OpenCV를 이용한 harris corner 검출 코드 리뷰

2017104034 최시원

코너란?

- 코너 : 영상의 지역적 특성중 하나. 에지의 방향이 급격히 변하는 부분입니다.
- 픽셀값의 차이가 커 영상의 특징이라고 할 수 있는 튀어나온 부분입니다.
- 에지나 직선과 같은 다른 지역적 특성에 비해 분별력이 높기 때문에 영상 분석에서 의미있는 특징점으로써 유용한 분별기준이 됩니다.

- 이러한 코너 검출을 위해선 다음 두 가지 과정을 거칩니다
- 1. 영상의 특정 픽셀 위치와, 그 주변점들과의 x방향, y방향 밝기 차이 계산
- 2. 계산한 값이 x방향, y방향으로 모두 값이 크게 나타나는지 계산하여 코너 여부 판단

- OpenCV에서는 .cornerHarris() 함수를 이용합니다.

이미지 전처리

```
# 이미지파일을 불러와서 ima_@에 read한다.
file_1 = 'derivative_image.jpg'
img_1 = cv.imread(file_1)
file_2 = 'gaussian_filter_image.jpg'
img_2 = cv.imread(file_2)
file_3 = 'laplacian_filter_image.jpg'
img_3 = cv.imread(file_3)
```

- 코너검출을 위해 이미지를 전처리 합니다.
- 각각 Image Gradient, Gaussian filter, Laplacian filter를 거친 이미지를 준비합니다.
- 맨 처음은 원본이며, 이후 순서대로 변환된 이미지입니다.




.cornrHarris() 함수의 사용예시

```
# 해리스 코너를 검출하기 위한 코드.
gray1 = cv.cvtColor(img_1,cv.COLOR_BGR2GRAY)
gray1 = np.float32(gray1)
dst1 = cv.cornerHarris(gray1, 4, 3, 0.04)

gray2 = cv.cvtColor(img_2,cv.COLOR_BGR2GRAY)
gray2 = np.float32(gray2)
dst2 = cv.cornerHarris(gray2, 4, 3, 0.04)

gray3 = cv.cvtColor(img_3,cv.COLOR_BGR2GRAY)
gray3 = np.float32(gray3)
dst3 = cv.cornerHarris(gray3, 4, 3, 0.04)
```

- gray1은 불러온 img_1 이미지를 그레이스케일, float32 타입으로 변환해준 것입니다.

- .cornerHarris(a, b, c, d)
- a = float32타입의 그레이스케일 이미지
- b = 코너 검출을 위해 고려할 주변 픽셀 범위
- c = sobel미분에 사용할 인자값
- d = 해리스 코너 검출 상수

코너를 붉은 점으로 마킹

```
img_1[dst1>0.01*dst1.max()]=[0,0,255]
img_2[dst2>0.01*dst2.max()]=[0,0,255]
img_3[dst3>0.01*dst3.max()]=[0,0,255]
```

- 이미지의 적당한 부분을 붉은색으로 표시합니다.
- 0.01 부분을 바꾸면 붉은색 마킹의 결과가 달라집니다. 숫자가 클수록 붉은색 마킹이 작아지며 1일 경우 사라집니다.
- 기존 0.01과 0.05로 변경했을때 각각의 결과입니다.

이미지 전 처리의 세가지 방법

- 1. derivative.py
- 이미지 픽셀 변화에 대한 값을 1차 미분합니다.

- 2. laplacian_filter.py
- 이미지 픽셀 변화에 대한 값을 2차 미분합니다.

- 3. gaussian_filter.py
- 정규분포에 따른 필터를 이미지에 적용합니다.

derivative.py

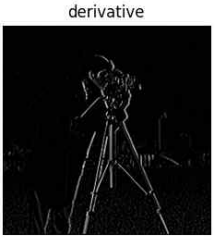
- 이미지에서 픽셀값의 변화가 가장 큰 부분은 일반적으로 edge, corner 입니다 .
- 이 코드에서는 수평 방향과 수직 방향으로 각각 미분하여 edge를 찾는 작업을 합니다.
- 다만 이미지는 연속된 데이터가 아니므로, 인접한 화소끼리의 차이를 구하는 연산을 합니다.
- derivative filter를 x축(수평), y축(수직) 기준으로 만들어 이를 이미지와 Convolution합니다.

```python
# Make derivative filter
filter_x = np.array([[-1, 0, 1]])
filter_y = np.array([[-1],
                     [0],
                     [1]])
```

```python
# Apply filter to image
dst = cv2.filter2D(img, -1, filter_x)
cv2.imwrite('derivative_image.jpg', dst)
```

```python
# Apply filter to image
dst = cv2.filter2D(img, -1, filter_y)
cv2.imwrite('derivative_image.jpg', dst)
```

derivative.py 결과

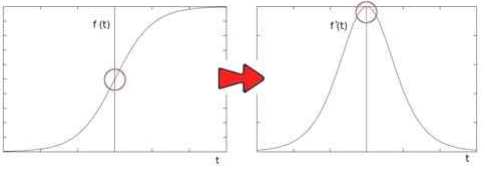


- 각각 filter_x, filter_y를 원본 이미지와 Convolution한 결과입니다.
- filter_x는 수평 방향으로 픽셀의 차이를 계산하였으므로 수직적인 특징이 도드라집니다.
- filter_y는 수직 방향으로 픽셀의 차이를 계산하였으므로 수평적인 특성이 도드라집니다.

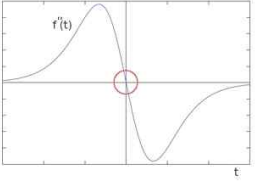
laplacian_filter.py

- 라플라시안 연산자는 2차 미분을 할때 필요한 연산자입니다.
- 이전에 수행했던 1차 미분에선 픽셀 변화 부분의 기울기에 따라 에지가 결정되었습니다. 라플라시안 필터는 이를 한차례 더 미분하여, 에지의 중심부, 즉 변곡점을 찾습니다. 조금 더 정확한 에지를 검출 가능합니다.

- 에지의 변화 부분과 그것의 1차 미분



2차 미분과 그 변곡점



laplacian_filter.py

```
# Make laplacian filter
filter = np.array([[-1, -1, -1],
                   [-1,  8, -1],
                   [-1, -1, -1]])
```

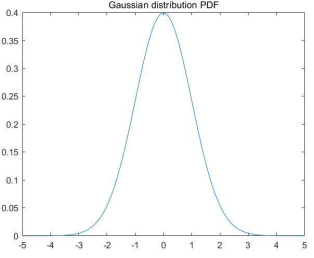
Original

Laplacian_filter

- 이 마스크는 라플라시안 필터를 적용하기 위한 필터입니다.
- 사용된 것은 8방향 마스킹입니다. 가로 세로뿐 아니라 양 대각선 방향으로 모두 미분 연산을 수행할 경우 나타나는 마스크입니다.
- 이를 적용하는 것으로 라플라스 연산자를 이용하는 것과 같은 효과를 얻을수 있습니다.
- 적용 결과는 다음과 같습니다.

gaussian_filter.py

- 중앙값을 도드라지게 보고 주변은 잘 안보이게 하는 마스킹의 형태입니다. 블러링bluring, 스무딩smoothing이라고 부르기도 합니다.
- 일반적으로 이미지를 흐리게 하거나 노이즈를 줄이는데 사용합니다.
- 이름답게 가우시안 분포, 즉 정규분포에 따라 필터마스크를 생성하여 적용합니다. 중앙부에서 비교적 큰 값을 가지고, 사이드로 갈수록 0에 가까운 작은 값을 가지게 됩니다.



gaussian_filter.py

```
# Get gaussian filter
filter = cv2.getGaussianKernel(ksize=5, sigma=1)
```

```
# Make 2D gaussian filter ( Option )
filter = filter * filter.T
```

- 가우시안 필터마스크를 생성하는 함수입니다. 커널 사이즈는 5이며, sigma는 가우시안 표준 편차로 1이 지정되어 있습니다.
- 자기자신과 행렬곱하여 2차원 필터를 생성합니다.

Original

Gaussian_blur

- 결과는 다음과 같습니다. 좌측의 원본 이미지와 비교하여 약간 흐릿해지며 부드러워진 모습을 볼 수 있습니다.