가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA

영상 특징 검출

미디어기술콘텐츠학과
강호철

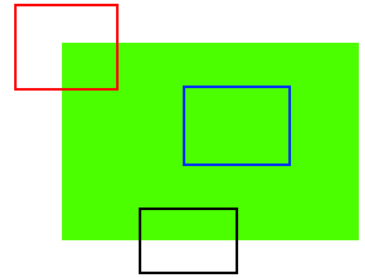
코너 검출

- 특징 (Feature)
  - 영상으로 부터 추출한 유용한 정보
  - 밝기 평균, 히스토그램, 에지, 코너
  - 전역 특징, 지역 특징

- 코너 검출 방법
  - 에지 방향이 급격히 변하는 부분
  - 꼭지점, 튀어나온 부분
  - 코너는 다른 지역 특징에 비해 분별력이 높고 영상 전체에 분포

코너 검출

- 코너
  - A: 평탄한 영역
  - B: 수평선
  - C: 수평선 + 코너



출처: https://m.blog.naver.com/samsjang/220611556387

코너 검출

- 코너점 검출
  - 미분 연산자에 의한 에지방향 이용
  - cv2.preCornerDetect(src, ksize)
    - dst의 local optima 값으로 검출

$$dst(x,y) = I_x^2 + I_{yy} + I_y^2 I_{xx} - 2I_x I_y I_{xy}$$

$$I_x = \frac{\partial I(x,y)}{\partial x}, I_y = \frac{\partial I(x,y)}{\partial y}, I_{xx} = \frac{\partial^2 I(x,y)}{\partial^2 x}$$

$$I_{yy} = \frac{\partial^2 I(x,y)}{\partial^2 y}, I_{xy} = \frac{\partial^2 I(x,y)}{\partial x \partial y}$$

출처: https://wjddyd66.github.io/opencv/OpenCV(7)/#%EC%BD%94%EB%84%88%EC%A0%90-%EA%B2%80%EC%B6%9C1-cv2precornerdetect

코너 검출

- 코너점 검출
  - cv2.preCornerEigenValsAndVecs(src, blockSize, ksize)
    - src : input
    - blockSize: 이웃 윈도우 크기
    - ksize sobel filter mask
    - 영상 내 각 이웃의 covariance matrix M의 eigenvalue, eigenvalue 를 계산하여 코너 검출

$$M = \begin{bmatrix} \sum_{Nbd(x,y)} I_x^2 & \sum_{Nbd(x,y)} I_x I_y \\ \sum_{Nbd(x,y)} I_x I_y & \sum_{Nbd(x,y)} I_y^2 \end{bmatrix}$$

- eigenvalue λ_1, λ_2가 모두 작은 값: 평평한 영역에 있는 점
- eigenvalue λ_1, λ_2 둘 중 하나는 크고 하나는 작은 값: 에지
- eigenvalue λ_1, λ_2 두 값이 모두 큰 값: 코너

출처: https://wjddyd66.github.io/opencv/OpenCV(7)/#%EC%BD%94%EB%84%88%EC%A0%90-%EA%B2%80%EC%B6%9C1-cv2precornerdetect

코너 검출

- 해리스 코너 검출 방법
  - 코너 검출 연구는 1970년대 후반부터 활발하게 진행
  - 1988년 해리스(C. Harris)가 개발한 코너 검출 방법은 코너 점 구분을 위한 기본적인 아이디어를 수학적으로 잘 정의하였다는 점에서 큰 의미가 있음 [Harris88]
  - 영상의 특정 위치 (x, y)에서 Δx와 Δy 만큼 떨어진 픽셀과의 밝기 차이를 다음 수식으로 표현함

$$E(\Delta x, \Delta y) = \sum_{x,y} w(x, y)\left[I(x + \Delta x, y + \Delta y) - I(x, y)\right]^2$$

코너 검출

- 해리스 코너 검출 방법
  - w(x, y): 균일한 값 또는 가우시안 형태의 가중치를 갖는 윈도우
  - E(Δx,Δy) 함수가 모든 방향으로 값이 크게 나타난다면 점 (x, y)는 코너라고 간주할 수 있음
  - 해리스는 수학적 기법을 적용하여 코너 응답 함수 R을 유도함
    - k : 보통 0.01 ~ 0.06
    - R이 0보다 충분히 큰 양수 : 코너
    - R이 0에 가까운 실수 : 평탄한 영역
    - R이 0보다 작은 음수 : 에지

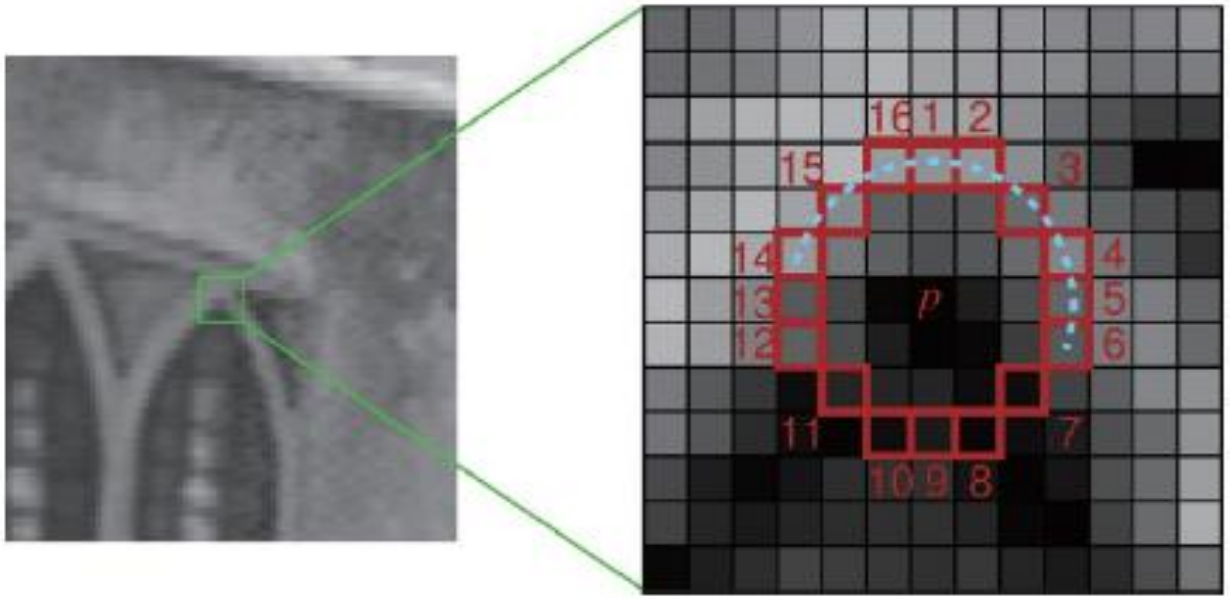
$$R = \mathrm{Det}(\mathbf{M}) - k \cdot \mathrm{Tr}(\mathbf{M})^2 \qquad \mathbf{M} = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

코너 검출

- 해리스 코너 구현
  - E(Δx,Δy) 함수가 모든 방향으로 값이 크게 나타난다면 점 (x, y)는 코너라고 간주할 수 있음
  - 해리스는 수학적 기법을 적용하여 코너 응답 함수 R을 유도함
    - k : 보통 0.04 ~ 0.06
    - R이 0보다 충분히 큰 양수 : 코너
    - R이 0에 가까운 실수 : 평탄한 영역
    - R이 0보다 작은 음수 : 에지

코너 검출

- FAST 코너 검출 방법 (2006년)
  - 16개의 주변 픽셀과 밝기를 비교하여 코너 여부 판별
    - p점 주변 1번부터 16번 픽셀과의 밝기 비교
    - 주변 16개의 픽셀 중에서 점 p보다 충분히 밝거나 또는 충분히 어두운 픽셀이 9개 이상 연속으로 존재하면 코너로 정의함 (원 논문은 12개)



코너 검출

- FAST 코너 검출 방법
  - 점 p에서의 밝기를 I_p라고 표현
  - 주변 16개의 픽셀 중에서 그 값이 $I_p + t$ 보다 큰 픽셀이 아홉 개 이상 연속으로 나타나면 점 p는 어두운 영역이 뾰족하게 돌출되어 있는 코너
  - 주변 16개의 픽셀 중에서 그 값이 $I_p - t$ 보다 작은 픽셀이 아홉 개 이상 연속으로 나타나면 점 p는 밝은 영역이 돌출되어 있는 코너
  - t는 밝고 어두움을 조절하기 위한 임계값
  - Non-maximal suppression

코너 검출

- FAST 코너 검출 방법
  - cv2.FastFeatureDetector
  - detect
  - setNonmaxSuppression
  - cv2.drawKeypoints

체스보드 패턴 코너점 검출

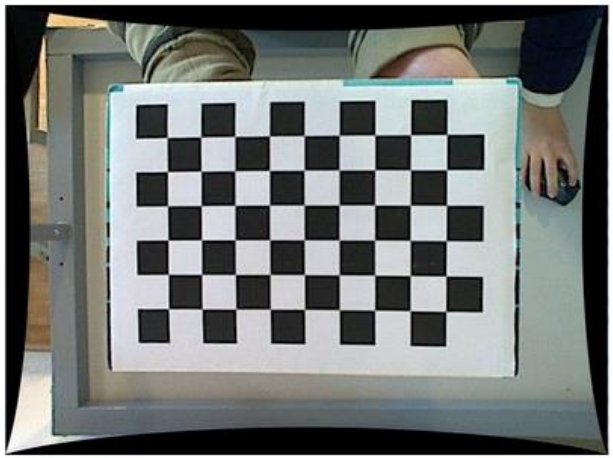
- 개념
  - 카메라 캘리브레이션에 자주 사용되는 체스보드 패턴 검출

`cv2.findChessboardCorners(image, patternSize[, corners[, flags]])` : 체스 판의 내부 모서리 위치를 찾는다.

parameter

- image: input(chessboard)
- patternSize: 체스보드 안의 한 칸 column, row size
- corners: output
- flag: 0이거나 다양한 값의 조합이 될 수 있음
- CALIB_CB_ADAPTIVE_THRESH
- CALIB_CB_NORMALIZE_IMAGE
- CALIB_CB_FILTER_QUADS
- CALIB_CB_FAST_CHECK

출처: https://wjddyd66.github.io/opencv/OpenCV(7)/#%EC%98%81%EC%83%81-%EB%AA%A8%EB%A9%98%ED%8A%B8



출처: https://docs.opencv.org/master/d4/d94/tutorial_camera_calibration.html

체스보드 패턴 코너점 검출

- 개념
  - 카메라 캘리브레이션에 자주 사용되는 체스보드 패턴 검출

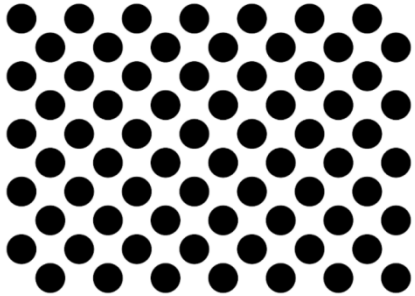
`cv2.drawChessboardCorners(image, patternSize, corners, patternWasFound)` : 검출된 코너점 배열 corners를 8비트 컬러 image에 표시

parameter

- image: Destination Image
- patternSize: Number of inner corners per a chessboard row and column
- corner: 검출한 코너점
- patternWasFound: Pattern을 검출하였나 안하였나 판단한는 Flag

`cv2.findCirclesGrid(image, patternSize[, centers[, flags]])` : 원 형태의 격자에서 원의 중심점을 검출한다.

출처: https://wjddyd66.github.io/opencv/OpenCV(7)/#%EC%98%81%EC%83%81-%EB%AA%A8%EB%A9%98%ED%8A%B8



모멘트

- 개념
  - 화소의 가중 평균으로 물체 인식을 위해 사용할 수 있는 desciptor
  - 영상 분할 후 각 객체에 대한 모멘트 계산
    - 면적, 무게중심, 기울어진 방향 등 정보 획득

- 계산
  - 공간 모멘트 M 행렬 정의

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y)$$

- 전체 합: $M_{00} = \sum_x \sum_y I(x,y)$
- x 합: $M_{10}(\bar{x}) = \sum_x \sum_y xI(x,y)$
- y 합: $M_{01}(\bar{y}) = \sum_x \sum_y yI(x,y)$

출처: https://wjddyd66.github.io/opencv/OpenCV(7)/#%EC%98%81%EC%83%81-%EB%AA%A8%EB%A9%98%ED%8A%B8

모멘트

- 계산
  - 중심 모멘트 $u_{ij} = \sum_x \sum_y (x - \bar{x})^i (y - \bar{y})^j I(x, y)$
  - 정규 중심 모멘트 $n_{ij} = \frac{u_{ij}}{u_{00}^{(i+j)/2+1}}$

    $n_{00}, n_{10}, n_{01} = 0$
  - Hu의 불변 모멘트
    - 정규 중심 모멘트를 이용하여 Hu의 7개 모멘트 계산
    - 이동, 스케일, 회전에 불변. (Hu 6은 영상 반사에 의해 부호 변경)

$$
\begin{aligned}
h_0 &= \eta_{20} + \eta_{02} \\
h_1 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
h_2 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
h_3 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
h_4 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
h_5 &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2 + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})] \\
h_6 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]
\end{aligned}
$$

출처: https://wjddyd66.github.io/opencv/OpenCV(7)/#%EC%98%81%EC%83%81-%EB%AA%A8%EB%A9%98%ED%8A%B8

모멘트

- 구현
  - 모멘트
    - cv2.moments(array[, binaryImage]) → retval
    - retval: 공간 모멘트, 중심 모멘트, 정규 중심 모멘트 반환
  - Hu의 불변 모멘트
    - cv2.HuMoments(m[,hu]) → hu
    - hu: h_0 ~ h_6까지 값 반환

모멘트를 이용한 매칭

- 모양 매칭
  - 템플릿 매칭의 한계
  - Hu의 불변 모멘트를 이용하여 물체의 윤곽선 모양 매칭
    - cv2.matchShapes(contour1, contour2, method, parameter)
    - 각 객체의 contour를 Hu 모멘트로 비교하여 matching 하는 방법
    - 객체와 유사할 수록 0에 가까운 수 반환

parameter

- contour1: Input Image1 Contour
- contour2: Input Image2 Contour
- method: 매칭 방법
- CV_CONTOURS_MATCH_I1: $I_1(A,B) = \sum_{i=0}^{6} |\frac{1}{m_i^A} - \frac{1}{m_i^B}|$
- CV_CONTOURS_MATCH_I2: $I_1(A,B) = \sum_{i=0}^{6} |m_i^A - m_i^B|$
- CV_CONTOURS_MATCH_I3: $I_1(A,B) = \sum_{i=0}^{6} max(\frac{|m_i^A - m_i^B|}{|m_i^A|})$
- $m_i^A = sign(hu(i)^A) x log(hu(i)^A)$
- $m_i^A = sign(hu(i)^A) x log(hu(i)^A)$

영상처리 프로그래밍 기초

- Python으로 배우는 OpenCV 프로그래밍
  - 김동근 지음
  - 가메출판사, 2018
- OpenCV4 로 배우는 컴퓨터 비전과 머신러닝
  - 황선규 지음
  - 길벗, 2019
- Visual C++ 영상처리 프로그래밍
  - 황선규 지음
  - 길벗, 2015