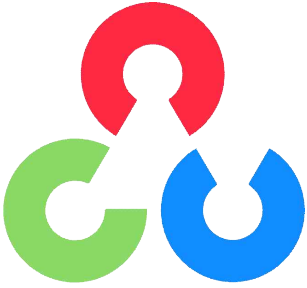
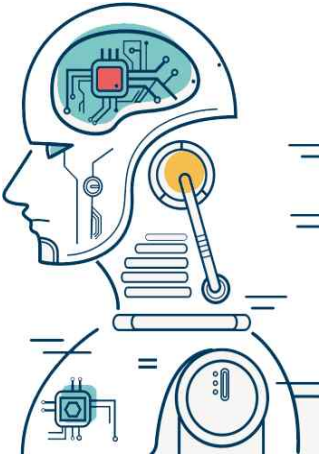


OpenCV

양재원

OpenCV

지난 강의 리뷰



컬러 스페이스

- BGR, BGRA, HSV, LUV, Gray Scale

스레시 홀딩

- 전역 스레시홀딩

  임계점 기준 두 가지로 나누는 방법

- 오츠 알고리즘

  최적의 임계점을 찾는 알고리즘

- 적응형 스레시홀딩

  이미지를 영역으로 나눠 주변 픽셀을 활용한 영역 별 임계값

이미지 연산

- 이미지 사이 단순 연산
- 알파 블렌딩

  합성하는 이미지 사이의 Weight를 부여한 합성
- 비트와이즈 연산

  특정 영역만 선택 혹은 제외하는 선별적 연산
- 이미지 차연산
- 이미지 합성과 마스킹

  블렌딩과 마스킹을 통한 이미지 사이의 합성

히스토그램

사진 합성

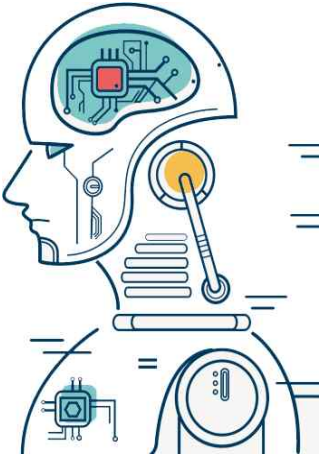
- 사람 얼굴과 해골 사진 합성하기




- 경계선이 뚜렷하며 어색 → Alpha Blending

OpenCV

히스토그램



히스토그램

- 도수 분포표를 그래프로 나타낸 것

  = 무엇이 몇 개 있는지 개수를 세어 놓은 것을 그래프로 나타낸 것

- Vision에서의 활용도?

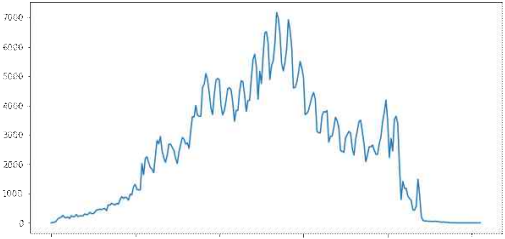
  이미지나 영상에서 0~255까지의 픽셀 값이 몇 개 인지 그래프로 나타냄

- 이미지나 영상의 픽셀들의 명암이나 색상의 분포를 파악

히스토그램

- Grayscale 이미지의 히스토그램

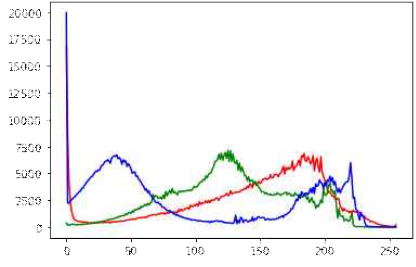
  → cv2.calcHist([img], [0], None, [256], [0,256])



히스토그램

- 컬러 이미지의 RGB 각각 채널의 히스토그램

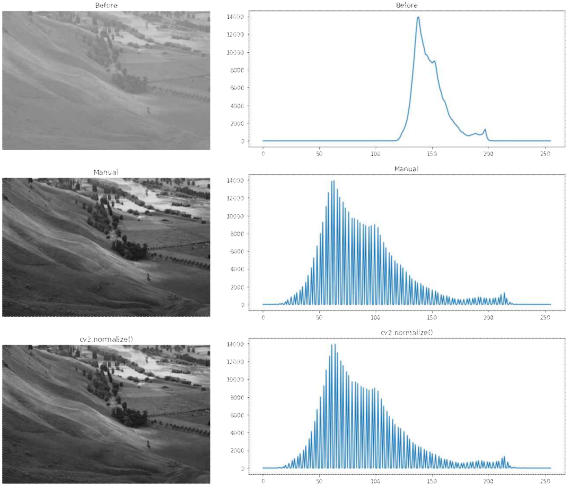
  → RGB 각 채널에 해당되는 cv2.calcHist를 for문을 통해



정규화(Normalization)

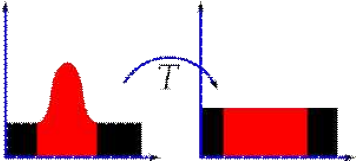
- 정규화를 이용한 화질 개선

  히스토그램 분포가 고르게 변화



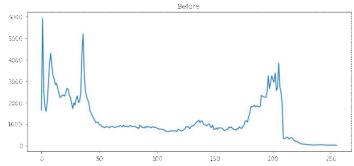
평탄화(Equalization)

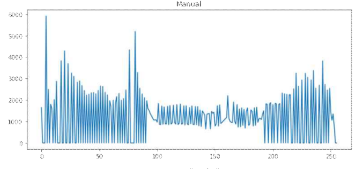
- 정규화는 분포가 한곳에 집중된 경우 효과적이나
  집중된 영역에 멀어질수록 효과가 떨어져
- 각각의 값이 전체 분포에 차지하는 비중에 따라 분포를 재분배
  명암 대비 개선 효과
- cv2.equalizeHist(src, dst)
  src: 대상 이지미, 8비트 1 채널
  dst: 결과 이미지

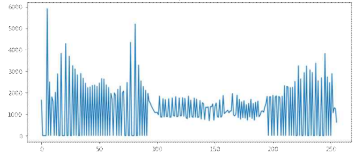


평탄화(Equalization)

- 평탄화를 이용한 명암 대비 개선
  히스토그램 분포가 고르게 변화



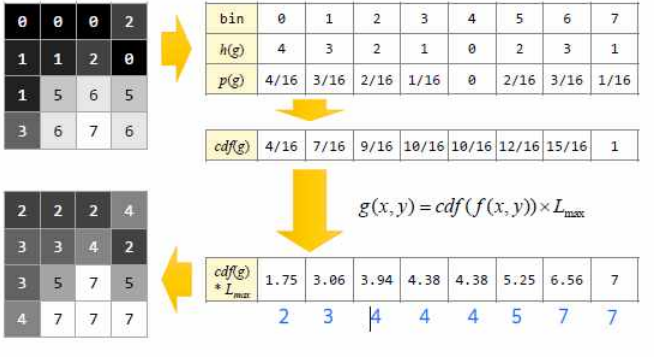




평탄화(Equalization)

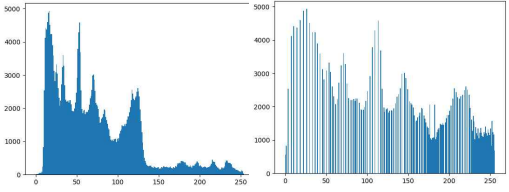
- 평탄화를 이용한 명암 대비 개선

  히스토그램 분포가 고르게 변화



CLAHE(Contrast Limiting Adaptive Histogram Equalization)

- 데이터 전체에 평탄화 적용 시 너무 밝은 부분이 생겨 날아가는 현상 방지

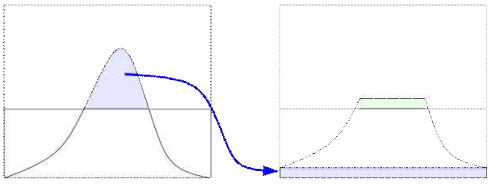


CLAHE(Contrast Limiting Adaptive Histogram Equalization)

- 평탄화 되는 부분을 일정한 영역으로 나눠 평탄화
- 노이즈가 증폭되는 것을 막기 위해 히스토그램 bins 혹은 제한 값을 넘으면 그 픽셀은 다른 bins로 배분 하고 평탄화 적용
- cv2.createCLAHE(clipLimit, tileGridSize)

  clipLimit: Contrast 제한 경계 값 (default: 40.0)

  tileGridSize: 영역의 크기 (default: 8 x 8)
- clahe.apply(src)



CLAHE(Contrast Limiting Adaptive Histogram Equalization)

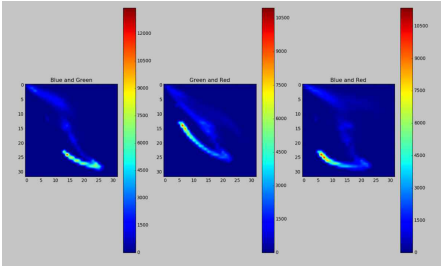
- clahe = cv2.createCLAHE()로 객체를 생성

  → clahe.apply(img)를 적용

- 밝은 부분을 남기고 다른 부분에 대한 보정

2차원 히스토그램(2D Histogram)

- 1차원 히스토그램은 이미지 픽셀 개수 표현
- 2차원 히스토그램은 축이 2개로, 각 축이 만나는 지점의 개수 표현



역투영(Back Projection)

- 2차원 히스토그램과 HSV 컬러 스페이스를 활용한 색상으로 특정 물체나 일부를 배경에서 분리
- 관심 영역의 히스토그램과 유사한 히스토그램을 갖는 영역을 찾는 방법
- cv2.calcBackProject(img, channel, hist, ranges, scale)

  channel: 처리할 채널

  hist: 역투영에 사용할 히스토그램

  ranges: 각 픽셀이 갖는 범위

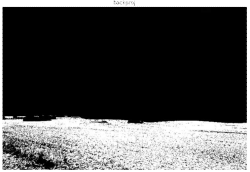
  scale: 결과에 적용할 배율 계수

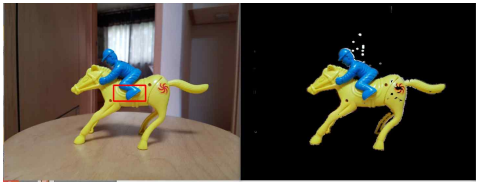
- Alpha channel, Chromakey와 같이 복잡함 없이 사물을 분리 가능

  관심 영역의 색상과 비슷한 다른 물체가 섞이면 효과가 떨어짐

역투영(Back Projection)

- RoI를 선택하여 유사한 부분 역투영하기 실습





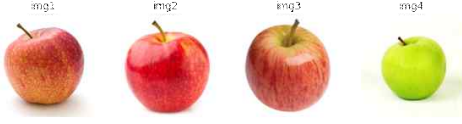
히스토그램 비교

- 히스토그램: 데이터의 픽셀 값의 분포를 갖는 성분
- 히스토그램 사이 비교

  → 데이터 사이의 사용한 픽셀의 색상 비중의 비슷한 정도 파악

- cv2.compareHist(hist1, hist2, method)

  비교할 2개의 히스토그램의 크기와 차원이 같아야 사용 가능

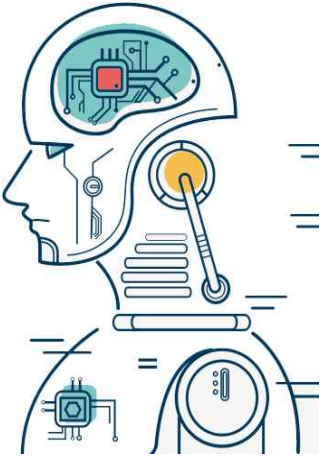



```
CORREL       img1:  1.00   img2:  1.00   img3:  1.00   img4:  0.01
CHISQR       img1:  0.00   img2: 40.49   img3:  7.94   img4:44417.81
INTERSECT    img1:  1.00   img2:  0.56   img3:  0.62   img4:  0.06
BHATTACHARYYA img1: 0.00   img2:  0.49   img3:  0.44   img4:  0.96
```

cv2.HISTCMP_CORREL : 상관관계(1:일치, -1:불일치, 0:무관계)
cv2.HISTCMP_CHISQR : 카이제곱(0:일치, 큰 값: 불일치)
cv2.HISTCMP_INTERSECT : 교차(1:일치, 0:불일치)
cv2.HISTCMP_BHATTACHARYYA : 바타차야(0:일치, 1:불일치)
cv2.HISTCMP_HELLINGER : 바타차야와 유사

OpenCV

이미지 변환



이미지 이동(Translation)

- 원래의 좌표에서 이동시킬 거리만큼 더하는 등의 연산

$$x_{new} = x_{old} + d_x,\ y_{new} = y_{old} + d_y$$

$$x_{new} = x_{old} + d_x = 1 * x_{old} + 0 * y_{old} + d_x, y_{new} = y_{old} + d_y = 0 * x_{old} + 1 * y_{old} + d_y$$

$$\begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} d_x + x \\ d_y + y \end{pmatrix}$$

- Linear Transformation 시켜주는 변환 행렬은 2 x 3
- 반환 행렬을 곱 연산하여 원하는 좌표로 이동

이미지 이동(Translation)

- dst = cv2.warpAffine(src, matrix, dsize, dst, flags, borderMode, borderValue)

src: 원본 이미지

matrix: 2 x 3 반환 행렬, dtype = float32

dsize: 결과 이미지의 크기, (width, height)

flags: 보간법 알고리즘 플래그

borderMode: 외곽 영역 보정 플래그

borderSize: cv2.BORDER_CONSTANT 외곽 영역 보정 플래그 시 사용할 색상 값

dst: 결과 이미지

flags

cv2.INTER_LINEAR(default): 인접한 4개 픽셀 값에 거리 가중치 사용

cv2.INTER_NEAREST: 가장 가까운 픽셀 값 사용

cv2.INTER_AREA: 픽셀 영역 관계를 이용한 재샘플링

cv2.INTER_CUBIC: 인정합 16개 픽셀 값에 거리 가중치 사용

borderMode

cv2.BORDER_CONSTANT: 고정 색상 값

cv2.BORDER_REPLICATE: 가장자리 복제

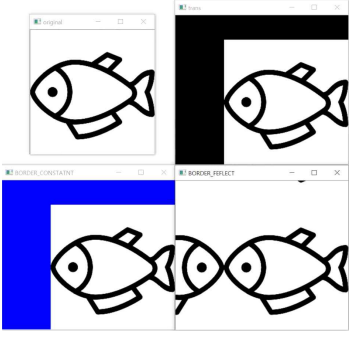
cv2.BORDER_WRAP: 반복

cv2.BORDER_REFLECT: 반사

이미지 이동(Translation)

- 이미지 가로(x) 방향, 세로(y) 방향 이동
- 평행 이동 시킬 시 기존 영역의 이미지가 잘리는 현상

  borderMode parameter를 통해 조정



이미지 확대 및 축소(Scaling)

- 원래의 이미지를 일정 비율로 확대 및 축소

$$x_{new} = \alpha * x_{old},\ y_{new} = \beta * y_{old}$$

$$x_{new} = \alpha * x_{old} = \alpha * x_{old} + 0 * y_{old} + 0 * 1, y_{new} = \beta * y_{old} = 0 * x_{old} + \beta * y_{old} + 0 * 1$$

$$\begin{pmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha * x \\ \beta * y \end{pmatrix}$$

- Scaling 시켜주는 변환 행렬은 2 x 3 → 함수에 입력해야 되는 행렬의 크기
- 반환 행렬을 곱 연산하여 원하는 비율로 확대 및 축소

이미지 확대 및 축소(Scaling)

- cv2.resize(src, dsize, dst, fx, fy, interpolation)

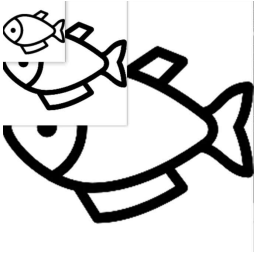
  src: 원본 이미지

  dsize: 결과 이미지의 크기, (width, height)

  fx, fy: 크기 배율, dsize가 주어지면 dsize를 적용

  interpolation: 보간법 알고리즘 선택 플래그 (cv2.warpAffine()과 동일)

  dst: 결과 이미지



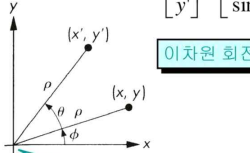
이미지 회전(Rotation)

- 원래의 이미지를 일정 각도로 회전
- 변환 행렬에 사용할 회전 각은 60진법에 따른 radian으로 변경

$1 \text{ radian} = 180/\pi$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

이차원 회전 변환의 행렬 표현



이미지 회전(Rotation)

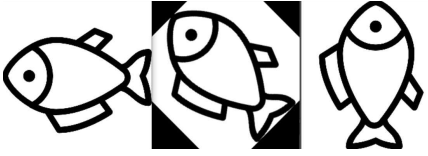
- 변환 행렬을 이용한 이미지 회전

$$x_{new} = \cos\theta * x_{old} - \sin\theta * y_{old},\ y_{new} = \sin\theta * x_{old} + \cos\theta * y_{old}$$

$$\begin{pmatrix} \cos\theta & -\sin\theta & ratio_1 \\ \sin\theta & \cos\theta & ratio_2 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x_{new} \\ y_{new} \end{pmatrix}$$

- $ratio_1$과 $ratio_2$는 회전으로 인한 프레임 바깥으로 벗어나는 것

  회전 후 평행 이동을 하여 프레임 안으로 들어오도록



이미지 회전(Rotation)

- mtrx = cv2.getRotationMatrix2D(center, angle, scale)

  center: 회전축 중심 좌표 (x, y)

  angle: 회전할 각도, radian

  scale: 확대 및 축소 비율

이미지 뒤틀기(Wraping)

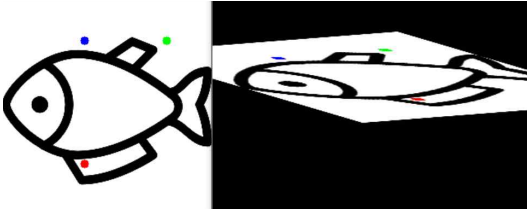
- Translation, Rotation, Scaling과 달리 모양이 달라지는 방법
- 어핀 변환(Affine Transform), 원근 변환(Perpective Transform)

어핀 변환(Affine Transform)

- 어핀 변환 전과 후의 3개의 점을 짝 지어 매핑하여 뒤틀기
- mtrx = cv2.getAffineTransform(pts1, pts2) → 2 x 3 행렬

  pts1: 변환 전 영상의 좌표 3개, 3 x 2

  pts2: 변환 후 영상의 좌표 3개, 3 x 2
- pts1이 pts2로 위치가 변한 만큼 이미지를 뒤트는 함수



원근 변환(Perspective Transform)

- 원근감은 3차원 좌표계로 느껴져 동차 좌표(homogeneous coordinates) 활용
- 원근감을 느끼기 위한 (x, y, 1)꼴의 좌표계가 필요하며 3 x 3 반환 행렬식

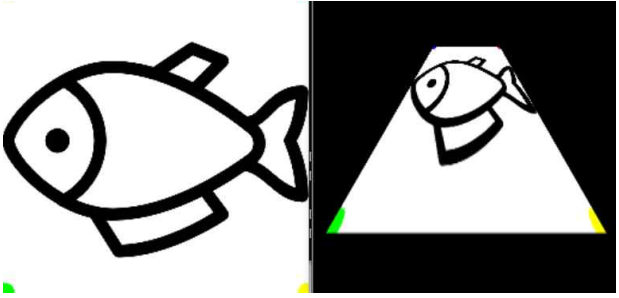
- $\omega \begin{pmatrix} x_{new} \\ y_{new} \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \cdot \begin{pmatrix} x_{old} \\ y_{old} \\ 1 \end{pmatrix}$, ω: homogeneous coordinates

원근 변환(Perspective Transform)

- mtrx = cv2.getPerspectiveTransform(pts1, pts2) → 3 x 3 행렬

  pts1: 변환 전 영상의 좌표 4개, 4 x 2, dtype = float32

  pts2: 변환 후 영상의 좌표 4개, 4 x 2



삼각형 어핀 변환(Delaunay Triangulation)

- 어떤 영역을 여러 개의 삼각형으로 나누는 기법: Delaunay Triangulation
- OpenCV는 기본적으로 영상을 대상으로 하여 사각 기하학적 변환 제공
- 영상 분야에서 하나의 물체가 다른 물체로 자연스레 변하는 morphing 기술
- 삼각형 어핀 변환

1) 변환 전, 후 삼각형 좌표 3개
2) 삼각형 좌표를 감싸는 외접 사각형 좌표
3) 2)번 과정의 사각형 영역을 RoI로 지정
4) 3)번 과정의 관심 영역을 기준으로 변환 전, 후의 삼각형 좌표 계산
5) 4)번 과정의 삼각형 좌표로 어핀 변환 행렬 계산
6) 5)번 과정의 어핀 변환 행렬로 어핀 변환
7) 6)번 과정의 변환된 RoI의 변환 후 삼각형 좌표만 마스킹
8) 7)번 과정의 마스크와 원본 이미지를 합성

삼각형 어핀 변환(Delaunay Triangulation)

- 삼각형 좌표를 감싸는 외접 사각형 좌표 구하기 위한 함수

  x, y, w, h = cv2.boundingRect(pts)

  pts: 다각형 좌표

  x, y, w, h: 외접 사각형의 좌표, 폭, 높이

- 마스크를 구하기 위한 함수

  cv2.fillConvexPoly(img, pts, color, lineTypes)

  pts: 다각형 좌표

  color: 다각형을 채울 색상

  lineType: 선 그리기 알고리즘 플래그

렌즈 왜곡(Lens Distortion)

- 변환 행렬을 통해 구할 수 없는 변환
- 렌즈 왜곡 변환 → 리매핑, 오목 렌즈 왜곡, 볼록 렌즈 왜곡, 방사 왜곡
- 일정한 규칙을 지니거나 수학적인 계산으로 모든 픽셀에 적용되지 않는 변환

리매핑(Remapping)

- 규칙성 없이 마음대로 이미지 모양을 변환
- dst = cv2.remap(src, mapx, mapy, interpolation, dst, borderMode,
  borderValue

  mapx, mapy: x축과 y축으로 이동할 좌표, src와 동일한 크기, dtype = float32

  cv2.wrapAffine()과 동일한 parameter
- 기존 픽셀을 원하는 위치로 재배치 하도록 하는 함수

리매핑(Remapping)

- remap() 함수는 특히 변환 행렬로 표현이 안되는 비선형 변환에 사용

변환 행렬로 실행 시 속도의 차이(ex) 삼각함수를 이용한 비선형 리매핑






오목 렌즈와 볼록 렌즈 왜곡

- 직교 좌표계(Cartesian Coordinate System)

  x축과 y축의 직각으로 각각 선을 그어 만나는 지점의 좌표를 (x, y)

  기준점: (0, 0)

- 극좌표계(Polar Coordinate System)

  원점으로부터 거리(r)와 사잇각(theta)를 이용한 (r, theta)

  기준점: 이미지의 중점

- r, theta = cv2.carToPolar(x, y): 직교 좌표 → 극 좌표

  x, y = cv2.polarto Cart(r, theta): 극 좌표 → 직교 좌표

- 기준점 변환도 함께 이루어져야 하여 좌표의 값을 -1 ~ 1 사이로 위한 정규화

오목 렌즈와 볼록 렌즈 왜곡

- 렌즈 왜곡 과정

  (직교 좌표 → 극 좌표 변환) → 왜곡 진행 → (극 좌표 → 직교 좌표 변환)

- 렌즈 왜곡 효과

  렌즈의 반지름 길이를 설정하여 해당 영역을 왜곡

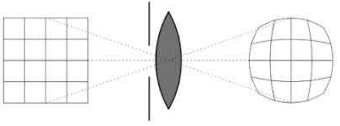
  왜곡 시키는 정도를 나타내는 값: ‘exp’

  → 0<exp<1: 오목 / exp>1: 볼록

방사 왜곡(Radial Distortion)

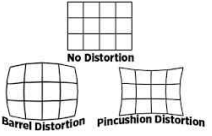
- 배럴 왜곡(Barrel Distortion)

  카메라를 통해 이미지를 촬영 시 생기는 가장자리의 왜곡 현상



- 핀쿠션 왜곡(Pincushion Distortion)

  가장자리 부분이 안쪽으로 들어가는 형태의 왜곡



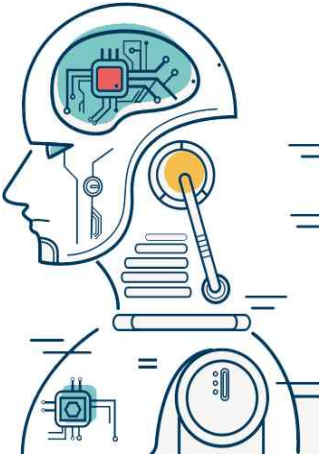
방사 왜곡(Radial Distortion)

- 왜곡 계수의 값에 따라 종류가 달라진다
- $r_u = r(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$

  k_1, k_2, k_3 값에 따라 배럴 왜곡, 핀쿠션 왜곡 조절

OpenCV

강의 리뷰



히스토그램

- 정규화, 평탄화, CLAHE
- 역투영

이미지 변환

- 이동

  이동 할 좌표만큼 더하여 이동

- 확대/축소

  원하는 비율만큼 곱하여 확대 및 축소

- 회전

  극 좌표를 직교 좌표로 변환하여 회전

이미지 변환

- 이미지 뒤틀기

  어핀 변환, 원근 변환, 삼각형 어핀변환

- 리매핑

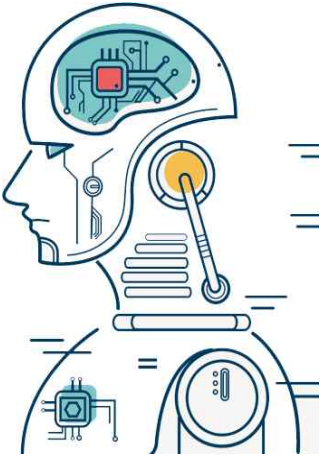
  규칙성 없이 마음대로 이미지 모양 변환

- 렌즈 왜곡

  오목 렌즈, 볼록 렌즈

OpenCV

실습 과제



카메라의 스캔 기능과 같이 찍힌 문서를 스캔한 문서처럼 만들기

- 마우스 이벤트를 통해 4개의 모서리를 설정
- 변환을 활용하여 원근감이 있는 평면 이미지로 바꿔주는 실습