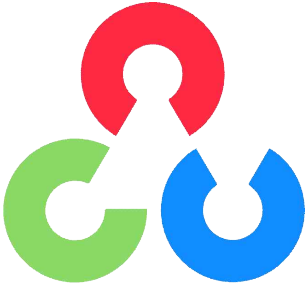
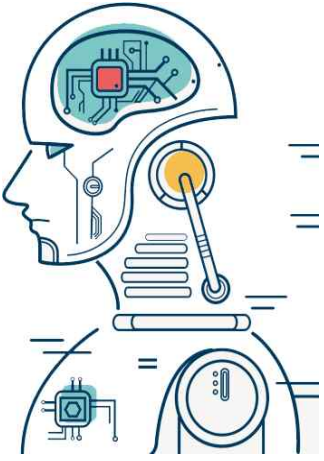


OpenCV

양재원

OpenCV

지난 강의 리뷰



도형 그리기

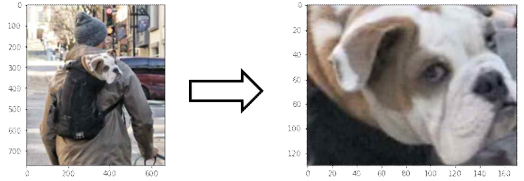
- 직선, 사각형, 다각형, 원 , 타원, 글씨 쓰기

창 관리 및 이벤트 처리

- 키보드, 마우스 및 트랙바 이벤트 처리

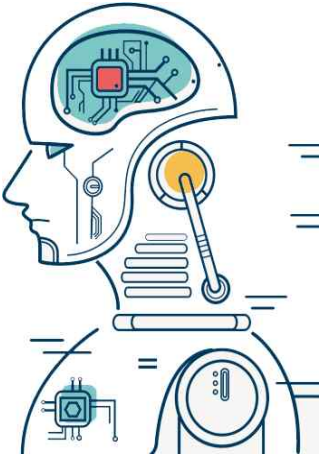
관심 영역 설정하기

- 마우스 드래그로 표시하기, Numpy 슬라이싱



OpenCV

컬러 스페이스



컬러 스페이스

- 색상을 표현하는 방법: RGB(Red, Green, Blue)
- OpenCV에선 RGB 대신 BGR로 표기
- 이미지는 3차원 배열로 (row x column x channel)로 표현

컬러 스페이스

- RGBA: A(알파, alpha)가 추가된 색상 표기법

  Alpha: 0~255 값을 가지며 투명도를 나타내는 값 (255: White, 0: Black)

- cv2.imread() 함수 두 번째 param(flag)에 cv2.IMREAD_COLOR 혹은 cv2.IMREAD_UNCHANGED를 통해 RGB, BGR, BGRA를 읽어 들이기

  알파 채널: 마스크 채널(mask channel)

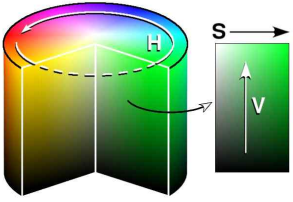
컬러 스페이스

- HSV 방식: RGB와 마찬가지로 3개의 채널을 갖는 색상 표현법

  Hue(색조), Saturation(채도), Value(명도)

- BRG → HSV로 변환하는 방법

  함수 cv2.cvtColor() 함수 두 번째 param(flag)에 cv2.COLOR_BRG2HSV



컬러 스페이스

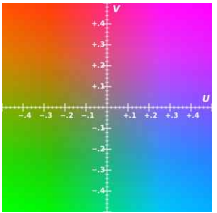
- YUV(YCbCr) 방식: RGB와 마찬가지로 3개의 채널을 갖는 색상 표현법

  Luma(밝기), 밝기와 파란색과의 색상 차(Chroma Blue, Cb),

  밝기와 빨간색과의 색상 차(Chroma Red,Cr)

- BRG → YUV로 변환하는 방법

  함수 cv2.cvtColor() 함수 두 번째 param(flag)에 cv2.COLOR_BRG2YUV



Y=0.5일 때

컬러 스페이스

- 색상을 알아내기 위해 BGR로 아는 것보다 HSV의 H만 알면 되어 효과적
- 밝기에 신경을 써야 하는 경우 BGR보다 YUV 방식을 선택
- OpenCV에선 BGR, BGRA, HSV, YUV 4가지 표현 방식

회색조(Grayscale)

- 한 개의 픽셀을 0~255의 값으로 표현

  255:White, 0: Black

- 연산의 양을 줄이기 위해 사용

회색조 이미지 변환

- 처음부터 회색조로 읽어 들이는 방법

  함수 cv2.imread() 함수 두 번째 param(flag)에 cv2.IMREAD_GRAYSCALE

- BGR로 컬러 이미지로 읽어 들인 후 회색조로 변경하는 법

  1) BGR 값들 평균 값 연산 후 적용

  → unit16으로 변환 후 연산 후 적용 시 다시 unit8로 적용

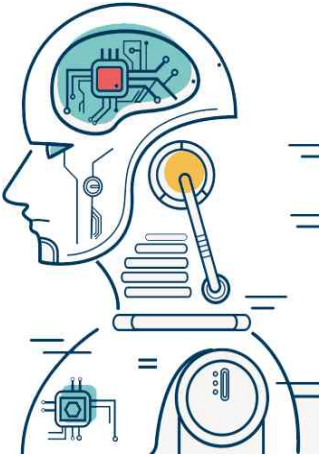
  2) cv2.cvtcolor() 함수 활용 (convert color)

  3) cv2.COLOR_BRG2GRAY를 cv2.cvtcolor(img, flag)의 flag 자리에 적용

cv2.COLOR_BGR2GRAY: BGR 색상 이미지를 회색조 이미지로 변환
cv2.COLOR_GRAY2BGR: 회색조 이미지를 BGR 색상 이미지로 변환
cv2.COLOR_BGR2RGB: BGR 색상 이미지를 RGB 색상 이미지로 변환
cv2.COLOR_BGR2HSV: BGR 색상 이미지를 HSV 색상 이미지로 변환
cv2.COLOR_HSV2BGR: HSV 색상 이미지를 BGR 색상 이미지로 변환
cv2.COLOR_BGR2YUV: BGR 색상 이미지를 YUV 색상 이미지로 변환
cv2.COLOR_YUV2BGR: YUB 색상 이미지를 BGR 색상 이미지로 변환

OpenCV

스레시홀딩



스레시홀딩 (thresholding)

- 바이너리 이미지를 만드는 대표적인 방법

  binary image: 검은색과 흰색만으로 표현한 이미지

- 스레시홀딩: 여러 값을 어떤 임계점을 기준으로 두 가지로 분류하는 방법

스레시홀딩 (thresholding)

- 전역 스레시홀딩

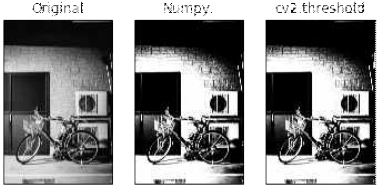
  픽셀 값이 어떤 임계 값을 정한 후 값을 넘으면 255, 넘지 않으면 0으로 지정하는 방식

- cv2.threshold(img, threshold, value, type_flag)

  threshold: 스레시홀딩 임계 값

  value: 임계값 기준에 만족하는 픽셀에 적용할 값

  type_flag: 스레시홀딩 적용 방법



스레시홀딩 (thresholding)

- 전역 스레시홀딩

  픽셀 값이 어떤 임계 값을 정한 후 값을 넘으면 255, 넘지 않으면 0으로 지정하는 방식

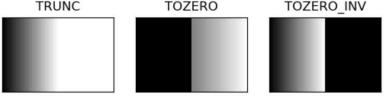
- cv2.threshold(img, threshold, value, type_flag)

  threshold: 스레시홀딩 임계 값

  value: 임계값 기준에 만족하는 픽셀에 적용할 값

  type_flag: 스레시홀딩 적용 방법

cv2.THRESH_BINARY: 픽셀 값이 임계 값을 넘으면 value로 지정하고, 넘지 못하면 0으로 지정
cv2.THRESH_BINARY_INV: cv2.THRESH_BINARY의 반대
cv2.THRESH_TRUNC: 픽셀 값이 임계 값을 넘으면 value로 지정하고, 넘지 못하면 원래 값 유지
cv2.THRESH_TOZERO: 픽셀 값이 임계 값을 넘으면 원래 값 유지, 넘지 못하면 0으로 지정
cv2.THRESH_TOZERO_INV: cv2.THRESH_TOZERO의 반대



오츠의 이진화 알고리즘 (Otsu's Method)

- 이진화 했을 때 흑과 백이 균일할수록 좋다
- 균일성은 분산으로 측정하며 분산이 작을수록 균일성이 높다
- 임계 값(T)보다 크거나 같으면 White, 작으면 Black

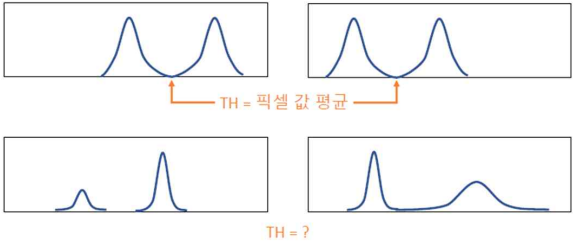
$$b(j,i) = \begin{cases} 1, & f(j,i) \geq T \\ 0, & f(j,i) < T \end{cases}$$

- 이진화에 따른 분류 에러를 최소화 시켜주는 임계 값: Optimal Threshold

  이진 분류된 픽셀의 비율의 차이가 가장 작은 Optimal T를 구하는 것

오츠의 이진화 알고리즘 (Otsu's Method)

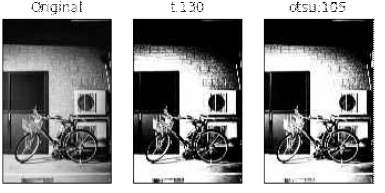
- 처음 분포 처럼 비슷하면 쉽게 구하지만, 아래 분포 처럼 차이가 클 때 유용



- 입력 이미지가 배경(Background)와 객체(Object) 두 개로 구성되어 있을 때 배경과 객체를 명확하게 구분하기 좋도록 하는 값

오츠의 이진화 알고리즘 (Otsu's Method)

- 임계 값을 임의로 정해 픽셀을 두 부류로 나누고
  두 부류의 명암 분포를 구하는 작업 반복
- 명암 분포가 가장 균일할 때의 임계 값 선택하는 알고리즘



스레시홀딩 (thresholding)

- 적응형 스레시홀딩(Adaptive Thresholding)

  원본 이미지의 조명이 일정하지 않거나 배경이 여러 색인 경우

  여러 영역으로 나눈 뒤, 주변 픽셀 값만 활용한 임계값 구하기

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

  value: 임계값 기준에 만족하는 픽셀에 적용할 값

  method: 임계값 결정 방법

  type_flag: 스레시홀딩 적용 방법

  block_size: 영역으로 나눌 이웃의 크기 (n x n), 홀수

  C: 계산된 임계값 결과에서 가감할 상수 (음수도 가능)

스레시홀딩 (thresholding)

- 적응형 스레시홀딩(Adaptive Thresholding)

  원본 이미지의 조명이 일정하지 않거나 배경이 여러 색인 경우

- Otsu 알고리즘 활용 시 어두운 부분이 구분이 어려운 문제

  → 전역 스레시홀딩의 문제

- 9등분 하여 9개의 블록별 임계 값을 정한 결과

  ADAPTIVE_THRESH_MEAN_C

  ADAPTIVE_THRESH_GAUSSIAN_C

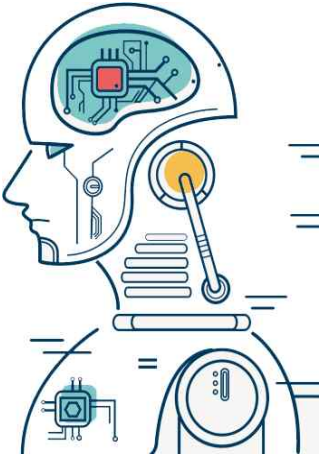
  → 평균 / Gaussian 분포에 따른 가중치의 합을 임계값

- Mean vs. Gaussian

  Mean이 선명도가 높지만 잡티가 더 나타남

OpenCV

이미지 연산



픽셀 연산

- Numpy Array 끼리 연산: pixel의 각 값들을 연산

  → 한 픽셀의 값의 범위는 0~255 사이지만 255 초과, 0 미만 값의 등장 가능성

- OpenCV의 사칙연산 함수: src1 * src2

  cv2.add(src1, src2, dest, mask, dtype)

  cv2.subtract(src1, src2, dest, mask, dtype)

  cv2.multiply(src1, src2, dest, scale, dtype)

  cv2.divide(src1, src2, dest, scale, dtype)

  dest: 출력 영상, mask: 0이 아닌 픽셀만 연산, scale: 연산 결과에 추가 연산 할 값

픽셀 연산

- 같은 연산을 해도 함수 사용여부에 따라 값이 달라짐

연산 예시1) A = np.unit8([200,50]), B = np.unit8([100,100]), (A + B)

numpy 연산 → [44, 150] : 연산하여 255가 높을 경우, 300-255-1 = 44

cv2.add() → [255,150] : 연산하여 255보다 높을 경우, 255

연산 예시2) A = np.unit8([200,50]), B = np.unit8([100,100]), (A - B)

numpy 연산 → [100, 206] : 연산하여 0보다 낮을 경우, -50+255+1 = 206

cv2.add() → [100,0] : 연산하여 0보다 낮을 경우, 0

픽셀 연산

- 함수의 dest, mask parameter

```
mask = np.array([[1, 0]], dtype = np.unit8)
cv2.add(A, B, None, mask)
cv2.add(A, B, B.copy(), mask)
cv2.add(A, B, B, mask)
```

이미지 합성

- numpy나 cv2.add() 함수를 활용

제대로 된 합성이 되지 않는다 → 대부분의 픽셀 값이 255 또는 0의 값을 갖는다

→ 두 픽셀의 값들을 단순 연산이 아닌 적절하게 분배한 합성 필요

이미지 합성

- 알파 블렌딩: 두 픽셀의 값들을 단순 연산이 아닌 적절하게 분배

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

가중치 α를 통한 조정

- cv2.addWeight(img1, alpha, img2, beta, gamma)

alpha: img1에 지정할 가중치

beta: img2에 지정할 가중치, 흔히 (1-alpha)

gamma: 연산 결과에 사용되는 상수, 흔히 0

비트와이즈(bitwise) 연산

- 두 이미지의 비트 단위 연산
- 두 이미지 합성 시, 특정 영역 선택/제외 등 선별적 연산

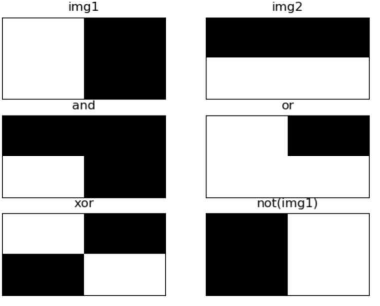
비트 연산자	설명
&	대응되는 비트가 모두 1이면 1을 반환함. (비트 AND 연산)
|	대응되는 비트 중에서 하나라도 1이면 1을 반환함. (비트 OR 연산)
^	대응되는 비트가 서로 다르면 1을 반환함. (비트 XOR 연산)
~	비트를 1이면 0으로, 0이면 1로 반전시킴. (비트 NOT 연산)

```
cv2.bitwise_and(img1, img2, mask=None)

cv2.bitwise_or(img1, img2, mask=None)

cv2.bitwise_xor(img1, img2, mask=None)

cv2.bitwise_not(img1, img2, mask=None)
```

비트와이즈(bitwise) 연산

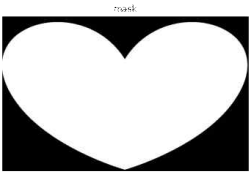
- 두 이미지의 비트 단위 연산
- 두 이미지 합성 시, 특정 영역 선택/제외 등 선별적 연산

비트 연산자	설명
&	대응되는 비트가 모두 1이면 1을 반환함. (비트 AND 연산)
|	대응되는 비트 중에서 하나라도 1이면 1을 반환함. (비트 OR 연산)
^	대응되는 비트가 서로 다르면 1을 반환함. (비트 XOR 연산)
~	비트를 1이면 0으로, 0이면 1로 반전시킴. (비트 NOT 연산)



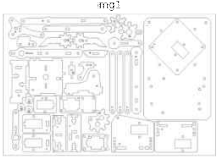
비트와이즈(bitwise) 연산 활용

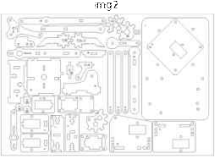
- 특정 모양의 부분만 출력
- 특정 모양의 부분 마스킹

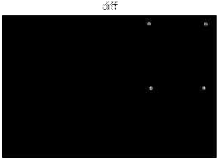


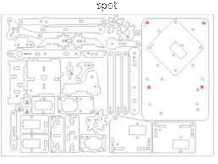
두 이미지의 차이

- 두 이미지를 빼면 음수가 나올 수 있음 → 절대값(absolute value(abs)) 활용
- cv2.absdiff(img1, img2)









이미지 합성과 마스킹

- 두 이미지를 합성 → 배경 + 전경

  전경의 원하는 객체의 부분만 추출? → Object Recognition!

- 한 곳에 함께 출력 할 경우 전경의 배경이 그대로 남아 어색한 합성

- 임의로 BGRA의 alpha를 통해 배경이 투명한 이미지 합성 실습

이미지 합성과 마스킹

- 임의로 BGRA의 alpha를 통해 배경이 투명한 이미지 합성 실습
- 배경이 투명 사진을 합성하기 위해 OpenCV의 mask 활용
  실습을 위해 배경의 alpha가 0인 이미지를 활용

- 마스킹: alpha 값을 기준으로 배경을 분리한 틀을 사용
- 마스킹한 전경의 이미지를 각각 전경과 배경 이미지에 적용 후
  두 이미지를 합성

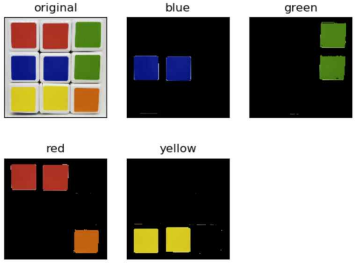
이미지 색상 별 추출

- 색상의 값 범위를 지정 후 범위에 해당되는 픽셀의 값에 따라 색상 필터

  색상에 따른 분류를 하기에 RGB보다 HSV의 컬러 스페이스 활용

- cv2.inRange(hsv, lower, upper)

  lower와 upper 사이 모든 값은 255 나머지는 0으로



크로마키(chroma key)

- alpha값에 따른 투명 배경이 아닌 특정 색상 배경을 지닌 사진의 합성
- blending, masking을 활용한 합성



크로마키(chroma key)

- blending의 alpha 값, masking의 좌표, 색상 선택 등 부가적인 작업 필요
- cv2.seamlessClone(src, dst, mask, coords, flags, output)

  mask: src에서 합성하고자 하는 영역은 255, 나머지는 0

  coords: src가 놓이기 원하는 dst에서의 좌표 (중앙)

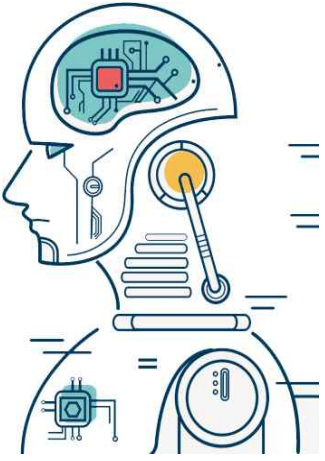
  flags: 합성 방식

  → cv2.NORMAL_CLONE: 입력 원본을 유지하는 방법 → src가 선명

  → cv2.MIXED_CLONE: 입력과 대상을 혼합하는 방법 → dst에 조화

OpenCV

강의 리뷰



컬러 스페이스

- BGR, BGRA, HSV, LUV, Gray Scale

스레시 홀딩

- 전역 스레시홀딩

  임계점 기준 두 가지로 나누는 방법

- 오츠 알고리즘

  최적의 임계점을 찾는 알고리즘

- 적응형 스레시홀딩

  이미지를 영역으로 나눠 주변 픽셀을 활용한 영역 별 임계값

이미지 연산

- 이미지 사이 단순 연산
- 알파 블렌딩

  합성하는 이미지 사이의 Weight를 부여한 합성
- 비트와이즈 연산

  특정 영역만 선택 혹은 제외하는 선별적 연산
- 이미지 차연산
- 이미지 합성과 마스킹

  블렌딩과 마스킹을 통한 이미지 사이의 합성

OpenCV

실습 과제

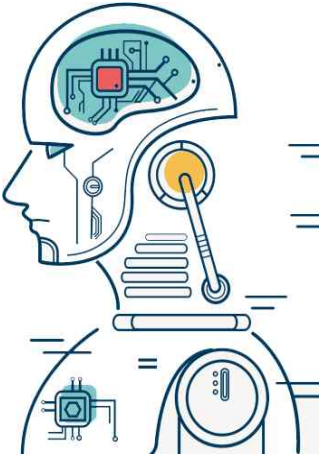


사진 합성 실습

- 사람 얼굴과 해골 사진 합성하기





- 경계선이 뚜렷하며 어색 → Alpha Blending

이미지 합성

- 알파 블렌딩: 두 픽셀의 값들을 단순 연산이 아닌 적절하게 분배

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

가중치 α를 통한 조정

- cv2.addWeight(img1, alpha, img2, beta, gamma)

alpha: img1에 지정할 가중치

beta: img2에 지정할 가중치, 흔히 (1-alpha)

gamma: 연산 결과에 사용되는 상수, 흔히 0