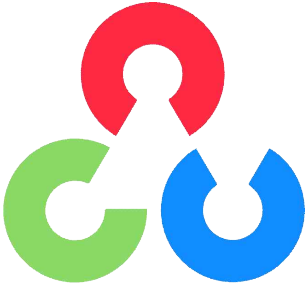
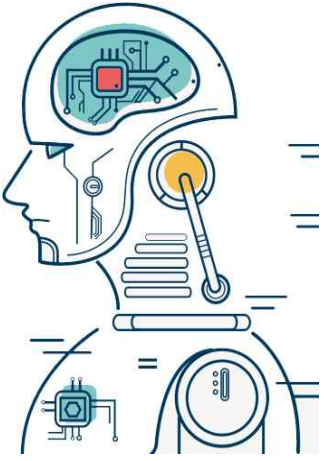


OpenCV

양재원

OpenCV

지난 강의 리뷰



필터, 컨볼루션 연산

- 평균 블러링, 가우시안 블러링, 미디언 블러링, 바이레터럴 필터

경계 검출

- 미분 필터
- 프리윗 필터
- 샤르 필터
- 캐니 엣지
- 로버츠 교차 필터
- 소벨 필터
- 라플라시안 필터

모폴로지

- 침식, 팽창
- 열림, 닫힘, 그레디언트

  열림 = 침식 + 팽창, 닫힘 = 팽창 + 침식, 그레디언트 = 팽창 - 침식
- 탑햇, 블랙햇

  탑햇 = 원본 - 열림, 블랙햇 = 닫힘 - 원본

이미지 피라미드

- 가우시안 피라미드
- 라플라시안 피라미드

이미지 분할

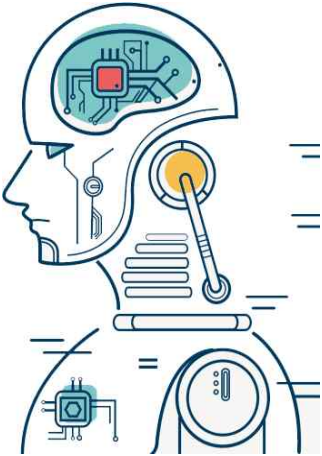
- 컨투어

블러링을 활용한 모자이크 처리

- 마우스로 드래그 하여 선택한 부분을 블러링 효과로 모자이크 처리
- 마우스 드래그 후 엔터 시 출력 결과

OpenCV

허프 변환



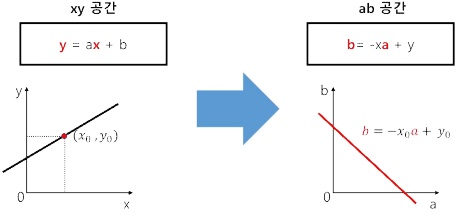
허프 변환(Hough Transform)

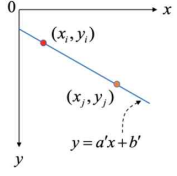
- 이미지에서 직선과 원등의 모양을 검출하는 알고리즘
- 이미지의 형태를 찾거나, 누락되거나 깨진 영역을 복원
- 하나의 점을 지나는 직선의 방정식을 활용하며 직교 좌표에서 극 좌표로 변환하여 표현 가능
- 극 좌표계에 각 점들을 지나는 직선들을 이어 만들어지는 선들의 교점을 통해 이미지 내에서 한 직선에 몇개의 점이 찍혀있는지 개수를 파악 가능

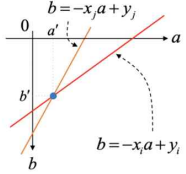
허프 선 변환

- 2차원의 x,y 좌표에서 직선의 방정식을 파라미터 공간으로 변환하여 직선 검출

$$y = a * x + b \quad \Rightarrow \quad b = -x * a + y$$

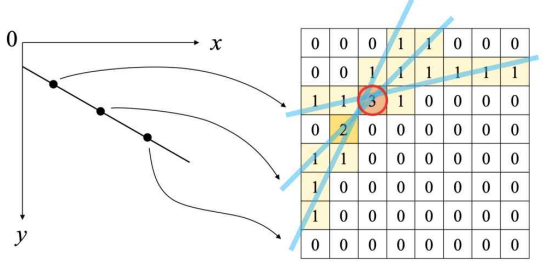






허프 선 변환

- xy 공간에서 이미지의 경계(직선) 픽셀들을 허프 변환을 통해 표현된 ab 공간 상의 직선들이 많이 교차되는 지점으로 경계(직선) 검출



- 축적 배열(accumulation array)

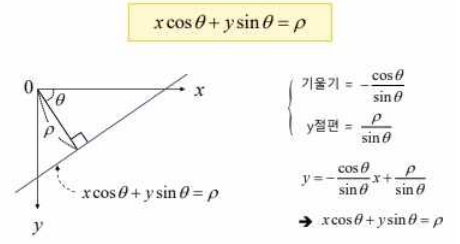
0으로 초기화된 2차원 배열에 직선이 지나가는 위치의 배열 원소 값을 1씩 증가

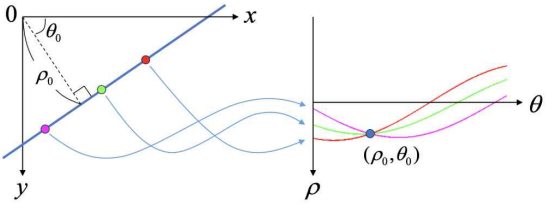
- y축과 평행한 수직선인 기울기가 무한인 직선은 ab 공간에서 표현이 불가능

허프 선 변환

- 직교 좌표의 직선을 극 좌표계 직선의 방정식으로 사용

$$y = ax + b \rightarrow \rho = x\cos\theta + y\sin\theta$$





허프 선 변환

- 픽셀 중 서로 직선 관계를 갖는 픽셀만 골라내는 것
- lines = cv2.HoughLines(img, rho, theta, threshold, lines, srn=0, stn=0

  min_theta, max_theta)

rho: 거리 측정 해상도 (0~1)

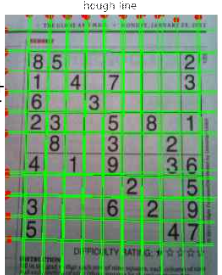
theta: 각도 (라디안 단위)

threshold: 직선으로 판단할 최소한의 동일 가 낮

lines: 검출 결과 (N x 1 x 2 배열)

srn, stn: 멀티 스케일 허프 변환 사용 (선 검

min_theta, max_theta: 검출 위해 사용할 최



확률적 허프 선 변환

- 수많은 선들을 이어보고 검출하기에 오래 걸리는 문제 개선
- 무작위로 선정한 픽셀에 대해 허프 변환을 수행하고 점차 그 수를 증가
- lines = cv2.HoughLinesP(img, rho, theta, threshold, lines,
  minLineLength, maxLineGap)

minLineLength: 선으로 인정할 최소한의 길이

maxLineGap: 선으로 판단한 최대 간격

허프 원 변환

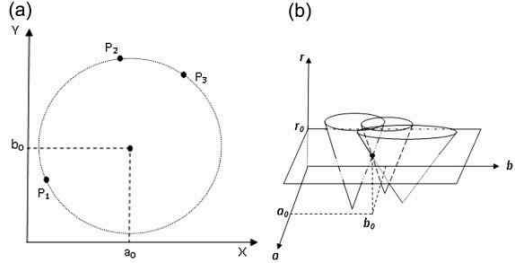
- 직선 검출은 xy 공간에서 경계로 판별된 점들을 p, theta 공간으로 변환하여 생성된 곡선들이 threshold 값 이상으로 교차되는 점들을 직선으로
- 원을 표현하는 방정식 → (a, b, r) 3차원 파라미터 공간으로 변환

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

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

허프 원 변환

- r에 상수 하나씩 대입하며 보면, xy 공간의 원위의 한점은 원뿔 모양으로 표현



- 원뿔들의 표면이 많이 교차하는 점을 찾아 원을 검출

허프 원 변환

- circles = cv2.HoughCircles(img, method, dp, minDist, circles, param1, param2, minRadius, maxRadius)

method: cv2.HOUGH_GRADIENT만 사용 가능

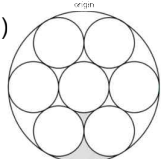
dp: 입력 영상과 경사 누적의 해상도 반비례율 (1: 입력과 동일, 값이 커질수록 부정확)

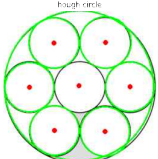
minDist: 원들 중심 간 최소거리 (동심원 검출 불가로 0: 에러)

circles: 검출 원 결과 (N x 1 x 3)

param1: 케니 엣지에 전달할 threshold 최대값 (최소는 최대의 절반 값을 전달)

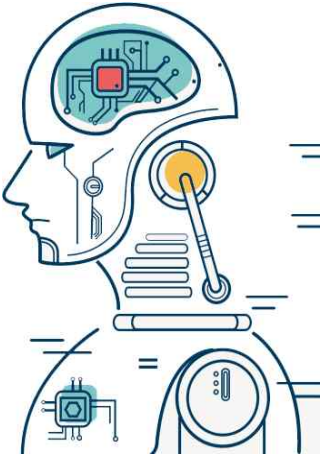
param2: 경사도 누적 경계 값 (값이 작을수록 잘못된 원 검출)





OpenCV

연속 영역 분할



거리 변환(Distance Transformation)

- 실제 이미지 상에는 노이즈도 많고 경계산도 뚜렷하지 않아 객체 분할에 어려움
- 연속된 영역을 찾아 분할하는 방법
- 물체 영역의 뼈대를 검출, 외곽 경계로부터 가장 멀리 떨어진 곳을 찾는 방법

  바이너리 스케일로 변환 후, 외곽 경계로부터 멀어질수록 흰색이 짙어지도록

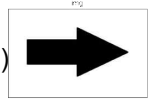
  → 픽셀 값 0으로부터 +1씩 더해지는 방법

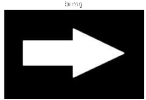
- cv2.distanceTransform(src, distanceType, maskSize)

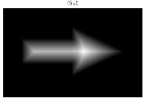
  distanceType: 거리 계산 방식

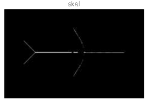
  maskSize: 거리 변환 커널 크기

cv2.DIST_L2
cv2.DIST_L1
cv2.DIST_L12
cv2.DIST_FAIR
cv2.DIST_WELSCH
cv2.DIST_HUBER

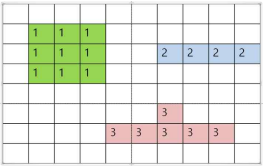








연결 요소 레이블링(Labeling)

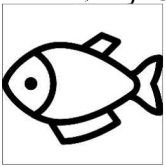


- 연결된 요소끼리 레이블링을 통한 분리
- 연결 요소 레이블링과 개수 반환

retval, labels = cv2.connectedComponents(src, labels,

connectivity=8, ltype)

connectivity: 연결성을 검사할 방향 개수 (4 or 8)



- 레이블링 된 각종 상태 반환 함수

retval, labels, stats, centroids =

cv2.connectedComponentsWithStats(src, labels, centroids,

connectivity, ltype

stats: N x 5 행렬(N: 레이블 수, 5: [x좌표, y좌표, 폭, 높이, 너비]

색 채우기

- 연속된 영역에 같은 색상을 채우는 기능
- retval, img, mask, rect = cv2.floodFill(img, mask, seed, newVal, loDiff, upDiff, flags)

mask: 입력 이미지보다 2 x 2 픽셀이 더 큰 배열 → 0이 아닌 영역을 만나면 채우기 중지

seed: 채우기 시작 좌표, newVal: 채울 색상 값

loDiff, upDiff: 채우기 진행을 결정 할 차이의 최소/최대 값

flags: 채우기 방식

cv2.FLOODFILL_MASK_ONLY: img가 아닌 mask에만 채우기 적용
cv2.FLOODFILL_FIXED_RANGE: 이웃 픽셀이 아닌 seed 픽셀과 비교

색 채우기

- retval, img, mask, rect = cv2.floodFill(img, mask, seed, newVal, loDiff, upDiff, flags)
- 이미지의 seed 좌표에서부터 newVal의 값으로 채우기 시작
- 이웃 픽셀에 채우기 진행 조건

  이웃 픽셀 – loDiff <= 현재 픽셀 <= 이웃 픽셀 + upDiff
- cv2.FLOODFIL_FIXED_RANGE: 이웃 픽셀이 아닌 seed 픽셀과 비교
- cv2.FLOODFILL_MASK_ONLY: 채우기가 아닌 mask에만 채우기

워터셰드(Watershed)

- 강물이 한 줄기로 흐르다 갈라지는 경계
- 색 채우기와 비슷하게 연속된 영역을 찾지만 seed가 여러 개 지정 (marker)

1) 0으로 채워진 marker 생성
2) 마우스가 움직이면 움직인 좌표에 해당되는 marker 좌표에 현재 marker 아이디 선언
3) 하나의 선 그리기를 끝내고 다음 marker를 위한 아이디 증가
4) 오른쪽 마우스 버튼을 누르면 워터셰드 실행
5) -1로 채워진 marker와 같은 좌표의 영상 픽셀을 초록색으로 변환 (경계선)
6) 미리 저장한 marker 아이디와 marker를 선택한 픽셀 값을 이용해 같은 marker 아이디 값을 갖는 영역을 같은 색으로 채움

워터셰드(Watershed)

- 강물이 한 줄기로 흐르다 갈라지는 경계
- markers = cv2.watershed(img, markers)

  markers: 입력 이미지와 크기가 같은 1차원 배열 (int32)

그랩컷(GrabCut)

- 객체로 분리할 부분에 사각형 표시
- 객체와 배경의 색상 분포 추정을 통해 동일 레이블을 갖는 연결된 영역의 객체 분리
- mask, bgdModel, fgdModel = cv2.grabCut(img, mask, rect, bgdModel, fgdModel, iterCount, mode)

mask: 입력 이미지와 크기가 같은 1채널 배열

bgdModel, fgdModel: 임시 배열 버퍼 (재사용 시 수정 X)

iterCount: 반복 횟수

mode: 동작 방법

mode
cv2.GC_INIT_WITH_RECT: rect에 지정한 좌표를 기준으로 그랩컷 수행
cv2.GC_INIT_WITH_MASK: mask에 지정한 값을 기준으로 그랩컷 수행
cv2.GC_EVAL: 재시도

mask
cv2.GC_BGD: 확실한 배경(0)
cv2.GC_FGD: 확실한 전경(1)
cv2.GC_PR_BGD: 아마도 배경(2)
cv2.GC_PR_FGD: 아마도 전경(3)

평균 이동 필터

- 객체를 물감으로 그린듯한 효과
- 이미지 피라미드를 만들어 작은 이미지의 평균이동 결과를 큰 이미지에 적용
- dst = cv2.pyrMeanShiftFiltering(src, sp, sr, dst, maxLevel, termcrit)

  sp: 공간 윈도 반지름 / sr: 색상 윈도 반지름

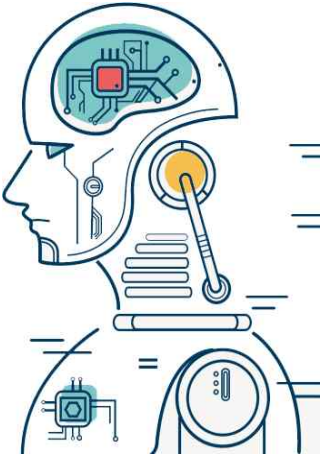
  maxLevel: 이미지 피라미드 최대 레벨

  termcrit: 반복 중지 요건

termcrit
cv2.TERM_CRITERIA_EPS: 정확도가 최소 정확도(epsilon) 보다 작아지면 중지
cv2.TERM_CRITERIA_MAX_ITER: 최대 반복 횟수(max_iter)에 도달하면 중지

OpenCV

이미지 매칭



이미지 매칭(Image Matching)

- 서로 다른 두 이미지를 비교하여 같은 객체인지 알아내는 것
- 여러 이미지의 객체를 숫자로 변환 후 비슷한 정도 판단 → 유사도 측정
- 특징을 대표할 수 있는 숫자: 특징 벡터

평균 해시 매칭(Average Hash Matching)

- 구현이 간단하고 효과는 떨어지는 방법
- 특징 벡터를 구하기 위해 평균값을 사용

1) 가로 세로 비율과 무관하게 특정 크기로 축소

2) 픽셀 전체의 평균값을 구해 각 픽셀의 값이 평균보다 작으면 0, 크면 1로 변환

3) 0 또는 1로 구성된 각 픽셀 값을 1행 1열로 변환

템플릿 매칭(Template Matching)

- 어떤 객체가 포함되어 있을 것이라 예상 될 이미지에서의 객체의 위치 찾기
- cv2.matchTemplate(img, templ, method, result, mask)

  templ: 템플릿 데이터

  result: 매칭 결과 2차원 배열

  method
  cv2.TM_SQDIFF : 제곱 차이 매칭, 완벽 매칭: 0, 나쁜 매칭: 큰 값
  cv2.TM_SQDIFF_NORMED : 제곱 차이 매칭의 정규화
  cv2.TM_CCORR : 상관관계 매칭, 완벽 매칭: 큰 값, 나쁜 매칭: 0
  cv2.TM_CCORR_NORMED : 상관관계 매칭의 정규화
  cv2.TM_CCOEFF : 상관계수 매칭, 완벽 매칭: 1, 나쁜 매칭: -1
  cv2.TM_CCOEFF_NORMED: 상관계수 매칭의 정규화

  mask: TM_SQDIFF, TM_CCORR_NORMED경우 사용할 마스크

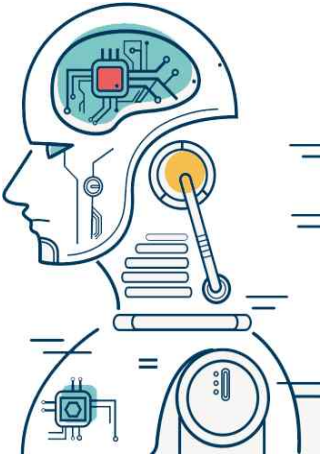
- minVal, maxVal, minLoc, maxLoc = cv2.minMaxLoc(src, mask)

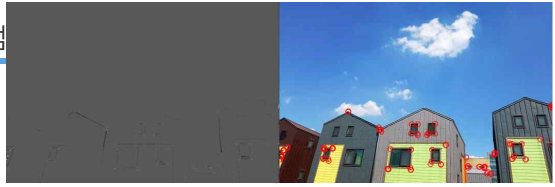
  minVal, maxVal: 배열 전체에서 최소/최대 값

  minLoc, maxLoc: 최소/최대 값의 좌표

OpenCV

이미지의 특징점과 검출기





이미지 특징점(Keypoints)

- 이미지에서 특징이 되는 부분
- 이미지끼리 서로 매칭이 되는지 확인 시 특징끼리 비교 → 대부분 코너(corner)

해리스 코너 검출(Harris Corner Detection)

- 소벨(Sobel) 미분으로 경계를 검출한 경사도 변화량이 크게 변하는 것을 코너로
- dst = cv2.cornerHarris(src, blockSize, ksize, k, dst, borderType)

  blockSize: 이웃 픽셀 범위

  ksize: 소벨 미분 필터 크기

  k: 코너 검출 상수, dst: 코너 검출 결과, borderType: 외곽 영역 보정 형식

시-토마시(Shi&Tomasi Detection)

- 코너점 등 직관에 의지한 특징점 찾는 법을 개선
- 평행이동만을 고려하지 않고 변환까지 고려한 특징점 선택
- cv2.goodFeaturesToTrack(img, maxCorners, qualityLevel, minDistance, corners, mask, blockSize, useHarrisDetector, k)

maxCorners: 얻고 싶은 코너 개수
qualityLevel: 코너로 판단할 임계값
minDistance: 코너 사이의 최소거리
mask: 검출 제외할 마스크
useHarrisDetector: 해리스 코너 사용할지 여부
corners:코너 검출 결과 (N x 1 x 2 크기)

특징점 검출기(Keypoints Detector)

- 코너 검출의 좌표 검출뿐 아니라 추가 정보 함께 반환
- keypoints = detector.detect(img, mask)
- cv2.Keypoint: 특징점 정보를 담는 객체

  pt: 특징점 좌표 (x, y) float 타입으로 정수 변환 필요

  size: 의미 있는 특징점 이웃의 반지름

  angle: 특징점 방향 (시계방향, -1=의미없음)

  response: 특징점 반응 강도(추출기에 따라 다름)

  octave: 발견된 이미지 피라미드 계층

  class_id:특징점이 속한 객체 ID

특징점 검출기(Keypoints Detector)

- OpenCV의 특징점 표시 전용 함수
- outImg = cv2.drawKeypoints(img, keypoints, outImg, color, flags)

  keypoints: 표시할 특징점 리스트

  outImg:특징점이 그려진 결과

  color: 표시할 색상 (default: random)

  flags:표시 방법

  cv2.DRAW_MATCHES_FLAGS_DEFAULT: 좌표 중심에 동그라미만 그림(default)

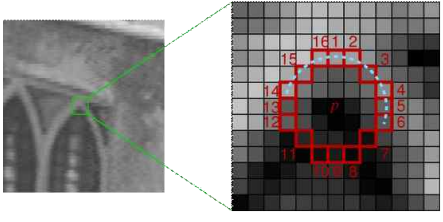
  cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS: 동그라미의 크기를 size와 angle을 반영해서 그림

GFTTDetector

- cv2.goodFeaturesToTrack() 과 같은 param들을 받아 사용
- detector = cv2.GFTTDetector_create(img, maxCorners, qualityLevel, minDistance, corners, mask, blockSize, useHarrisDetector, k)
- 해리스 코너 검출과 시-토마시 검출보다 많은 코너가 검출되며 다양한 특징을 지니는 코너까지 검출

FAST(Feature from Accelerated Segment Test)

- 기존 검출기보다 속도가 빠른 검출기 → 미분 연산 하지 않아 빠르다
- 픽셀 중심으로 특정 개수의 픽셀로 원을 그려 그 안의 중심 픽셀 값보다 임계값 이상 밝거나 어두운 것이 일정 개수 이상 연속되면 특징점으로로 판단
- 어떤 점 p가 특징점인지 확인 할 때, p 중심으로 원 상 16개 픽셀을 보고, 임계 값 이상보다 n개 이상 연속 밝거나 어두울 때 특징점으로 판단



FAST(Feature from Accelerated Segment Test)

- detector = cv2.FastFeatureDetector_create(threshold, nonmaxSuppresion, type)

threshold: 코너 판단 임계값 (default: 10)

nonmaxSuppresion: 최대 점수가 아닌 코너 억제

type: 엣지 검출 패턴 (default: cv2.FastFeatureDetector_TYPE_9_16: 16개 중 9개 연속, cv2.FastFeatureDetector_TYPE_7_12: 12개 중 7개 연속, cv2.FastFeatureDetector_TYPE_5_8: 8개 중 5개 연속)

SimpleBlobDetector

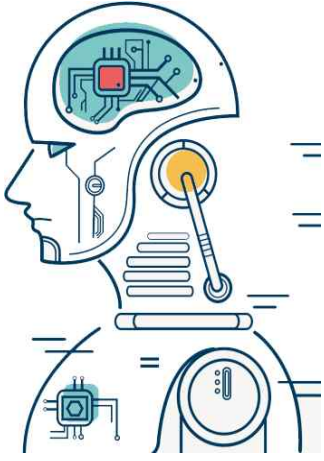
- BLOB(Binary Large Object): 이진 스케일로 연결된 픽셀 그룹
- 작은 객체는 노이즈로 판단하고 특정 크기 이상의 큰 객체만 관심 갖는 방법
- detector = cv2.SimpleBlobDetector_create([parameters])
- cv2.SimpleBlobDetector_Params()

minThreshold, maxThreshold, thresholdStep: BLOB를 생성하기 위한 경계 값
(minThreshold에서 maxThreshold를 넘지 않을 때까지 thresholdStep만큼 증가)
minRepeatability: BLOB에 참여하기 위한 연속된 경계 값의 개수
minDistBetweenBlobs: 두 BLOB을 하나의 BLOB으로 간주하는 거리
filterByArea: 면적 필터 옵션
minArea, maxArea: min~max 범위의 면적만 BLOB으로 검출
filterByCircularity: 원형 비율 필터 옵션

minCircularity, maxCircularity: min~max 범위의 원형 비율만 BLOB으로 검출
filterByColor: 밝기를 이용한 필터 옵션
blobColor: 0 = 검은색 BLOB 검출, 255 = 흰색 BLOB 검출
filterByConvexity: 볼록 비율 필터 옵션
minConvexity, maxConvexity: min~max 범위의 볼록 비율만 BLOB으로 검출
filterByInertia: 관성 비율 필터 옵션
minInertiaRatio, maxInertiaRatio: min~max 범위의 관성 비율만 BLOB으로 검출

OpenCV

특징 스크립터 검출기



특징 스크립터(Feature Descriptor)

- 특징점: 객체의 좌표와 주변 픽셀과의 관계에 대한 정보
  크기, 각도, 경사도, 방향 정보 포함

- 특징 디스크립터: 특징점 주변 픽셀을 일정한 크기와 블록으로 나눠
  각 블록에 속한 픽셀의 기울기(Gradient) 히스토그램을 계산
  주변의 밝기, 색상, 방향, 크기 등의 정보 포함

- 일반적으로 특징점 주변 블록 크기에 8방향 경사도를 표현

특징 스크립터(Feature Descriptor)

- keypoints, descriptor = detector.compute(image, keypoints, descriptors)

  특징점을 전달하면 특징 디스크립터를 계산하여 반환

- kepoints, desecriptors = detector.detectAndCompute(img, mask,

  descriptors, useProvidedKeypoints)

  keypoints: 디스크립터 계산을 위해 사용될 특징점

  mask:특징점 검출에 사용할 마스크

  useProvidedKeypoints: True일 경우 특징점 검출을 수행하지 않고 주어진 특징점 사용

- keypoint를 구하고 detector.compute()를 사용하는 대신 하나의 함수로 수행

SIFT(Scale-Invariant Feature Transform)

- 해리스 코너 검출의 크기 변화 민감 문제 해결
- 이미지 피라미드를 이용해 크기 변화에 따른 특징점 검출 문제 해결 알고리즘
- detector = cv2.xfeature2d.SIFT_create(nfeatures, nOctaveLayers, contrastThreshold, edgeThreshold, sigma)

nfeatures: 검출 최대 특징 수

nOctaveLayers: 이미지 피라미드에 사용할 계층 수

contrastThreshold: 필터링할 빈약한 특징 임계 값

edgeThreshold: 필터링할 엣지 임계 값

sigma: 이미지 피라미드의 0 계층에서 사용할 가우시안 필터의 시그마 값

```
keypoint: 413 descriptor: (413, 128
[[  1.   1.   1. ...   0.   0.   1.]
 [  8.  24.   0. ...   1.   0.   4.]
 [  0.   0.   0. ...   0.   0.   2.]
 ...
 [  1.   8.  71. ...  73. 127.   3.]
 [ 35.   2.   7. ...   0.   0.   9.]
 [ 36.  34.   3. ...   0.   0.   1.]]
```

ORB(Oriented and Rotated BRIEF)

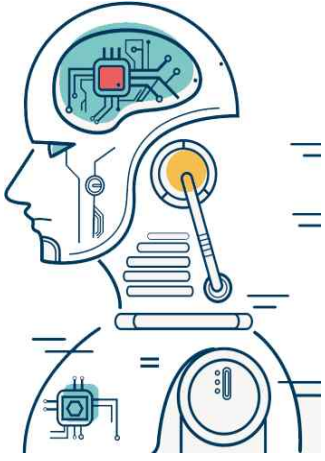
- 특징점 검출은 지원하지 않는 디스크립터 추출기인 BRIEF(Binary Robust Elementary Features)에 방향과 회전을 고려하도록 개선한 알고리즘
- 특징점 검출을 FAST를 통해 진행 후 회전과 방향을 고려
- detector = cv2.ORB_create(nfeatures, scaleFactor, nlevels, edgeThreshold, firstLevel, WTA_K, scoreType, patchSize, fastThreshold)

nfeatures(optional): 검출할 최대 특징 수 (default: 500)
scaleFactor(optional): 이미지 피라미드 비율 (default: 1.2)
nlevels(optional): 이미지 피라미드 계층 수 (default: 8)
edgeThreshold(optional): 검색에서 제외할 테두리 크기, patchSize와 맞출 것 (default: 31)
firstLevel(optional): 이미지 피라미드 첫 계층 단계 (default: 0)

WTA_K(optional): 임의
scoreType(optional):
(cv2.ORB_HARRIS_
cv2.ORB_FAST_SCO
patchSize(optional):
fastThreshold(optiona
(default=20)

OpenCV

특징 매칭



특징 매칭(Feature Matching)

- 두 이미지의 특징점과 특징 디스크립터를 비교해 객체를 연결
- matcher = cv2.DescriptorMatcher_create(matcherType)

matcherType: 생성할 디스크립터의 구현 알고리즘

BruteForce: NORM_L2를 사용하는 BFMatcher

BruteForce-L1: NORM_L1을 사용하는 BFMatcher

BruteForce-Hamming: NORM_HAMMING을 사용하는 BRMatcher

BruteForce-Hamming(2): NORM_HAMMING2를 사용하는 BFMatcher

FlannBased: NORM_L2를 사용하는 FlannBasedMatcher

해밍 거리(Hamming Distance): 두 부호어 사이의 거리 → 두 문자열에서 서로 다른 문자 쌍의 개수

ex) 010, 000: 한 위치의 값만 달라 거리 1 / 010, 111: 두 위치의 값이 달라 거리 2

→ 두 문자열(부호어)가 얼마나 다른지에 대한 정량화 값

특징 매칭(Feature Matching)

- 디스크립터 생성 후 두 디스크립터 비교 매칭 함수

  match(): 1개의 최적 매칭

  knnMatch(): k개의 가장 근접한 매칭

  radiusMatch(): 정해둔 반경의 비슷한 것을 다 매칭

- matches = matcher.match(queryDescriptors, trainDescriptors, mask)

  queryDescriptors: 매칭의 기준이 될 디스크립터

  trainDescriptors: 매칭의 대상이 될 디스크립터

  matches: 디스크립터 매치의 객체 리스트

특징 매칭(Feature Matching)

- matches = matcher.knnMatch(qD, tD, k, mask, compactResult)

  k: 매칭할 근접 이웃 개수

  compactResult: True일 경우 매칭이 없는 경우 매칭 결과에 불포함 (Default: False)

- matches = matcher.radiusMatch(qD, tD, maxDistance, mask, cR)

  maxDistance:매칭 대상 거리 (설정하는 반경)

- 세 함수의 반환 결과인 matches는 DMatch 객체 리스트

  queryIdx: queryDescriptor의 인덱스

  trainIdx: trainDescriptor의 인덱스

  imgIdx: trainDescriptor의 이미지 인덱스

  distance: 유사도 거리

특징 매칭(Feature Matching)

- 매칭 결과 시각적 표현하기 위한 두 이미지 매칭점끼리 선으로 연결
- cv2.drawMatches(img1, kp1, img2, kp2, matches, flags)

  img1, kp1 / img2, kp2: qD와 tD의 이미지와 특징점

  matches: 매칭 결과

  flags: 매칭점 그리기 옵션

  cv2.DRAW_MATCHES_FLAGS_DRAW_OVER_OUTIMG: 결과 이미지 새로 생성 안 함

  cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS: 특징점 크기와 방향도 그리기

  cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS: 한쪽만 있는 매칭 결과 그리기 제외)

BFMatcher(Brute-Force Matcher)

- qD와 tD를 하나하나 확인해 매칭되는지 판단하는 알고리즘
- matcher = cv.BFMatcher_create(normType, crossCheck)

  normType: 거리 측정 알고리즘

  crossCheck: 상호 매칭되는 것만 반영(default: False)

유클리디안 거리 측정 or 해밍거리 측정

cv2.NORM_L1

cv2.NORM_L2(default)

cv2.NORM_L2SQR

cv2.NORM_HAMMING

cv2.NORM_HAMMING2

cv2.NORM_L1: $\sum_i abs(a_i - b_i)$

cv2.NORM_L2: $\sum_i (a_i - b_i)^2$

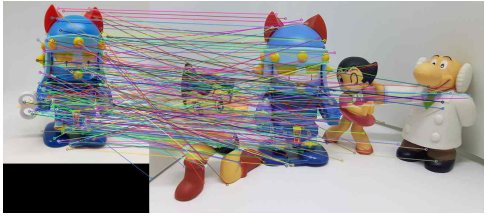
cv2.NORM_L2SQR: $\sqrt{\sum_i (a_i - b_i)^2}$

cv2.NORM_HAMMING: $\sum_i (a_i == b_i)\,?\,1:0$

cv2.NORM_HAMMING2: $\sum_i (a_i == b_i)\&\&(a_{i+1} == b_{i+1})\,?\,1:0$

FLANN(Fast Library for Approximate Nearest Neighbors Matching)

- BFMatcher는 모든 디스크립터를 전수 조사하여 속도가 느림
- FLANN은 이웃하는 디스크립터끼리 비교



FLANN(Fast Library for Approximate Nearest Neighbors Matching)

- matcher = cv2.FlannBasedMatcher(indexParams, searchParams)

indexParams: 인덱스 파라미터 (딕셔너리)

algorithm: 알고리즘 선택 키, 선택할 알고리즘에 따라 종속 키를 결정하면 됨

FLANN_INDEX_LINEAR=0: 선형 인덱싱, BFMatcher와 동일

FLANN_INDEX_KDTREE=1: KD-트리 인덱싱 (trees=4: 트리 개수(16을 권장))

FLANN_INDEX_KMEANS=2: K-평균 트리 인덱싱 (branching=32: 트리 분기 개수, iterations=11: 반복 횟수, centers_init=0: 초기 중심점 방식)

FLANN_INDEX_COMPOSITE=3: KD-트리, K-평균 혼합 인덱싱 (trees=4: 트리 개수, branching=32: 트리 분기 새수, iterations=11: 반복 횟수, centers_init=0: 초기 중심점 방식)

FLANN_INDEX_LSH=6: LSH 인덱싱 (table_number: 해시 테이블 수, key_size: 키 비트 크기, multi_probe_level: 인접 버킷 검색)

FLANN_INDEX_AUTOTUNED=255: 자동 인덱스 (target_precision=0.9: 검색 백분율, build_weight=0.01: 속도 우선순위, memory_weight=0.0: 메모리 우선순위, sample_fraction=0.1: 샘플 비율)

searchParams: 검색 파라미터 (딕셔너리)

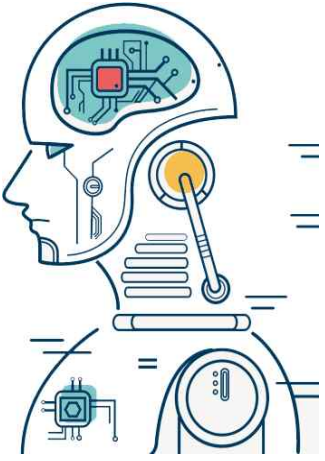
checks=32: 검색할 후보 수

eps=0.0: 사용 안 함

sorted=True: 정렬해서 반환

OpenCV

강의 리뷰



허프변환

- 허프 선 변환
- 허프 원 변환

연속 영역 분할

- 거리 변환
- 색 채우기
- 그랩컷
- 레이블링
- 워터셰드
- 평균 이동 필터

이미지 매칭

- 평균 해시 매칭
- 템플릿 매칭

특징점 찾기와 검출기

- 특징점 찾기

  해리스 코너 검출, 시-토마시 검출

- 특징점 검출기

  GFTT Detector, FAST, BLOB

특징 디스크립터 검출기

- SIFT
- ORB

특징 매칭

- BFMatcher
- FLANN

올바른 매칭 찾기

- 매칭 영역 원근 변환