hist)
20
21  cv2.imshow("image", image)
22  cv2.imshow("hist_img", hist_img)
23  cv2.waitKey(0)
```

◆실행결과



24

6.3.3 OpenCV 함수 활용

심화예제 6.3.4 색상 히스토그램 그리기 – hue_histogram.py

```
01 import numpy as np, cv2
02
03 def make_palette(rows):                              # hue 채널 팔레트 행렬 생성 함수
04     ## 리스트 생성 방식
05     hue = [round(i * 180 / rows) for i in range(rows)] # hue 값 리스트 계산
06     hsv = [[[h, 255, 255]] for h in hue]               # (hue, 255,255) 화소값 계산
07     hsv = np.array(hsv, np.uint8)                      # 정수(uint8)형 행렬 변환
08     ## 반복문 방식
09     # hsv = np.full((rows, 1, 3), (255, 255, 255), np.uint8)
10     # for i in range(0, rows):                         # 행수만큼 반복
11     #     hue = round(i / rows * 180 )                 # 색상 계산
12     #     hsv[i] = (hue, 255, 255)                     # HSV 컬러 지정
13
14     return cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)        # HSV 컬러→BGR 컬러
15
16 def draw_hist_hue(hist, shape=(200, 256, 3)):         # 색상 히스토그램 그리기 함수
17     hsv_palette = make_palette(hist.shape[0])          # 색상 팔레트 생성
18     hist_img = np.full(shape, 255, np.uint8)
19     cv2.normalize(hist, hist, 0, shape[0], cv2.NORM_MINMAX) # 영상 높이값으로 정규화
20
21     gap = hist_img.shape[1] / hist.shape[0]                 # 한 계급 크기
22     for i, h in enumerate(hist):
23         x, w = int(round(i * gap)), int(round(gap))
24         color = tuple(map(int, hsv_palette[i][0]))          # 정수형 튜플로 변환
25         cv2.rectangle(hist_img, (x, 0, w, int(h)), color, cv2.FILLED)  # 팔레트 색으로 그리기
26
27     return cv2.flip(hist_img, 0)
28
```

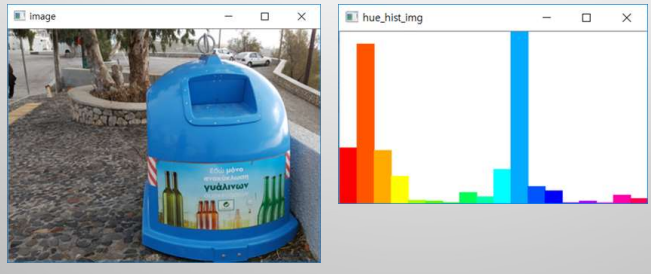
반복문 방식

정수형 변경 후, 튜플로

6.3.3 OpenCV 함수 활용

```
29 image = cv2.imread("images/hue_hist.jpg", cv2.IMREAD_COLOR)  # 영상 읽기
30 if image is None: raise Exception("영상파일 읽기 오류")
31
32 hsv_img = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)            # BGR 컬러→HSV 컬러
33 hue_hist = cv2.calcHist( [hsv_img], [0], None, [18], [0,180]) # Hue 채널 히스토그램 계산
34 hue_hist_img = draw_hist_hue(hue_hist, (200, 360, 3))       # 히스토그램 그래프
35
36 cv2.imshow("image", image)
37 cv2.imshow("hue_hist_img", hue_hist_img)
38 cv2.waitKey(0)
```

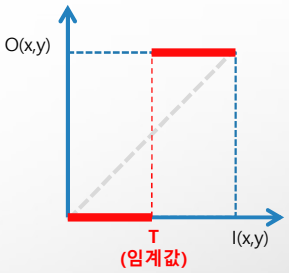
◆실행결과

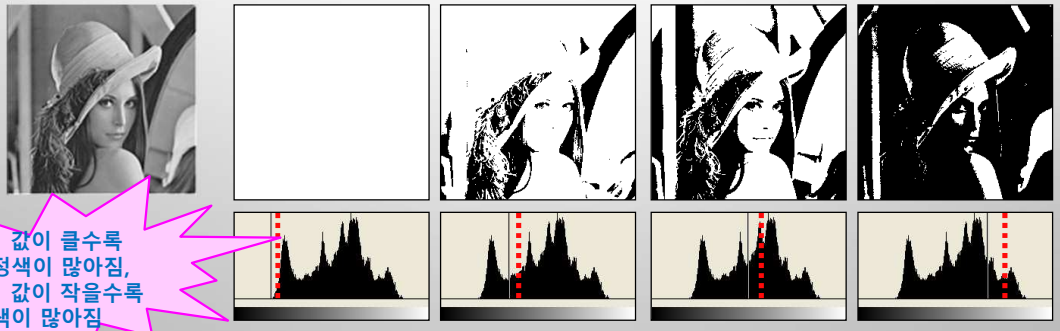


임계값(THRESHOLD) 영상

- **이진화**
  - 2가지 값을 갖는 영상으로 변환함으로써 영상의 분석을 용이하게 할 수 있음
  - 영상의 배경과 물체 분리에 자주 이용

$$O(x,y) = \begin{cases} 255 & I(x,y) \geq T \\ 0 & I(x,y) < T \end{cases}$$



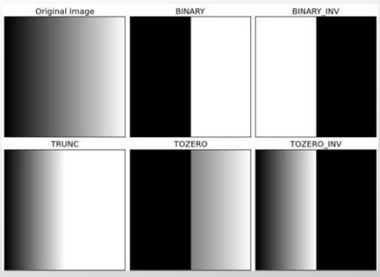


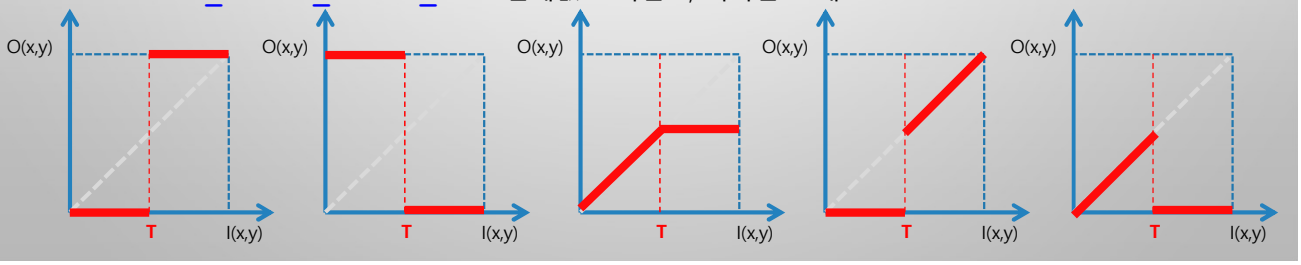
T의 값이 클수록 검정색이 많아짐,
T의 값이 작을수록 흰색이 많아짐

27

임계값(THRESHOLD) 영상

- **이진화**
  - ❖ `CV2.THRESHOLD(SRC, THRESH, MAX_VAL, TYPE[, DST]) ➔ RETVAL, DST`
    - SRC 입력 영상을 THRESH 임계값으로 이진화하여 DST로 반환하는 함수
    - MAX_VAL은 임계값을 넘을 때 적용한 최대 값
    - **TYPE**
      - **CV_THRESH_BINARY** : 임계값 초과는 255, 이하는 0
      - **CV_THRESH_BINARY_INV** : 임계값 초과는 0, 이하는 255
      - **CV_THRESH_TRUNC** : 임계값 초과는 임계값, 이하는 그대로
      - **CV_THRESH_TOZERO** : 임계값 초과는 그대로, 이하는 0
      - **CV_THRESH_TOZERO_INV** : 임계값 초과는 0, 이하는 그대로





28

6.3.4 히스토그램 스트레칭

◆히스토그램의 분포가 좁아서 영상의 대비가 좋지 않은 영상

(a) 밝은 부분을 많이 분포하는 영상과 히스토그램

(b) 어두운 부분을 많이 분포하는 영상과 히스토그램

29

6.3.4 히스토그램 스트레칭

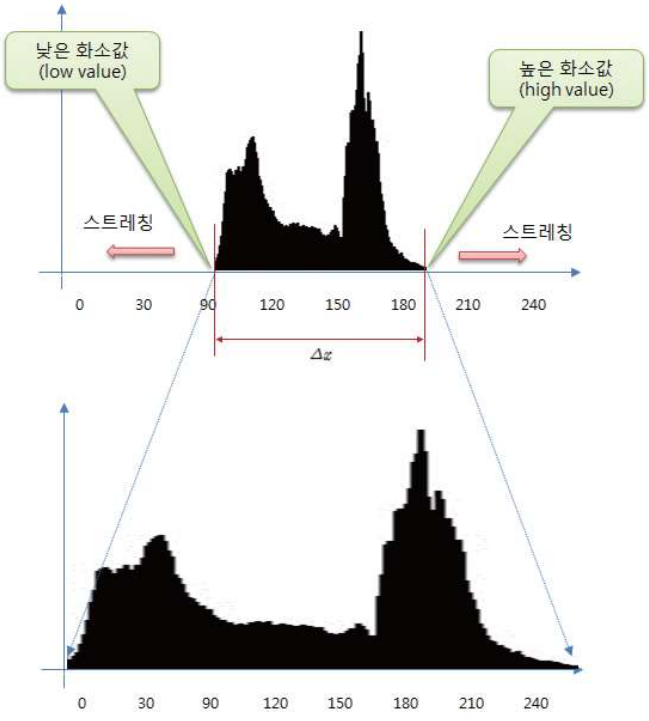
◆히스토그램 스트레칭

❖ 명암 분포가 좁은 히스토그램을 좌우로 잡아당겨(스트레칭해서) 고른 명암 분포를 가진 히스토그램이 되게 하는 것

$$새\ 화소값 = \frac{(화소값 - low)}{high - low} * 255$$

64	64	64	64
143	143	191	223
239	143	143	191
207	191	143	255

Histogram Stretching

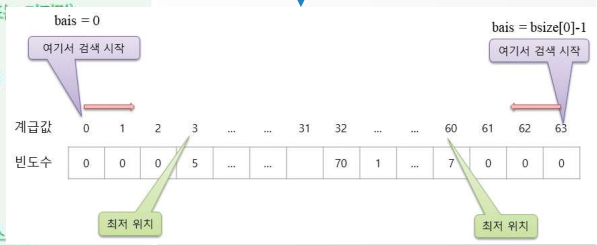


30

6.3.4 히스토그램 스트레칭

예제 6.3.5 히스토그램 스트래칭 – histogram_stretching.py

```
01  import numpy as np, cv2
02  from Common.histogram import draw_histo              # 함수 재사용 위한 임포트
03
04  def search_value_idx(hist, bias=0):                  # 값있는 첫 계급 검색 함수
05      for i in range(hist.shape[0]):
06          idx = np.abs(bias - i)                       # 검색 위치(처음 또
07          if hist[idx] > 0: return idx                 # 위치 반환
08      return -1                                        # 대상 없으면 반환
09
10  image = cv2.imread("images/hist_stretch.jpg", cv2.IMREAD_GRAYSCALE)
11  if image is None: raise Exception("영상파일 읽기 오류")
12
13  bsize, ranges = [64], [0,256]                        # 계급 개수 및 화소
14  hist = cv2.calcHist([image], [0], None, bsize, ranges)
15
16  bin_width = ranges[1]/bsize[0]                       # 한 계급 너비
17  low = search_value_idx(hist, 0) * bin_width          # 최저 화소값
18  high= search_value_idx(hist, bsize[0] - 1) * bin_width   # 최고 화소값
19
20  idx = np.arange(0, 256)                              # 룩업 인덱스(0~255) 생성
21  idx = (idx - low)/(high- low) * 255                  # 수식 적용하여 룩업 인덱스 완성
22  idx[0:int(low)] = 0                                  # 히스토그램 하위 부분
23  idx[int(high+1):] = 255                              # 히스토그램 상위 부분
24
```



31

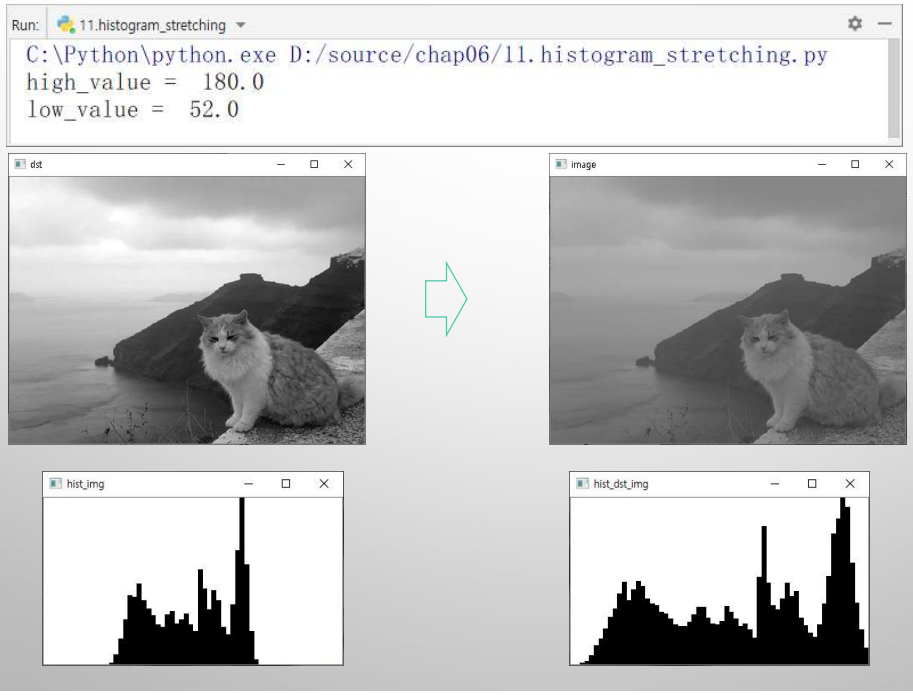
6.3.4 히스토그램 스트레칭

```
25  dst = cv2.LUT(image, idx.astype('uint8'))            # 룩업 테이블 사용
26  ## 룩업 테이블 사용하지 않고 직접 구현
27  # dst = np.zeros(image.shape, dtype=image.dtype)
28  # for i in range(dst.shape[0]):
29  #         for j in range(dst.shape[1]):
30  #             dst[i,j] = idx[image[i,j]]
31
32  hist_dst = cv2.calcHist([dst], [0], None, bsize, ranges)# 결과 영상 히스토그램 재계산
33  hist_img = draw_histo(hist, (200,360))               # 원본 영상 히스토그램 그리기
34  hist_dst_img = draw_histo(hist_dst, (200,360))       # 결과 영상 히스토그램 그리기
35
36  print("high_vlue =", high)
37  print("low_vlue =", low)
38  cv2.imshow("image", image);     cv2.imshow("hist_img", hist_img)
39  cv2.imshow("dst", dst);         cv2.imshow("hist_dst_img", hist_dst_img)
40  cv2.waitKey(0)
```

32

6.3.4 히스토그램 스트레칭

◆실행결과



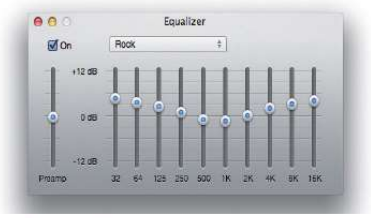
33

6.3.5 히스토그램 평활화

◆이퀄라이저(Equalizer)

❖ 주파수 특성을 균등하게 보정하는 기기

❖ 주파수 특성을 어느 특정의 목적에 맞추어 임의로 변화시켜 원하는 음색



〈그림 6.3.4〉 MP3 플레이어의 이퀄라이저의 예

◆평활화 알고리즘

❖ 히토그램 평활화의 사전적 의미인 "분포의 균등"이라는 방법을 이용해 명암 대비 증가

❖ → 영상의 인지도 높이며, 영상의 화질 개선

① 영상의 히스토그램을 계산한다.
② 히스토그램 빈도값에서 누적 빈도수(누적합)를 계산한다.
③ 누적 빈도수를 정규화(정규화 누적합) 한다.
④ 결과 화소값 = 정규화 누적합 * 최대 화소값

34

6.3.5 히스토그램 평활화

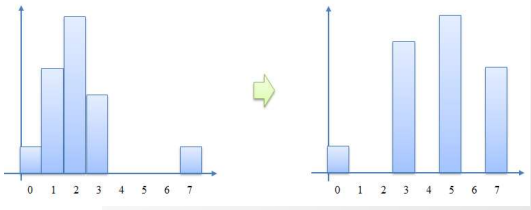
◆평활화 과정 예시

0	2	2	1
1	2	3	2
1	2	3	2
1	3	1	7

입력 영상 화소값

0	5	5	3
3	5	7	5
3	5	7	5
3	7	3	7

평활화 완료 영상 화소값



화소값	0	1	2	3	4	5	6	7
빈도수	1	5	6	3	0	0	0	1
누적 빈도수	1	6	12	15	15	15	15	16
정규화누적합	1/16	6/16	12/16	15//16	15//16	15/16	15/16	16/16
	0.0625	0.375	0.75	0.9375	0.9375	0.9375	0.9375	1
누적합 *최댓값	0.4375	2.625	5.25	6.5625	6.5625	6.5625	6.5625	7
평활화 결과	0	3	5	7	7	7	7	7

<그림 6.3.5> 평활화 계산 과정 예시

최대값이 7

히스토그램 평활화의 정규화 식은 오직 디지털화된 픽셀값을 정규화시킬 수 있다.
즉, 디지털화(양자화) 과정에서 잃어버린 값은 더이상 알 수 없으므로, 같은 디지털값으로 양자화된 픽셀값을 서로 다른 값으로 정규화시킬 수 없다.

35

6.3.5 히스토그램 평활화

최대값이 255

40	40	50	60	70
40	50	60	70	70
50	60	70	70	80
60	70	70	80	90
70	70	80	90	100

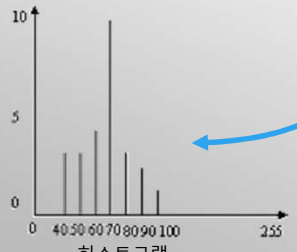
원영상

명암값	빈도수	누적 빈도수	정규화누적합 * 최댓값
40	3	3	30
50	3	6	61
60	4	10	102
70	9	19	193
80	3	22	224
90	2	24	244
100	1	25	255

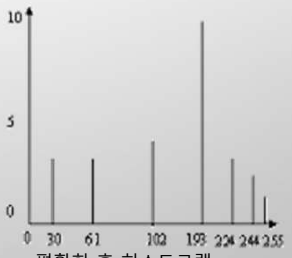
정규화

30	30	61	102	193
30	61	102	193	193
61	102	193	193	224
102	193	193	224	244
193	193	224	244	255

평활화된 영상



히스토그램



평활화 후 히스토그램

36

6.3.5 히스토그램 평활화

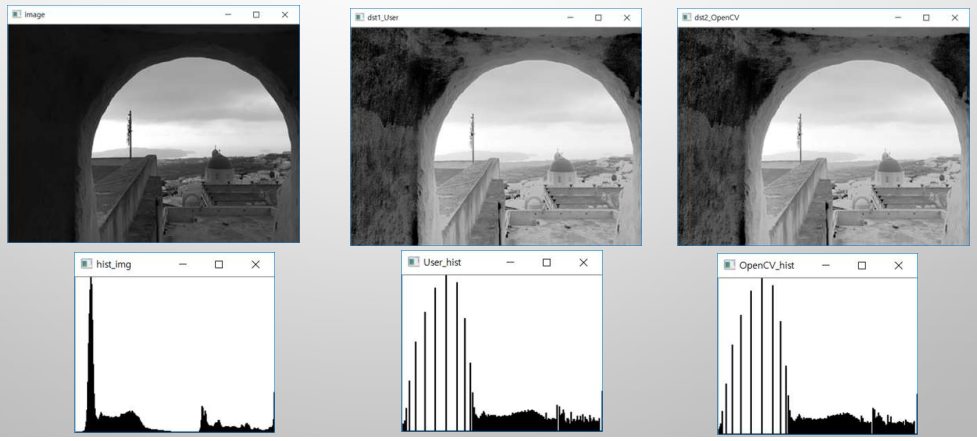
예제 6.3.6 히스토그램 평활화 – histogram_equalize.py

```
01  import numpy as np, cv2
02  from Common.histogram import draw_histo              # 히스토그램 그리기 함수 임포트
03
04  image = cv2.imread("images/equalize_test.jpg", cv2.IMREAD_GRAYSCALE) # 영상 읽기
05  if image is None: raise Exception("영상파일 읽기 오류")
06
07  bins, ranges = [256], [0, 256]
08  hist = cv2.calcHist([image], [0], None, bins, ranges)            # 히스토그램 계산
09
10  ## 히스토그램 누적합 계산
11  accum_hist = np.zeros(hist.shape[:2], np.float32)
12  accum_hist[0] = hist[0]
13  for i in range(1, hist.shape[0]):
14      accum_hist[i] = accum_hist[i - 1] + hist[i]
15
16  accum_hist = (accum_hist / sum(hist)) * 255                      # 누적합의 정규화
17  dst1 = [[accum_hist[val] for val in row] for row in image]       # 화소값 할당
18  dst1 = np.array(dst1, np.uint8)
19
20  ##numpy 함수 및 OpenCV 룩업 테이블 사용
21  # accum_hist = np.cumsum(hist)                                   # 누적합 계산
22  # cv2.normalize(accum_hist, accum_hist, 0, 255, cv2.NORM_MINMAX)  # 정규화
23  # dst1 = cv2.LUT(image, accum_hist.astype('uint8'))              # 룩업 테이블로 화소값 할당
24
```


6.3.5 히스토그램 평활화

```
25  dst2 = cv2.equalizeHist(image)                                 # OpenCV 히스토그램 평활화
26  hist1 = cv2.calcHist([dst1], [0], None, bins, ranges)          # 히스토그램 계산
27  hist2 = cv2.calcHist([dst2], [0], None, bins, ranges)
28  hist_img = draw_histo(hist)
29  hist_img1 = draw_histo(hist1)
30  hist_img2 = draw_histo(hist2)
31
32  cv2.imshow("image", image);          cv2.imshow("hist_img", hist_img)
33  cv2.imshow("dst1_User", dst1);       cv2.imshow("User_hist", hist_img1)
34  cv2.imshow("dst2_OpenCV", dst2);     cv2.imshow("OpenCV_hist", hist_img2)
35  cv2.waitKey(0)
```

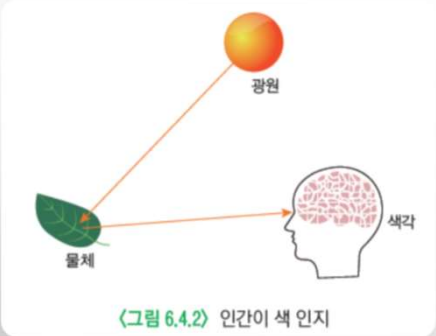
◆실행결과



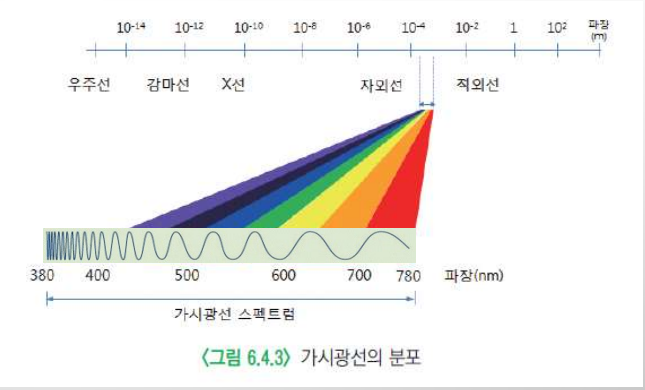
6.4 컬러 공간 변환
6.4.1 컬러 및 컬러 공간

◆색

❖ 색각으로 느낀 빛에서 주파수(파장)의 차이에 따라 다르게 느껴지는 색상



〈그림 6.4.2〉 인간이 색 인지



〈그림 6.4.3〉 가시광선의 분포

◆가시 광선

❖ 전체 전자기파 중에서 인간이 인지할 수 있는 약 380 nm~780 nm 사이의 파장
❖ 인간은 가시광선으로 색(Color)을 인식함.
❖ 파장의 길이에 따라 성질이 변화하여 각각의 색깔로 나타나는데,
빨강색에서 보라색으로 갈수록 파장이 짧아짐.

39

6.4.1 컬러 및 컬러 공간

◆컬러 공간(color space)

❖ 색 표시계(color system)의 모든 색들을 색 공간에서 3차원 좌표로 표현한 것
● 색 표시계 - RGB, CMY, HSV, LAB, YUV 등의 색 체계
❖ 공간상의 좌표로 표현 → 어떤 컬러와 다른 컬러들과의 관계를 표현하는 논리적인 방법 제공

◆영상처리에서 컬러 공간 다양하게 활용

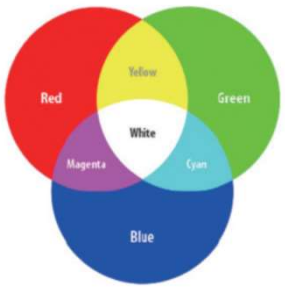
❖ 어떤 영상에서 각각의 색상이 다른 객체를 분리하기 위해서 적절한 컬러 공간을 이용해 전체 영상을 컬러 영역별 분리
❖ 전선의 연결 오류 검사를 위해 기준 색상과 비슷한 색상의 전선인지를 체크하기 위해 사용
❖ 내용 기반 영상 검색에서 색상 정보를 이용하여 원하는 물체를 검색하기 위해 특정 컬러 공간 이용

40

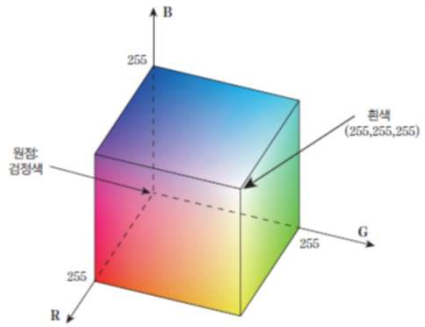
6.4.2 RGB 컬러 공간

◆RGB 컬러 공간

- ❖ 가산 혼합(additive mixture) – 빛을 섞을 수록 밝아짐
- ❖ 빛을 이용해서 색 생성
- ❖ 빛의 삼원색(빨강 빛, 초록 빛, 파랑 빛) 사용
- ❖ 모니터, 텔레비전, 빔 프로젝터와 같은 디스플레이 장비들에서 기본 컬러 공간으로 사용



〈그림 6.4.5〉 가산 혼합



〈그림 6.4.6〉 RGB 컬러 공간

41

6.4.2 RGB 컬러 공간

〈표 6.4.1〉 대표적인 색상에 대한 RGB 표현

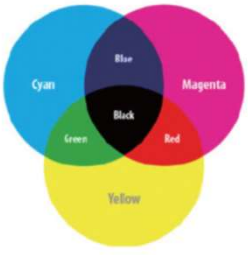
RGB 화소값	색상	RGB 화소값	색상
0, 0, 0	white	240, 230, 140	khaki
255, 255, 255	black	238, 130, 238	violet
128, 128, 128	gray	255, 165, 0	orange
192, 192, 192	silver	255, 215, 0	gold
255, 0, 0	red	0, 0, 128	navy
0, 255, 0	green	160, 32, 240	purple
0, 0, 255	blue	0, 128, 128	olive
255, 255, 0	yellow	75, 0, 130	indigo
255, 0, 255	magenta	255, 192, 203	pink
0, 255, 255	cyan	135, 206, 235	skyblue

42

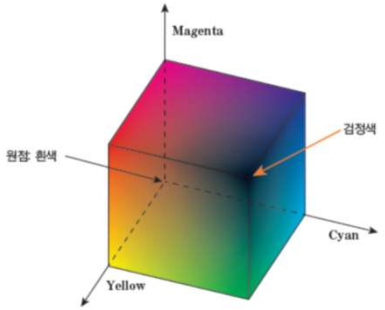
6.4.3 CMY(K) 컬러 공간

◆CMY 컬러 공간

- ❖ 감산 혼합(subtractive mixture) – 섞을 수록 어두워지는 방식
- ❖ 색의 삼원색(청록색(Cyan), 자홍색(Magenta), 노랑색(Yellow))으로 색 만듦
- ❖ RGB 컬러 공간과 보색 관계



〈그림 6.4.7〉 감산 혼합



〈그림 6.4.8〉 CMY 컬러 공간

C = 255 − R	R = 255 − C
M = 255 − G	G = 255 − M
Y = 255 − R	B = 255 − Y

43

6.4.3 CMY(K) 컬러 공간

◆CMYK 컬러공간

- ❖ 순수한 검정색 사용
  - ●뛰어난 대비를 제공, 검정 잉크가 컬러 잉크보다 비용이 적은 장점

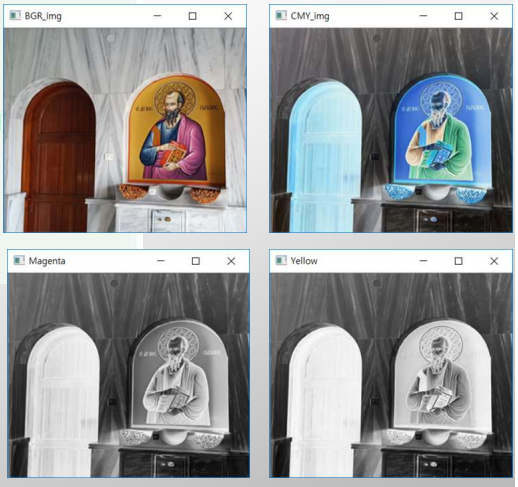
blacK	= min(Cyan, Magenta, Yellow)
Cyan	= Cyan - blacK
Magenta	= Magenta- blacK
Yellow	= Yellow - blacK

44

6.4.3 CMY(K) 컬러 공간

예제 6.4.1 컬러 공간 변환(BGR→CMY) – convert_CMY.py

```
01  import numpy as np, cv2
02
03  BGR_img = cv2.imread("images/color_model.jpg", cv2.IMREAD_COLOR) # 컬러 영상 읽기
04  if BGR_img is None: raise Exception("영상파일 읽기 오류")
05
06  white = np.array([255, 255, 255], np.uint8)
07  CMY_img = white - BGR_img
08  Yellow, Magenta, Cyan = cv2.split(CMY_img)                    # 채널 분리
09
10  titles = ['BGR_img', 'CMY_img', 'Yellow', 'Magenta', 'Cyan']
11  for t in titles: cv2.imshow(t, eval(t))
12  cv2.waitKey(0)
```



45

6.4.3 CMY(K) 컬러 공간

예제 6.4.2 컬러 공간 변환(BGR→CMYK) – 14.conver_CMYK.py

```
01  import numpy as np, cv2
02
03  BGR_img = cv2.imread("images/color_model.jpg", cv2.IMREAD_COLOR)      # 컬러 영상 읽기
04  if BGR_img is None: raise Exception("영상파일 읽기 오류")
05
06  white = np.array([255, 255, 255], np.uint8)
07  CMY_img = white - BGR_img
08  CMY = cv2.split(CMY_img)                                  # 채널 분리
09
10  black = cv2.min(CMY[0], cv2.min(CMY[1], CMY[2]))          # 원소 간의 최솟값 저장
11  Yellow, Magenta, Cyan = CMY - black                       # 3개 행렬 화소값 차분
12
13  titles = ['black', 'Yellow', 'Magenta', 'Cyan']
14  [cv2.imshow(t, eval(t)) for t in titles]                  # 리스트 생성 방식 활용
15  cv2.waitKey(0)
```

46

6.4.4 HSI 컬러 공간

◆인간이 컬러 영상 정보를 인지하는 방법

❖ 색상(Hue), 채도(Saturation), 명도(Intensity,Value)라는 세 가지 지각 변수로 분류

◆HSI 컬러 공간

❖ 인간이 색상을 인식하는 3가지 요인인 색상, 채도, 명도를 컬러 공간으로 옮긴 것

❖ 색상 - 원판의 0~360도까지 회전

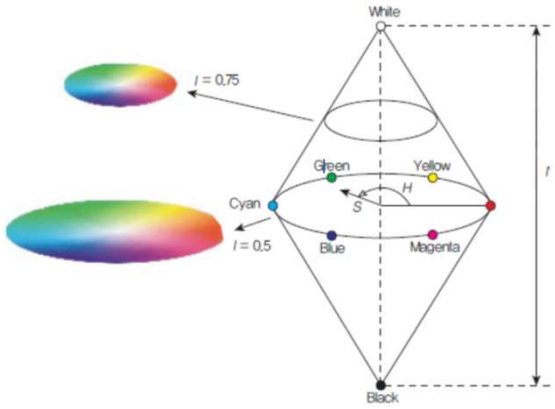
- ●0도: 빨간색, 60도: 노란색,
- ●120도: 녹색, 180도: 청록(Cyan),
- ●240도: 파란색, 300도: 다홍(Magenta)

❖ 채도 - 색의 순수한 정도

- ●흰색의 혼합 비율에 따라서 0~100까지의 값

❖ 명도 - 빛의 세기, 색의 밝고 어두운 정도

- ●가장 아래쪽 0이 검은색, 가장 위쪽이 100으로 흰색



〈그림 6.4.9〉 HSI 컬러 공간

47

6.4.4 HSI 컬러 공간

RGB → HSI 변환 수식

$$\theta = \cos^{-1}\left[\frac{((R-G)+(R-B))*0.5}{\sqrt{(R-G)^2+(R-B)\cdot(G-B)}}\right]$$

$$H = \begin{cases} \theta, & \text{if } B \le G \\ 360-\theta, & otherwise \end{cases}$$

$$S = 1 - \frac{3\cdot\min(R,G,B)}{(R+G+B)}$$

$$I = \frac{1}{3}(R+G+B)$$

OPENCV HSV 변환 수식

$$H = \begin{cases} \frac{(G-B)*60}{S}, & \text{if } V = R \\ \frac{(G-B)*60}{S}+120, & \text{if } V = G \\ \frac{(G-B)*60}{S}+240, & \text{if } V = B \end{cases}$$

$$S = \begin{cases} V - \frac{\min(R,G,B)}{V}, & \text{if } V \neq 0 \\ 0, & otherwise \end{cases}$$

$$V = \max(R,G,B)$$

48

6.4.4 HSI 컬러 공간

예제 6.4.3 컬러 공간 변환(BGR→HSV) – 15.conver_HSV.py

```
01  import numpy as np, cv2, math
02
03  def calc_hsi(bgr):                                  # 한 화소 hsi 계산 함수
04      # B, G, R = bgr.astype(float)                   # float 형 변환
05      B, G, R = float(bgr[0]), float(bgr[1]), float(bgr[2])   # 속도면에 유리
06      bgr_sum = (R + G + B)
07      ##    색상 계산
08      tmp1 = ((R - G) + (R - B)) * 0.5
09      tmp2 = math.sqrt((R - G) * (R - G) + (R - B) * (G - B))
10      angle = math.acos(tmp1 / tmp2) * (180 / np.pi) if tmp2 else 0      # 각도
11
12      H = angle if B <= G else 360 - angle                # 색상
13      S = 1.0 - 3 * min([R, G, B]) / bgr_sum if bgr_sum else 0  # 채도
14      I = bgr_sum / 3                                     # 명도
15      return (H/2, S*255, I)                              # 3 원소 튜플로 반환
16
17  ## BGR 컬러→HSI 컬러 변환 함수
18  def bgr2hsi(image):
19      hsv = [[calc_hsi(pixel) for pixel in row] for row in image]     # 2차원 배열 순회
20      return cv2.convertScaleAbs(np.array(hsv))
21
```

49

6.4.4 HSI 컬러 공간

```
22  BGR_img = cv2.imread("images/color_space.jpg", cv2.IMREAD_COLOR)
23  if BGR_img is None: raise Exception("영상파일 읽기 오류")
24
25  HSI_img = bgr2hsi(BGR_img)                          # BGR→ HSI 변환
26  HSV_img = cv2.cvtColor(BGR_img, cv2.COLOR_BGR2HSV)  # OpenCV 함수
27  Hue, Saturation, Intensity = cv2.split(HSI_img)     # 채널 분리
28  Hue2, Saturation2, Intensity2 = cv2.split(HSV_img)  # 채널 분리
29
30  titles = ['BGR_img', 'Hue', 'Saturation', 'Intensity']
31  [cv2.imshow(t, eval(t)) for t in titles]            # User 구현 결과 영상표시
32  [cv2.imshow('OpenCV_'+t, eval(t+'2')) for t in titles[1:]]   # OpenCV 결과 영상 표시
33  cv2.waitKey(0)
```

50

6.4.5 기타 컬러 공간

◆YCbCr 컬러 공간

❖ 색차 신호(Cr, Cb) 성분을 휘도(Y) 성분보다 상대적으로 낮은 해상도로 구성

- 인간의 시각에서 화질의 큰 저하 없이 영상 데이터의 용량 감소, 효과적인 영상 압축 가능
- JPEG이나 MPEG에서 압축을 위한 기본 컬러 공간

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$
$$Cb = (R - Y) \cdot 0.564 + 128$$
$$Cr = (B - Y) \cdot 0.713 + 128$$

$$R = Y + 1.403 \cdot (Cr - 128)$$
$$G = Y - 0.714 \cdot (Cr - 128) - 0.344(Cb - 128)$$
$$B = Y + 1.773 \cdot (Cb - 128)$$

◆YUV 컬러 공간

❖ TV 방송 규격에서 사용하는 컬러 표현 방식

❖ PAL 방식의 아날로그 비디오를 위해 개발

$$Y = +0.2160 \cdot R + 0.7152 \cdot G + 0.0722 \cdot B$$
$$U = -0.0999 \cdot R - 0.3360 \cdot G + 0.4360 \cdot B$$
$$V = +0.6150 \cdot R - 0.5586 \cdot G - 0.05639 \cdot B$$

$$R = Y + 1.28033 \cdot V$$
$$G = Y - 0.21482 \cdot U - 0.38059 \cdot V$$
$$B = Y + 2.12798 \cdot U$$

51

6.4.5 기타 컬러 공간

〈표 6.4.2〉 컬러 공간 변환을 위한 옵션 상수(cv2. 생략)

옵션 상수	값	옵션 상수	값	옵션 상수	값
COLOR_BGR2BGRA	0	COLOR_BGR2YCrCb	36	COLOR_HSV2BGR	54
COLOR_BGRA2BGR	1	COLOR_RGB2YCrCb	37	COLOR_HSV2RGB	55
COLOR_BGRA2RGB	2	COLOR_YCrCb2BGR	38	COLOR_LAB2BGR	56
COLOR_RGBA2BGR	3	COLOR_YCrCb2RGB	39	COLOR_LAB2RGB	57
COLOR_BGR2RGB ,	4	COLOR_BGR2HSV	40	COLOR_LUV2BGR	58
COLOR_BGRA2RGBA	5	COLOR_RGB2HSV	41	COLOR_LUV2RGB	59
COLOR_BGR2GRAY	6	COLOR_BGR2LAB	44	COLOR_HLS2BGR	60
COLOR_RGB2GRAY	7	COLOR_RGB2LAB	45	COLOR_HLS2RGB	61
COLOR_GRAY2BGR	8	COLOR_BayerBG2BGR	46	COLOR_BGR2YUV	82
COLOR_GRAY2BGRA	9	COLOR_BayerGB2BGR	47	COLOR_RGB2YUV	83
COLOR_BGRA2GRAY	10	COLOR_BayerRG2BGR	48	COLOR_YUV2BGR	84
COLOR_RGBA2GRAY	11	COLOR_BayerGR2BGR	49	COLOR_YUV2RGB	85
COLOR_BGR2XYZ	32	COLOR_BGR2LUV	50	COLOR_BayerBG2GRAY	86
COLOR_RGB2XYZ	33	COLOR_RGB2LUV	51	COLOR_BayerGB2GRAY	87
COLOR_XYZ2BGR	34	COLOR_BGR2HLS	52	COLOR_BayerRG2GRAY	88
COLOR_XYZ2RGB	35	COLOR_RGB2HLS	53	COLOR_BayerGR2GRAY	89

52

6.4.5 기타 컬러 공간

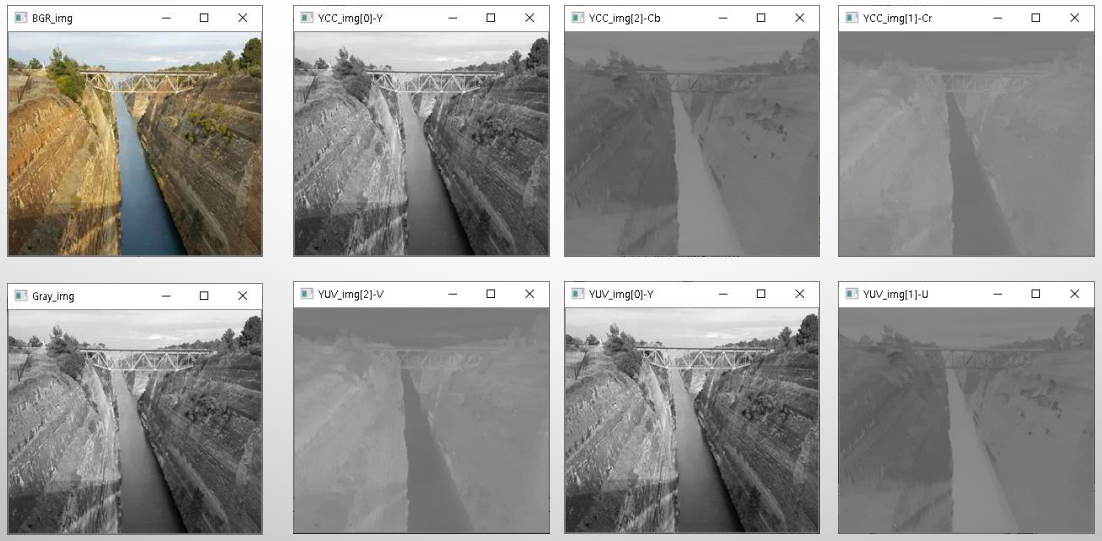
예제 6.4.4 다양한 컬러 공간 변환 – convert_others.py

```
01  import cv2
02
03  BGR_img = cv2.imread("images/color_space.jpg", cv2.IMREAD_COLOR) # 컬러 영상 읽기
04  if BGR_img is None: raise Exception("영상파일 읽기 오류")
05
06  Gray_img = cv2.cvtColor(BGR_img, cv2.COLOR_BGR2GRAY)          # 명암도 영상 변환
07  YCC_img = cv2.cvtColor(BGR_img, cv2.COLOR_BGR2YCrCb)          # YCbCr 컬러 공간 변환
08  YUV_img = cv2.cvtColor(BGR_img, cv2.COLOR_BGR2YUV)            # YUV 컬러 공간 변환
09  LAB_img = cv2.cvtColor(BGR_img, cv2.COLOR_BGR2LAB)            # La*b* 컬러 공간 변환
10
11  YCC_ch = cv2.split(YCC_img)                                   # 채널 분리
12  YUV_ch = cv2.split(YUV_img)
13  Lab_ch = cv2.split(LAB_img)
14
15  cv2.imshow("BGR_img", BGR_img)
16  cv2.imshow("Gray_img", Gray_img)
17
18  sp1, sp2, sp3 = ['Y', 'Cr', 'Cb'] , ['Y', 'U', 'V'], ['L', 'A', 'B']
19  for i in range(len(ch1)):
20      cv2.imshow("YCC_img[%d]-%s" %(i, sp1[i]), YCC_ch[i])
21      cv2.imshow("YUV_img[%d]-%s" %(i, sp2[i]), YUV_ch[i])
22      cv2.imshow("LAB_img[%d]-%s" %(i, sp3[i]), Lab_ch[i])
23  cv2.waitKey(0)
```

53

6.4.5 기타 컬러 공간

◆실행 결과



54

6.4.5 기타 컬러 공간

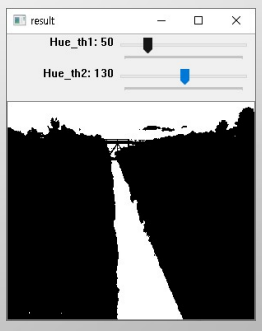
심화예제 6.4.5 Hue 채널을 이용한 객체 검출 – 17.hue_threshold.py

```
01  import numpy as np, cv2
02
03  def onThreshold(value):
04      th[0] = cv2.getTrackbarPos("Hue_th1", "result")
05      th[1] = cv2.getTrackbarPos("Hue_th2", "result")
06
07      ## 이진화– 화소 직접 접근 방법
08      # result = np.zeros(hue.shape, np.uint8)
09      # for i in range(result.shape[0]):
10      #     for j in range(result.shape[1]):
11      #         if th[0] <= hue[i, j] < th[1] : result[i, j] = 255
12
13      ## 이진화– 넘파이 함수 활용 방식
14      # result = np.logical_and(hue < th[1], hue >= th[0])
15      # result = result.astype('uint8') * 255
16
17      ## OpenCV 이진화 함수 이용– 상위 값과 하위 값 제거
18      _, result = cv2.threshold(hue, th[1], 255, cv2.THRESH_TOZERO_INV)
19      cv2.threshold(result, th[0], 255, cv2.THRESH_BINARY, result)
20      cv2.imshow("result", result)
21
```

55

6.4.5 기타 컬러 공간

```
22  BGR_img = cv2.imread("images/color_space.jpg", cv2.IMREAD_COLOR)      # 컬러 영상 읽기
23  if BGR_img is None: raise Exception("영상파일 읽기 오류")
24
25  HSV_img = cv2.cvtColor(BGR_img, cv2.COLOR_BGR2HSV)      # 컬러 공간 변환
26  hue = np.copy(HSV_img[:, :, 0])                         # hue 행렬에 색상 채널 복사
27
28  th = [50, 100]                                          # 트랙바로 선택할 범위 변수
29  cv2.namedWindow("result")
30  cv2.createTrackbar("Hue_th1", "result", th[0], 255, onThreshold)
31  cv2.createTrackbar("Hue_th2", "result", th[1], 255, onThreshold)
32  onThreshold(th[0])                                      # 이진화 수행
33  cv2.imshow("BGR_img", BGR_img)
34  cv2.waitKey(0)
```



56

단원 핵심 요약

◆디지털 영상은 화소들로 구성되며, 하나의 화소값은 0~255의 값을 가진다. 화소값 0은 검은색을, 255는 흰색을 의미한다. 그 사이의 값들은 진한 회색에서 연한 회색까지를 나타낸다. 화소값이 회색의 비율 정도로 표현되고, 이 값을 가지는 화소들이 모여서 구성된 영상을 그레이 스케일 영상이라 한다.

◆2. 행렬(ndarray 객체)의 모든 원소에 스칼라 값을 더하면 영상의 밝기를 밝게 하며, 스칼라 값을 빼면 영상 밝기를 어둡게 한다. 또한, 행렬에 스칼라 값을 곱하면 영상의 대비를 조절할 수 있다.

◆3. 히스토그램은 어떤 데이터가 얼마나 많은지를 나타내는 도수 분포표를 그래프로 나타낸 것이다. 가로축이 계급, 세로축이 도수(빈도수)를 뜻한다.

◆4. 히스토그램을 계산하는 cv2.calcHist() 함수는 인수로 입력영상 행렬들, 입력영상 개수, 채널 목록, 마스크 행렬, 출력 결과행렬, 결과행렬 차원수, 계급 크기, 각채널 범위 등으로 구성된다.

57

단원 핵심 요약

◆5. 히스토그램의 분포가 한쪽으로 치우쳐서 분포가 좁아서 영상의 대비가 좋지 않은 영상의 화질을 개선할 수 있는 알고리즘이 히스토그램 스트레칭(histogram stretching)이다.

◆6. 히스토그램 평활화(histogram e qualization)는 특정 부 분에서 한쪽으로 치우친 명암 분포를 가진 영상을 히스토그램의 재분배 과정을 거쳐서 균등한 히스토그램 분포를 갖게 하는 알고리즘이다.

◆7. 컬러 공간이란 색 표시계의 모든 색들을 색 공간에서 3차원 좌표로 표현한 것이다. 따라서 컬러 공간은 공간상의 좌표로 표현되기 때문에 어떤 컬러와 다른 컬러들과의 관계를 표현하는 논리적인 방법을 제공한다.

◆8.컬러 공간에서는 모니터에서 주로 사용하는 RGB, 프린터에서 사용하는 CMY, 인간시각 시스템과 유사한 HSI 컬러공간이 있다. 또한 JPEG 등의 압축에 주로 사용하는 YCbCr, 방송시스템에 많이 사용하는 YUV 컬러공간이 있다. 이외에도 XYZ, La*b* 등의 다양한 컬러 공간이 이며, OpenCV에서는 cv2.cvtColor() 함수를 이용해서 쉽게 컬러 공간을 변환할 수 있다.

58