**OpenCV**

* **cv2.cornerHarris(src, blockSize, ksize, k)** : Harris corner detection을 수행한다
  + src : corner detection 을 수행할 grayscale 이미지
  + blockSize : corner를 구할 때, 고려할 block size
  + ksize : Gradient를 구할 때, sobel kernel size
  + K : 경험적 상수 K ( 0.01~ 0.06 을 많이 사용 )
  + Return : 해당 지점이 corner일 신뢰도
  + 신뢰도를 return 받기 때문에,
    - Cor = cv2.cornerHarris(src, blockSize, ksize, k) 후
    - Src[cor > 0.01 \* cor.max()] = [color]와 같이 threshold 해 주어야 한다. (이미지에 표시)
    - 수치는 자유롭게 곱하면 된다.
* **Cv2.dilate(src, kernel)** : 이미지를 팽창 시킨다.
  + Kernel을 np.ones((ksize,ksize))를 준다고 가정
  + 이 경우 현재 pixel을 기준으로 kernel 크기 만큼의 이웃에서 최댓값을 찾는다.
  + 해당 최댓값으로 kernel이 씌워진 부분을 채운다.
  + 즉, 주변 값의 최댓값이 현재 픽셀 위치에 씌워진 return이 나온다
  + Src : 팽창시킬 이미지
  + Kernel : 씌워줄 Kernel (size가 아니라 완성된 kernel을 넣어야 한다)
* **Cv2.erode(src, kernel)** : 이미지를 침식 시킨다.
  + Kernel을 np.ones((ksize,ksize))를 준다고 가정
  + 이 경우 현재 pixel을 기준으로 kernel 크기 만큼의 이웃에서 최소값을 찾는다.
  + 해당 최소값으로 kernel이 씌워진 부분을 채운다.
  + 즉, 주변 값의 최소값이 현재 픽셀 위치에 씌워진 return이 나온다
  + Src : 침식시킬 이미지
  + Kernel : 씌워줄 kernel

**Numpy**

* **Np.linalg.eigvals(arr)** : arr의 고유값을 리턴한다.
* **Np.linalg.eig(arr)** : arr의 고유값과, 고유벡터를 리턴한다.
* **Arr.min()** : Arr에서 최소값을 반환한다.
* **Arr.max()** : Arr에서 최대값을 반환한다.

**과제 관련 내용**

**주의**

* My\_HCD\_pixel.py 파일에서는 굉장히 느리게, 픽셀마다 연산을 수행하도록 되어 있습니다. 저 코드는 참조만 해주세요.
* 반드시 과제를 수행하며 작성하는 코드는 이미지 단위의 연산을 수행해주세요.
* **Local maxima 구하기**
  + Dilate와 erode를 적절히 사용하고, bit연산을 이용해서 수행하면 비교적 간단합니다.
  + Erode를 통해 구한 최소값 보다는 크고 dilate를 통해 구한 최대값과 같은 위치가 local maxima입니다.
* **Threshold**
  + Corner detection을 수행 한 뒤, R 값이 입력된 배열을 Return 받게 된다.
  + 해당 R의 최대값인 R.max()에 대해 일정 비율을 주고, 그 비율보다 작은 값을 가졌다면 0으로 만들어준다.
  + 보통은 0.01 정도를 사용해 수행한다.

**이 아래쪽은 궁금하신 분만 보세요. 짧은 지식이나마 작성 해 두었습니다.**

**Harris corner detection**

* 왜 행렬 M이 선형 이동 ? 어떤 행렬 A = [ 가 있을 때, A에 어떤 벡터 를 곱하면, 좌표 평면 상에서의 어떤 선형 이동(회전, 확대 등)으로 나타난다.
* **고유벡터(eigenvector)** : 원래의 view에 어떤 왜곡(Distortion)(선형 이동)을 준다고 했을 때, 변환 전과 변환 후 크기만 다르고 같은 방향을 가지는 Vector
  + A \* [고유벡터(x,y)]의 행렬 곱은 고유값 \* ( x,y )와 같다(A \* (x,y) = )
  + 어떤 vector의 처음과 끝을 이은 선을 span이라고 하는데, 선형 변환시 span을 벗어나지 않는 vector라고 이해할 수 있다.
* **고유값(eigenvalue)** : 고유벡터의 크기가 변환된 배율
* **Det(A**) : 행렬식 A = ad-bc = 이는 두 고유 값의 곱과 같다. (
* **Trace(A)** : 대각합 A = a+d = 이는 두 고유 값의 합과 같다.
* **Error function**
  + 이미지 상에서 Window(Block)를 움직였을 때, 생기는 영역의 밝기 변화량.
  + 이 때, 움직인 좌표가 (u,v)라면
  + 이 때, (u,v)가 0에 가까운 굉장히 작은 값이라고 했을 때,
  + 이 때, A = BC 라면 , 이다. 의 경우 결과값이 (1,1)인 하나의 값이다. ( 그냥 계산 후 풀어서 제곱을 취해줘도 된다. )
  + 따라서 다음이 성립한다.
  + 결국 위와 같은 식으로 근사 되며, 이동거리 (u,v)에 따른 변화량은 공분산 행렬 M에 따라서 결정된다.
  + 위의 식에서, 두 고유값 가 모두 크다면 두 방향으로의 변화량이 모두 크다는 의미가 된다.
  + 대각 성분을 기준으로 대칭인 Symmetric matrix의 경우, 두 고유벡터는 항상 직교한다.
  + 따라서, 고유벡터 방향으로의 변화량이 모두 크다면 해당 지점은 corner로 간주할 수 있다.
    - 아래 그림에서 타원의 두 지름 상에 고유벡터가 각각 존재한다고 생각하면 편하다.
    - 두 고유값이 큰 값을 가진다는 것은 두 방향으로의 변화량이 모두 크다는 것 = Corner
  + Harris corner detection에서는 이를 직접 구하지 않고, det와 trace를 이용해서 계산한다.