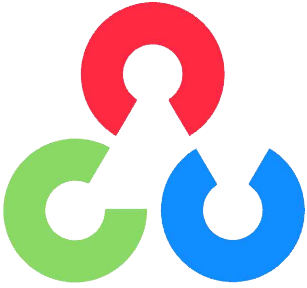
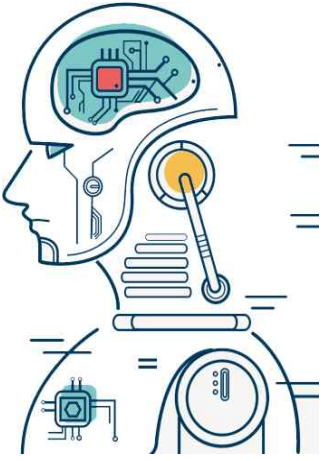


OpenCV

양재원

OpenCV

지난 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

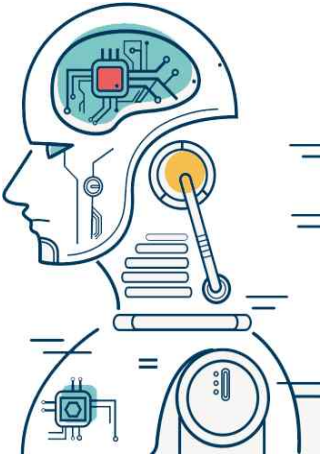
- SIFT
- ORB

특징 매칭

- BFMatcher
- FLANN

OpenCV

올바른 매칭점 찾기



특징 매칭(Feature Matching)

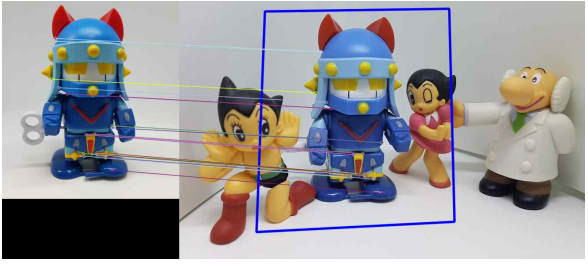
- 특징 매칭을 하는 과정에서 잘못된 매칭 결과가 많이 포함
- 매칭 결과에서 쓸모없는 매칭점은 버리고 옳은 매칭점만 골라내는 작업
- 올바른 매칭점의 개수가 적으면 서로 연관관계가 없다고 판단

올바른 매칭점 찾기

- radiusMatch()는 maxDistance 파라미터를 조절 통한 보완
- match(): 가장 작은 거리값과 큰 거리값의 상위 몇퍼센트만 고르기
- knnMatch(): 디스크립터 당 k개 중 가까운 것들 위주로 고르기

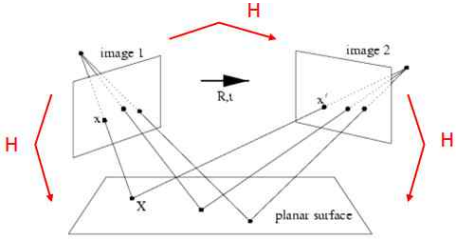
매칭 영역 원근 변환

- 비교될 이미지가 회전되어있거나 물체가 회전되어 있을때 활용
- 원근 변환 행렬을 구해 찾고자 하는 객체의 위치
- 원근 변환 행렬에 맞지 않는 매칭점을 구분하여 올바른 매칭점 찾기 효과
- 원근 변환 행렬 구하던 cv2.getPerspectiveTransform()과 비슷



매칭 영역 원근 변환

- 호모그래피(Homography): 두 평면 사이의 perspective transform
- H1: 1번 평면에서 바닥을 찍어 기울어진 이미지와 원본 이미지의 관계
- H2: 2번 평면에서 바닥을 찍어 기울어진 이미지와 원본 이미지의 관계
- H12: 기울어지게 찍힌 두 이미지 사이의 관계



$$s\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

매칭 영역 원근 변환

- mtrx, mask = cv2.findHomography(srcPoints, dstPoints, method,
  ransacReprojThreshold, mask, maxiters, confidence)

  srcPoints, dstPoints: 원본, 결과 좌표 배열

  method=0: 근사 계산 알고리즘 선택, 0: 모든 점으로 최소 제곱 오차 계산

  maxiters=2000: 근사 계산 반복 횟수

  confidence= 0.995: 신뢰도

  mask: 정상치 판별 결과 (N x 1 배열, 0: 비정상, 1: 정상)

  ransacReprojThreshold=3: 정상치 임계거리 (method가 RANSAC, RHO일 경우)

  RANSAC: Random Sample Consensus

  → 임의 좌표만 선정해 만족도 구하는 방식으로 만족도가 큰 것만 선정해 근사 계산

  → 임계값을 기준으로 넘으면 정상치 못넘으면 이상치

매칭 영역 원근 변환

- mtrx, mask = cv2.findHomography(srcPoints, dstPoints, method,
  ransacReprojThreshold, mask, maxiters, confidence)

method=0: 근사 계산 알고리즘 선택, 0: 모든 점으로 최소 제곱 오차 계산

method 종류

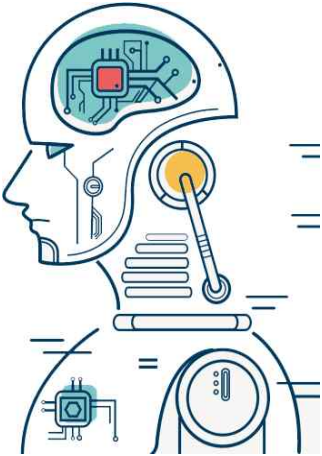
RANSAC(Random Sample Consensus): 임의 좌표만 선정해 만족도 구하는 방식으로 만족도가 큰 것만 선정해 근사 계산 → 임계값을 기준으로 넘으면 정상치 못넘으면 이상치

LMEDS(Least Median of Squares): 제곱의 최소 중간값을 활용하나 정상치가 50% 이상인 경우에만 정상 작동

RHO(Progressive Sample Consensus): RANSAC과 유사하나 개선 알고리즘

OpenCV

배경 제거

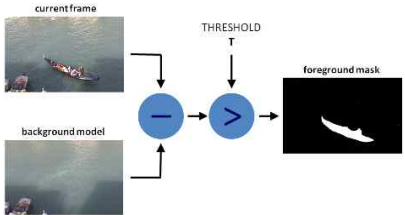


객체 추적(Object Tracking)

- 이미지에서 객체를 탐지
- 동영상에선 지속적으로 움직이는 객체를 찾는 방법

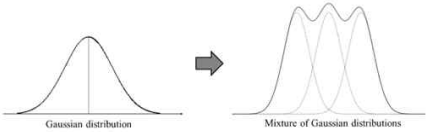
배경 제거(Background Subtraction)

- 객체를 파악하기 위한 방법으로 배경을 제거
- 객체를 포함하는 영상에서 객체가 없는 배경 영상을 빼는 방법
- 배경을 모두 제거하여 객체만 남기는 방법



배경 제거(Background Substraction)

- 배경은 일반적으로 시간이 지나도 크게 변함이 없다.
- 현재 프레임의 픽셀과 배경 픽셀 값을 비교하여 차이가 큰 부분을 움직이는 객체로 판단 → 고정된 카메라로 찍은 영상에서 정지 영상이 배경
- 각 픽셀에 대해 여러개의 가우시안 분포(Gaussian Mixture Model, GMM)를 활용한 배경을 모델링
- 특정 픽셀이 동일한 장소에 머물고 있는 시간 비율



배경 제거(Background Substraction)

- cv2.bsegm.createBackgroundSubtractorMOG(history, nmixtures, bacgroundRation, noiseSigma)

  history=200

  nmixtures=5 (가우시안 믹스처의 개수)

  backgroundRation=0.7(배경 비율)

  noiseSigma=0(노이즈 강도)

- foregroundmask = backgroundsubtractor.apply(img, foregroundmask, learningrate)

- backgroundImage = backgroundsubtractor.getBack(backgroundImage)

배경 제거(Background Substraction)

- cv2.createBackgroundSubtractMOG2(history,varThreshold, detectShadows)

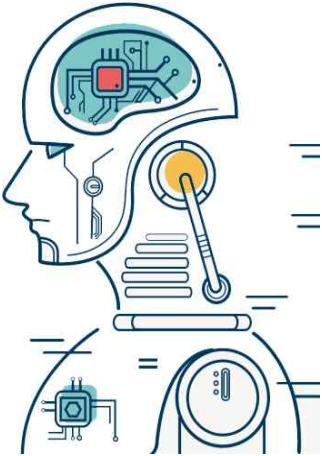
  varThreshold=16(분산 임계 값)

  detectShadows=True(그림자 표시)

- 각 픽셀에 대해 적절한 가우시안 분포 값을 선택해서 적용
- 빛의 변화가 심한 영상에 적용

OpenCV

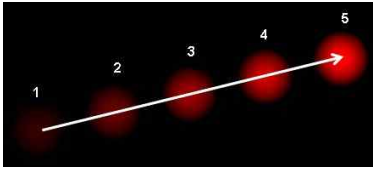
광학 흐름



광학 흐름(Optical Flow)

- 영상 내에서 물체의 움직임 패턴
- 이전 프레임과 다음 프레임 간 픽셀이 이동한 방향과 거리 분포
- Optical Flow를 통해 물체가 어느 방향으로 얼마나 움직였는지 파악 가능
- Optical Flow 계산의 두 가정

1. 연속된 프레임 사이에 움직이는 물체의 강도(intensity)의 변화가 없다.
2. 이웃하는 픽셀은 비슷한 움직임을 갖는다.



루카스-카나데(Lucas-Kanade) 알고리즘

- (3 x 3) 사이즈의 커널을 사용한 움직임 계산
- 이미지 피라미드를 활용

  작은 사이즈의 커널을 활용하기에 물체의 움직임이 크면 문제가 생기지만 이미지 피라미드의 상위단계에서 작은 움직임은 티가 덜 나고 큰 움직임은 작은 움직임 같아 보여 검출 가능
- 픽셀 전체를 계산하지 않으며, cv2.goodFeaturesToTrack()의 키포인트 계산

루카스-카나데(Lucas-Kanade) 알고리즘

- nextPts, status, err = cv2.calcOpticalFlowPyrLK(prevImg, nextImg, prevPts, nextPts, status, err, wirnSize, maxLevel, criteria, flags, minEigThreshold)

prevImg, nextImg: 이전, 다음 프레임 영상

prevPts, nextPts: 이전, 다음 프레임에서 이동한 코너 Keypoint

status: 결과 상태 백터, nextPts와 같은 길이로, 대응점이 있으면 1, 없으면 0

err

winSize=(21,21): 각 이미지 피라미드의 검색 윈도 크기

maxLevel=3: 이미지 피라미드 계층 수

루카스-카나데(Lucas-Kanade) 알고리즘

- nextPts, status, err = cv2.calcOpticalFlowPyrLK(prevImg, nextImg, prevPts, nextPts, status, err, wirnSize, maxLevel, criteria, flags, minEigThreshold)

criteria=(COUNT+EPS,30,0.001): 반복 탐색 중지 요건

max_iter: 최대 반복 횟수

epsilon: 최소 정확도

cv2.TERM_CRITERIA_EPS: 정확도가 epsilon보다 작으면 중지

cv2.TERM_CRITERIA_MAX_ITER: max_iter 횟수를 채우면 중지

cv2.TERM_CRITERIA_COUNT: MAX_ITER와 동일

flags=0:연산 모드

0: prevPts를 nextPts의 초기 값으로 사용

cv2.OPTFLOW_USE_INITAL_FLOW: nextPts의 값을 초기 값으로 사용

cv2.OPTFLOW_LK_GET_MIN_EIGENVALS: 오차를 최소 고유 값으로 계산

minEigThreshold=1e-4: 대응점 계산에 사용할 최소 임계 고유 값

군나르 파너백(Gunner Farneback) 알고리즘

- 밀집 방식으로 Optical Flow 계산 알고리즘
- 영상의 전체 픽셀을 활용한 계산으로 특징점 전달이 필요 없지만 느림
- flow = cv2.calcOpticalFlowFarneback(prev, next, pyr_scale, levels, winsize
  iterations, poly_n, poly_sigma, flags)

prev, next: 이전, 이후 프레임

flow:광학 흐름 계산 결과

pyr_scale: 피라미드 스케일 levels:피라미드 개수 winsize: 평균 윈도 크기

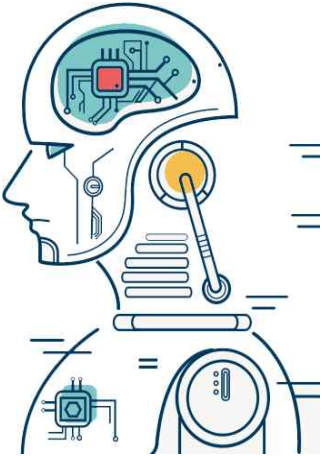
iterations: 피라미드 반복 횟수 poly_n: 다항식 근사를 위한 이웃 크기(5 or 7)

poly_sigma: 다항식 근사 활용 가우시안 시그마(poly_n=5: 1.1, poly_n=7: 1.5)

flags: cv2.OPTFLOW_USE_INITAL_FLOW, cv2.OPTFLOW_FARNEBACK_GAUSSIAN

OpenCV

객체 추적을 위한 TrackingAPI



Tracking API

- 추적하고자 하는 객체만 지정하면 API가 알아서 추적
- Tracking API종류

tracker = cv2.TrackerBoosting_create()	AdaBoost
tracker = cv2.TrackerMIL_create()	Multiple Instance Learning
tracker = cv2.TrackerKCF_create()	Kernelized Correlation Filters
tracker = cv2.TrackerTLD_create()	Tracking Learning and Detection
tracker = cv2.TrackerMedianFlow_create()	객체의 전/역방향 추적 후 불일치성 측정
tracker = cv2.TrackerGOTURN_create()	CNN기반 (OpenCV 3.4에선 동작 안됨)
tracker = cv2.TrackerCSRT_create()	Channel and Spatial Reliability
tracker = cv2.TrackerMOSSE_create()	내부적으로 그레이 스케일로 사용

Tracking API

- retval = cv2.Tracker.init(img, boundingBox) → Tracker 초기화

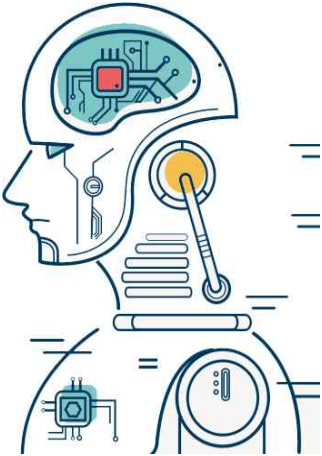
  boundingBox: 추적 대상 객체의 좌표 (x, y)

- retval, boundingBox = cv2.Tracker.update(img) → 프레임에서 객체 찾기

  retval: 추적 성공 여부, boundingBox: 새로운 프레임에서의 추적 대상 새로운 위치 (x, y, w, h)

OpenCV

HOG 디스크립터

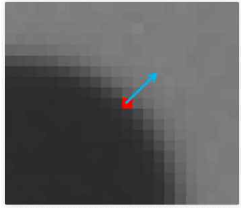


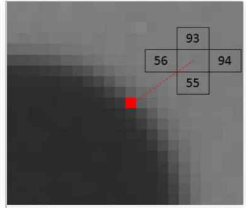
기울기 벡터(Gradient Vectors)

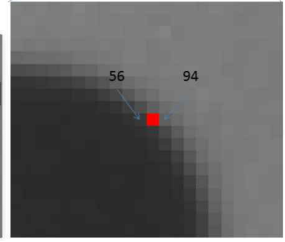
- 영상 내 하나의 픽셀 기준 주변 픽셀에 대한 기울기
- 그림의 빨간 픽셀 기준 주변 GrayScale 값의 차이로 기울기 계산 및 방향과 강도 표현

픽셀(Pixels), 셀(Cells), 블록(Blocks)

- 픽셀이 모여 셀, 셀이 모여 블록







HOG 보행자 인식

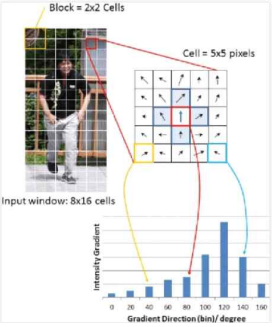
- 보행자 검출을 위해 만들어진 특징 디스크립터
- 이미지 경계의 기울기 벡터 크기(magnitude)와 방향(direction)의 히스토그램
- 검출하고자 하는 영역을 잘라 소벨 필터를 적용 하여 기울기 방향과 크기 계산
- Window = 8 x 16 Blocks = 16 x 32 Cells = 40 x 80 Pixel

  Block = 2 x 2 Cells

  Cell = 5 x 5 Pixels
- 기울기 벡터의 방향(각도)를 x축 bin으로 0~180 사이 값으

  하나의 셀을 기준으로 히스토그램 계산

  0~180 / 180~360 같은 방향 표현 각이 있어서



HOG 보행자 인식

- 보행자 검출에서 보통 Window = 64 x 128 Pixels, Cell = 8 x 8 Pixels
- 히스토그램 계산 후 정규화 → 경계 값의 밝기에 민감(픽셀 값 차이)을 완화
- 정규화는 블록(Cell의 2배, 16 x 16 Pixels)에 적용
- 블록은 전체 윈도우를 순회하며 진행하고 겹치는 부분(block stride)
- 64 x 128 Window, 16 x 16 Block, 8 x 8 Block Stride → 정규화 7*15=105회

HOG 보행자 인식

- descriptor = cv2.HOGDescriptor(winSize, blockSize, blockStride, cellSize, nbins)

  winSize: HOG 추출 영역(window)　　blockSize: 정규화 영역(block)

  nbins: 히스토그램 계급 수(x 갯수)

- hog = descriptor.compute(img)

scvmdetector = cv2.HOGDescriptor_getDefaultPeopleDetector(): 64 x 128 윈도 크기로 훈련된 모델

scvmdetector = cv2.HOGDescriptor_getDaimlerPeopleDetector(): 48 x 96 윈도 크기로 훈련된 모델

descriptor = cv2.HOGDescriptor(winSize, blockSize, blockStride, cellSize, nbins): HOG 생성

descriptor.setSVMDetector(svmdetector): 훈련된 SVM 모델 설정

rects, weights = descriptor.detectMultiScale(img): 객체 검출

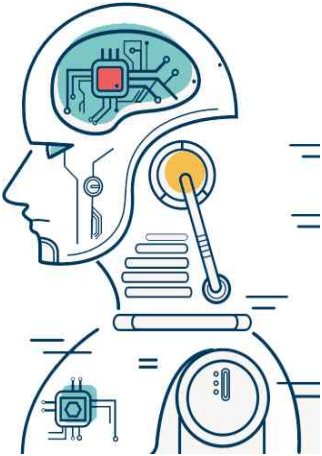
img: 검출하고자 하는 이미지

rects: 검출된 결과 영역 좌표 N x 4 (x, y, w, h)

weights: 검출된 결과 계수 N x 1

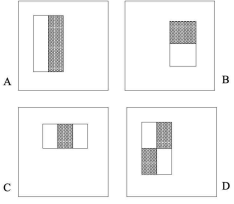
OpenCV

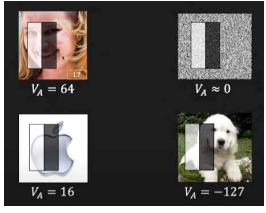
캐스케이드 얼굴 검출

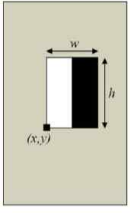


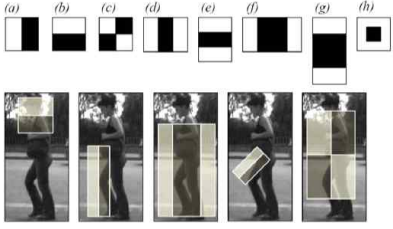
Haar 필터 사용

- Haar 필터 결과: 흰색 영역과 검은색 영역의 픽셀 합의 차이
- 다양한 형태의 필터를 활용한 경계rjacnf
- (a): 좌우방향, (b): 위아래방향, (c): 대각방향, (d): 가운데 영역의 특징 등









Haar 필터 사용

- 필터 내의 픽셀 합을 모든 필터, 스케일 당 진행 → 연산량이 매우 많다
- 통합 이미지(integral image): 첫 픽셀부터 해당되는 위치까지의 픽셀 합 미리
- 차연산 4번으로 영역 비교가 가능

Original

5	2	3	1
1	5	4	3
2	2	1	4
3	5	6	5
4	1	3	6

Integral

5	7	10	14	15
6	13	20	26	30
8	17	25	34	42
11	25	39	52	65
15	30	47	62	81

Original

5	2	3	1
1	5	4	3
2	2	1	4
3	5	6	5
4	1	3	6

Integral

5	7	10	14	15
6	13	20	26	30
8	17	25	34	42
11	25	39	52	65
15	30	47	62	81

하르 캐스케이드 얼굴 검출기

- https://github.com/opencv/opencv/tree/master/data

  기존에 학습되어있는 Weight를 활용

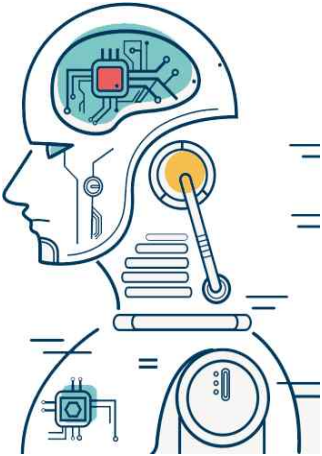
- cascade_face_detector = cv2.CascadeClassifier()
- face_detections = cascade_face_detector.detectMultiScale(img)

  scaleFactor=1.1 (큰얼굴 작게 작은얼굴 크게 조정하며 감지)

  minNeighbors (여러 후보 bounding box 생성 후 가장 잘 둘러싸는 것 선택)

OpenCV

LBPH 얼굴 인식



LBPH(Local Binary Patterns Histograms)

- 얼굴 검출만이 아닌 개인 식별을 위한 얼굴 인식
- 빠른 계산 속도와 단순한 알고리즘
- LBP: 영상에서 텍스쳐를 인식하거나 분석에 사용되는 특징 추출 기술
- 이미지의 각 픽셀 주변 픽셀 값을 이진수로 변환해 이미지 특징 추출

  중심 픽셀과 이웃 픽셀의 상대적 밝기 차이 따라 생성, 이웃이 중앙보다 크면 1 작으면 0

LBPH(Local Binary Patterns Histograms)

- Neighbors: 이웃 픽셀 수

  LBP 만들 때 사용할 이웃 픽셀 수, 많아질수록 계산량이 많아짐

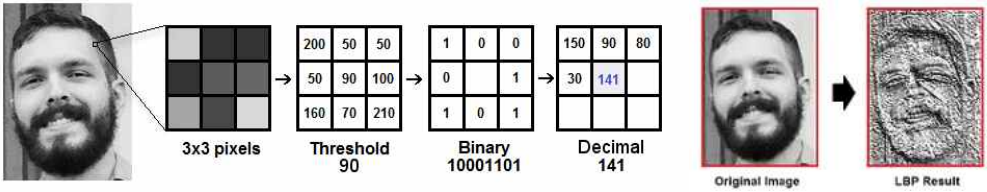
- Grid X: 수평 방향 분할수, Grid Y: 수직 방향 분할 수

  수평, 수직 방향으로 셀을 분할할 개수 (보통 8)

- LBP처리 하며 얼굴의 주목할 특징을 강조

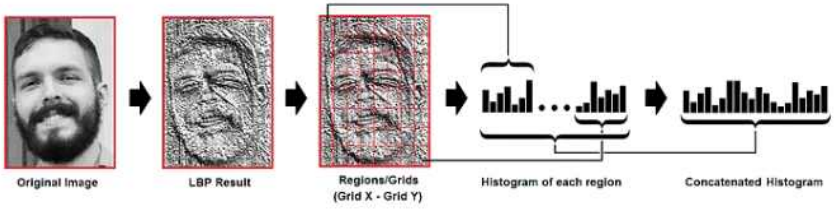
LBPH(Local Binary Patterns Histograms)

- 원본 흑백 이미지에 3 x 3 픽셀의 윈도로 계산
- 윈도 내의 가운데 픽셀의 값을 임계값으로 설정 후 임계값에 따라 0, 1 처리



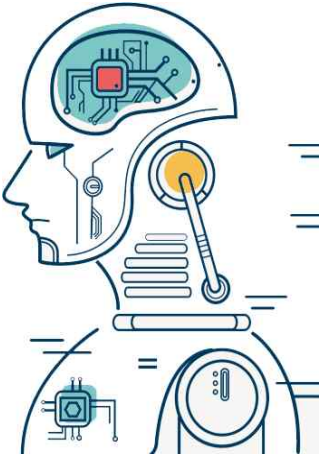
LBPH(Local Binary Patterns Histograms)

- 히스토그램 만들기
- 변환된 이미지를 (Grid X, Grid Y)로 분할하여 히스토그램 그리기



OpenCV

강의 리뷰



특징 디스크립터 검출기

- SIFT
- ORB

특징 매칭

- BFMatcher
- FLANN

올바른 매칭 찾기

- 매칭 영역 원근 변환

배경 제거

- 영상 속 멈춰있는 픽셀 제거

광학 흐름

- 물체의 움직임 패턴

객체 추적을 위한 Tracking API

HOG 디스크립터