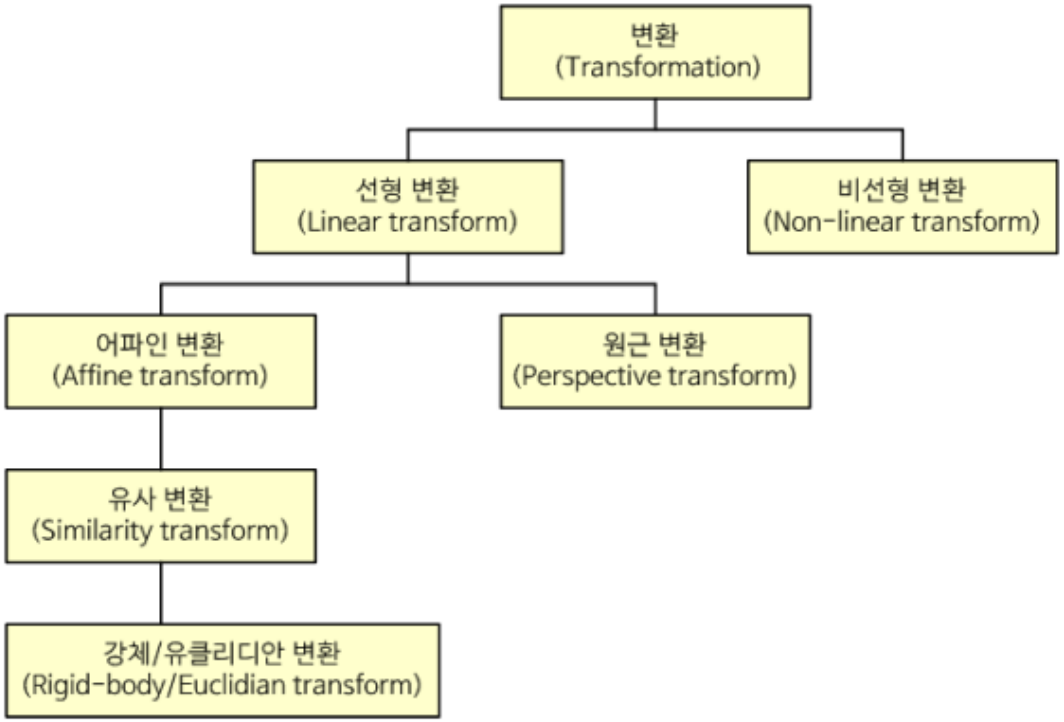
기하학적 변환

3.1 기하학적 변환의 종류

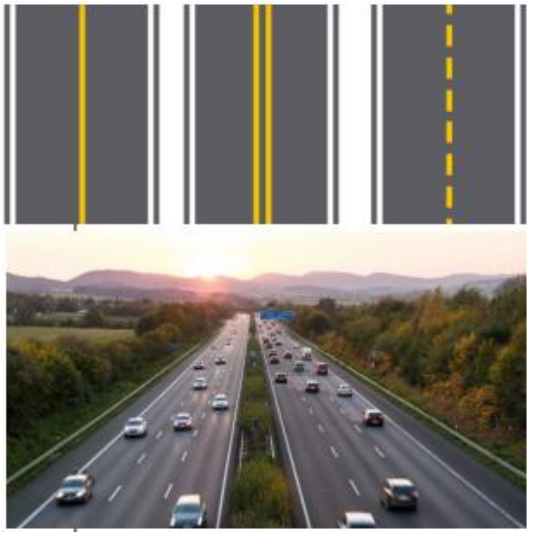
- 기하학적 변환(geometric transformation)
  - 영상을 인위적으로 확대, 축소, 위치 변경, 회전, 왜곡하는 등 영상의 형태를 변환하는 것을 의미
  - 영상을 구성하는 픽셀 좌표 값의 위치를 재배치

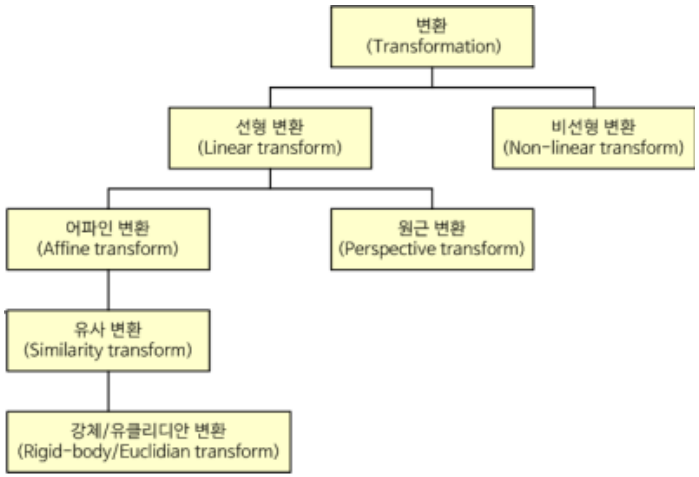
- 기하 변환 종류



3.1 기하학적 변환의 종류

- 기하 변환 종류
  - **강체/유클리디안 변환(rigid-body / euclidean transformation)**
    - 크기 및 각도가 보존되는 변환 (ex. Translation, Rotation)
  - **유사 변환(similarity transformation)**
    - 강체 변환과 균등 크기 조절 변환 및 반사 변환
    - 크기는 변하지만 각도는 보존되는 변환 (ex. Scaling)
  - **어파인 변환(affine transformation)**
    - 유사 변환과 차등 크기 조절 변환 및 전단 변환
    - 물체의 타입이 유지되고, 평행선이 보존되는 변환 (ex. 사각형-> 평행사변형)
  - **원근 변환(perspective transformation)**
    - 직전이 직선으로 유지되지만, 평행선이 만나는 변환
  - **선형 변환(linear transformation)**
    - 어파인 변환과 원근 변환
    - 선형 조합(linear combination)으로 표시되는 변환
  - **비선형 변환(non-linear transformation)**

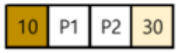


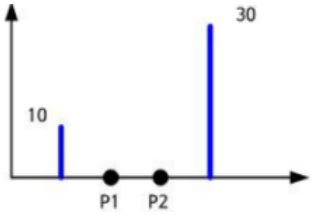


3.2 보간(interpolation)

보간 필요성

- 영상에 기하학적 변형을 수행하는 과정에서 좌표값이 정수가 아닌 실수로 계산되기 때문에 해당 위치에서 밝기값이 결정적으로 존재하지 않음
- 주변 픽셀들 밝기값을 고려하여 해당 위치에서 밝기값을 결정해야 하는데, 이와 같은 처리를 보간법(interpolation method)
- 1차원 신호, P1과 P2의 위치 신호값을 결정하는 방법





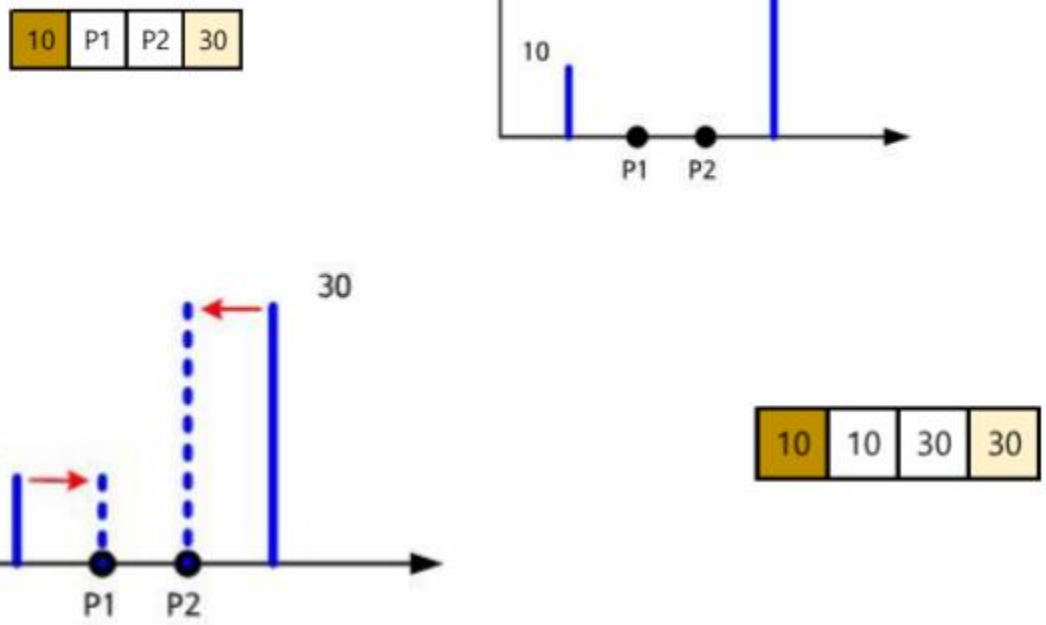
보간법 종류

- 주변 픽셀들의 범위와 이 픽셀들을 어떻게 고려하는지에 따라 다양한 보간 방법들이 존재
- 최소 인접 이웃 보간법(Nearest Neighborhood interpolation)과 양방향 선형 보간법(Bilinear interpolation)이 많이 활용되는 기법

3.2 보간(interpolation)

- 최소 인접 이웃 보간법 (Nearest Neighborhood Interpolation)
  - 해당 위치와 가장 가까운 화소의 값을 채택하는 방법

    ex) P1은 10, P2는 30으로 값을 채택

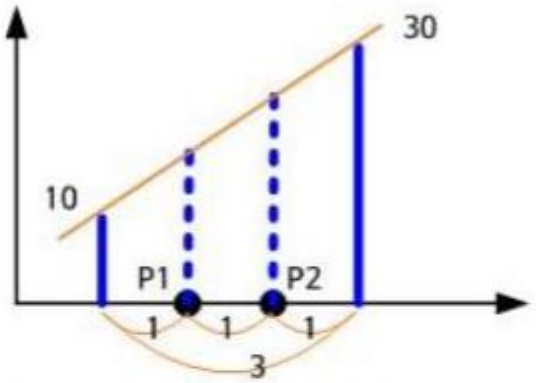


3.2 보간(interpolation)

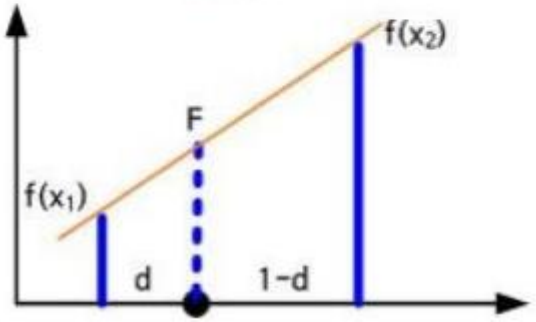
- 양방향 선형 보간법 (Bilinear Interpolation)
  - 인접한 신호들로부터 동시에 영향을 받으므로 거리에 반비례하도록 신호값을 계산하는 방법

ex) P1은 $10 \times \left(\frac{2}{3}\right) + 30 \times (\frac{1}{3})$로서 50/3으로 값을 채택

P2는 $10 \times \left(\frac{1}{3}\right) + 30 \times (\frac{2}{3})$로서 70/3으로 값을 채택



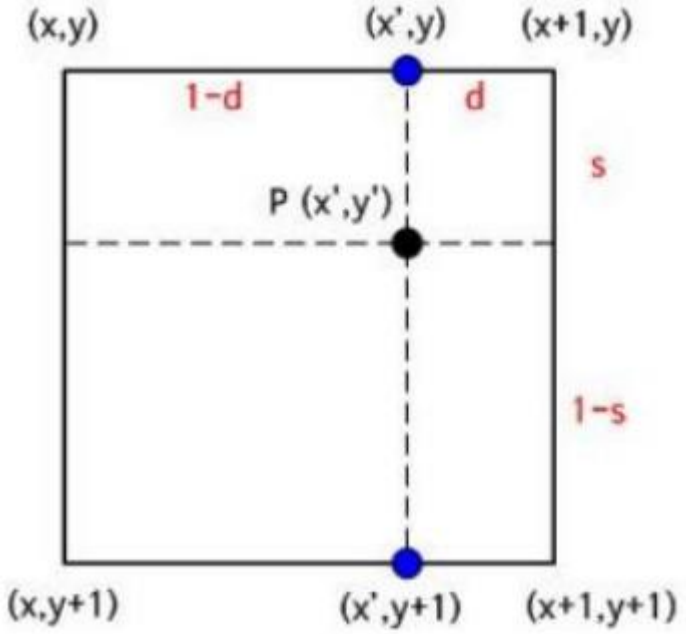
10	50/3	70/3	30



$$F = d \cdot f(x_2) + (1-d) \cdot f(x_1)$$

3.2 보간(interpolation)

- 2차원 선형 보간법
  - 2차원 영상에서 양방향 선형 보간법을 적용하는 경우 1개의 점은 인접한 4개의 점으로부터 영향



$$f(x',y) = d \cdot f(x,y) + (1-d) \cdot f(x+1,y)$$

$$f(x',y+1) = d \cdot f(x,y+1) + (1-d) \cdot f(x+1,y+1)$$

$$f(x',y') = s \cdot f(x',y+1) + (1-s) \cdot f(x',y)$$

3.2 보간(interpolation)

- 최소 인접 이웃 보간법 및 양방향 선형 보간법
  - 최소 인접 이웃 보간법: NN 보간법
  - 양방향 선형 보간법: Bilinear 보간법
    - 새로운 색상 정보 생성/경계 Blur 처리

원본 영상

Nearest Neighborhood Interpolation

Bilinear Interpolation

3.3 기본 변환

RST 변환

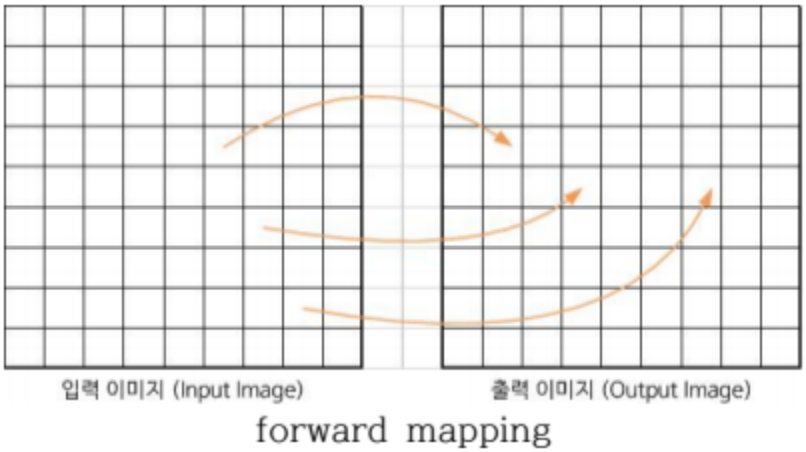
- 회전(Rotation), 크기조정(Scaling), 이동(Translation) 의미하는 대표적인 기하학적인 변환
  - 행렬식으로 표현
  - 기하학적 변환에 의한 좌표값
  - 계산은 행렬 연산 수행

이동 변환 T	$P'(x,y) = T \cdot P(x,y)$ $T = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$	x축으로 t_x, y축으로 t_y 이동
회전 변환 R	$P'(x,y) = R \cdot P(x,y)$ $R = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$	원점을 중심으로 시계방향으로 θ도 회전
크기 변환 S	$P'(x,y) = S \cdot P(x,y)$ $S = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$	x축으로 s_x, y축으로 s_y 크기 조정

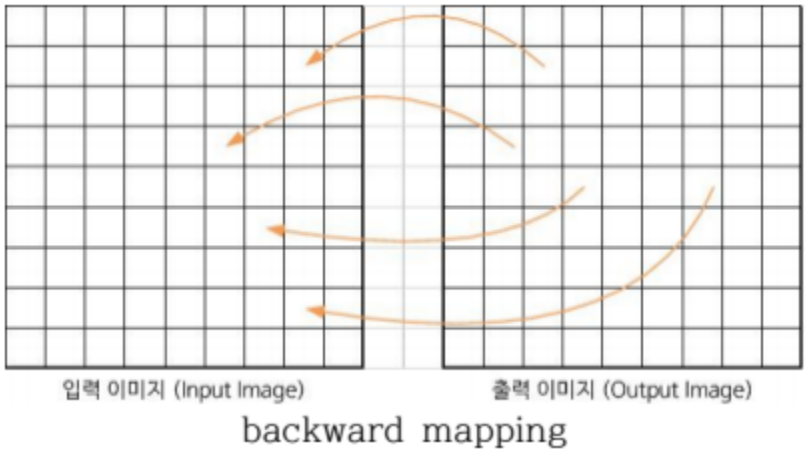
3.3 기본 변환

전방향 매핑 및 역방향 매핑

- 기하학적 변환의 과정은 전방향 매핑(forward mapping)과 역방향 매핑(backward mapping)이 있다.
  - 전방향 매핑은 입력 영상의 픽셀을 출력 영상의 픽셀로 변환하여 채우는 것
  - 역방향 매핑은 출력 영상의 픽셀이 입력 영상의 어떤 픽셀에서 왔는지 계산해서 채우는 방법



forward mapping



backward mapping

※ 전방향 매핑의 과정에서 출력 영상의 픽셀의 값이 비어있는 홀(hole)이 생길 수 있기 때문에 일반적으로 역방향 매핑을 활용

3.3 기본 변환 - scaling

- 크기 변환 – 확대 및 축소
  - 영상의 크기를 변경하는 처리
  - OpenCV에서는 cv2.resize() 함수를 사용하여 크기를 변환
  - 픽셀값 보간 과정에서는 최소 이웃 보간법(nearest neighbor interpolation), 양방향 선형 보간법(bilinear interpolation), Cubic 보간법(cubic interpolation) 방법들을 많이 사용

크기 변환 S	$P'(x,y) = S \cdot P(x,y)$ $R = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$	x축으로 s_x, y축으로 s_y 크기 조정

3.3 기본 변환 - scaling

- 영상 크기 변환 함수: resize() 함수
  - 입력된 영상 데이터에 대하여 크기를 변환

함수명	cv2.resize(src, dsize, fx, fy, interpolation)
매개변수	- src (numpy.ndarray) 변환시킬 영상 데이터 - dsize 가로, 세로 형태의 튜플 ex) (100,200) - fx 가로 크기의 배수, 2배로 크게 하려면 2, 반으로 줄이려면 0.5 - fy 세로 크기의 배수, 2배로 크게 하려면 2, 반으로 줄이려면 0.5 - interpolation 보간법 방식 cv2.INTER_NEAREST: 최소 근접 보간 cv2.INTER_LINEAR: 양방향 선형 보간 cv2.INTER_CUBIC: 양방향 큐빅 보간 cv2.INTER_AREA: 픽셀 리샘플링 보간 cv2.INTER_LANCZOS4: Lanczos 보간 cv2.INTER_LINEAR_EXACT: Bit exact bilinear interpolation cv2.INTER_NEAREST_EXACT: Bit exact nearest neighbor interpolation
리턴값	출력 영상 데이터 (numpy.ndarray)

3.3 기본 변환 - scaling

(ex3-1)

크기 변환

- 512x512 크기의 컬러 영상에 대하여 64x64 크기로 축소한 후에 다양한 보간 방법으로 확대하여 출력

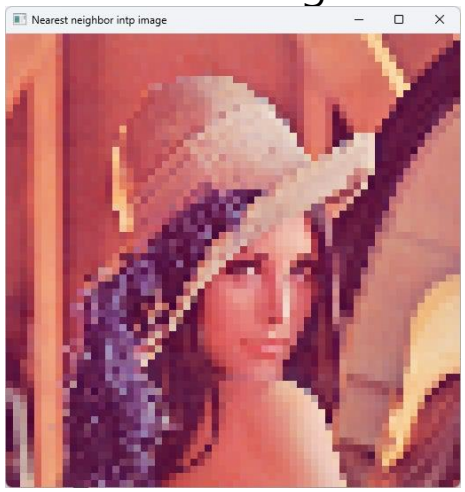
```
import cv2

if __name__ == '__main__':
    ori_img = cv2.imread("./images/Lenna.jpg", cv2.IMREAD_UNCHANGED)
    cv2.imshow('Original image', ori_img)

    rs_img = cv2.resize(ori_img, (64, 64), interpolation=cv2.INTER_LINEAR)
    cv2.imshow('128x64 image', rs_img)

    z_img1 = cv2.resize(rs_img, (512, 512), interpolation=cv2.INTER_NEAREST)
    z_img2 = cv2.resize(rs_img, None, fx=8, fy=8, interpolation=cv2.INTER_LINEAR)
    z_img3 = cv2.resize(rs_img, None, fx=8, fy=8, interpolation=cv2.INTER_CUBIC)
    z_img4 = cv2.resize(rs_img, None, fx=8, fy=8, interpolation=cv2.INTER_AREA)

    cv2.imshow('Nearest neighbor intp image', z_img1)
    cv2.imshow('Bilinear intp image', z_img2)
    cv2.imshow('Cubic intp image', z_img3)
    cv2.imshow('Area intp image', z_img4)

    cv2.waitKey(0)
```

3.3 기본 변환 - scaling

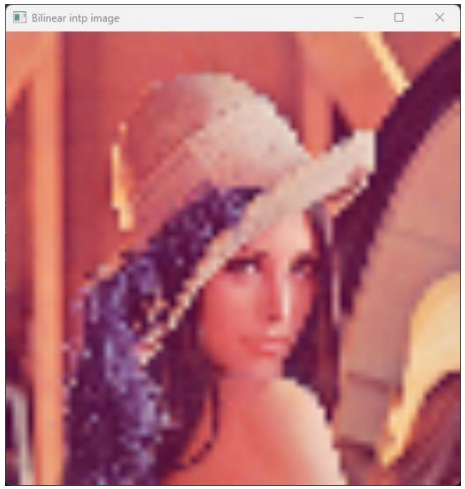
크기 변환 실행 결과

- 최소 근접(Nearest Neighbor) 보간: 경계가 선명하지만 픽셀 자체가 커지기 때문에 경계에서의 정밀도는 떨어짐
- 양방향 선형(Bilinear) 보간: 밝기 값이 자연스럽게 변하는 효과가 있으나 경계가 뭉개지는 현상을 보임
- 큐빅(Cubic) 보간: 경계가 Bilinear 방식 보다는 조금 적게 뭉개지는 효과가 있음

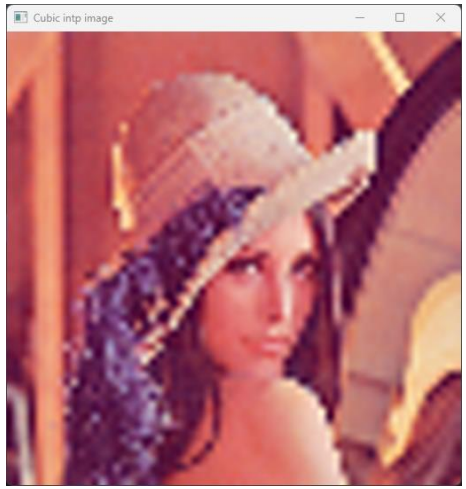
Nearest Neigbor



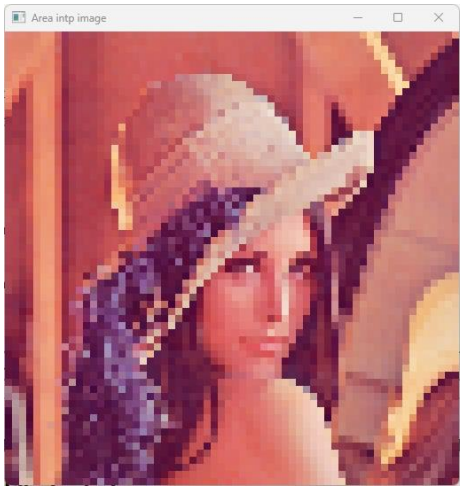
Bilinear



Cubic



Area



3.3 기본 변환 - Translation

- 이동 변환(Translation)
  - 영상을 이동시키는 처리
  - 정수형 이동시에는 보간이 불필요함 (실수형 이동시에 필요)
  - OpenCV에서는 cv2.warpAffine() 함수를 사용하여 이동 변환

이동 변환 T	$P'(x,y) = T \cdot P(x,y)$ $T = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$	x축으로 t_x, y축으로 t_y 이동

3.3 기본 변환 - Translation

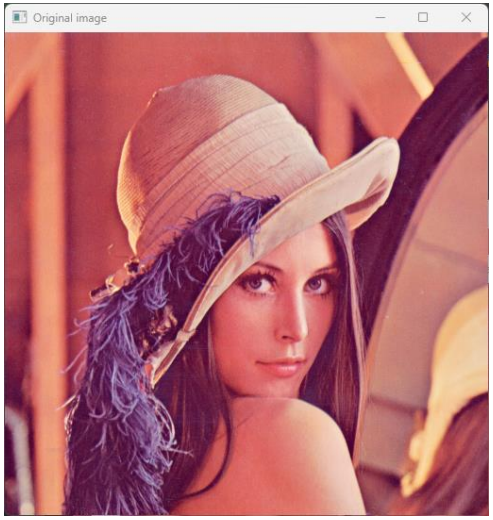
(ex3-2)

- 512x512 크기 컬러 영상에 대하여 이동 변환을 수행

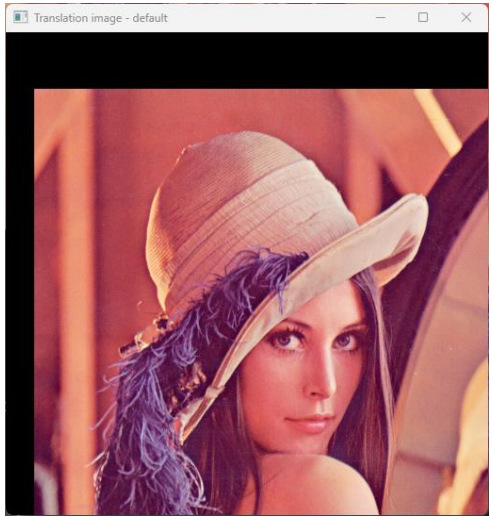
```
import numpy as np
import cv2
if __name__ == '__main__':
    ori_img = cv2.imread("./images/Lenna.jpg", cv2.IMREAD_UNCHANGED)
    cv2.imshow('Original image', ori_img)
    rows, cols = ori_img.shape[:2] # channel 여부 무시
    Mat = np.float32([[1, 0, 30],
                      [0, 1, 60]])
    t_image1 = cv2.warpAffine(ori_img, Mat, (cols, rows))
    t_image2 = cv2.warpAffine(ori_img, Mat, (cols, rows),
                              borderMode=cv2.BORDER_CONSTANT,
                              borderValue=(255,255,255))
    t_image3 = cv2.warpAffine(ori_img, Mat, (cols, rows),
                              borderMode=cv2.BORDER_REPLICATE)
    t_image4 = cv2.warpAffine(ori_img, Mat, (cols, rows),
                              borderMode=cv2.BORDER_REFLECT)
    t_image5 = cv2.warpAffine(ori_img, Mat, (cols, rows),
                              borderMode=cv2.BORDER_WRAP)

    cv2.imshow('Translation image - default', t_image1)
    cv2.imshow('Translation image - BORDER_CONSTANT', t_image2)
    cv2.imshow('Translation image - BORDER_REPLICATE', t_image3)
    cv2.imshow('Translation image - BORDER_REFLECT', t_image4)
    cv2.imshow('Translation image - BORDER_WRAP', t_image5)
    cv2.waitKey(0)
```

3.3 기본 변환 - Translation

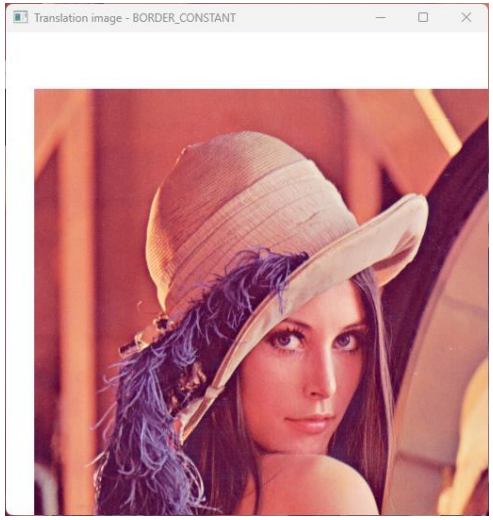
- 실행결과



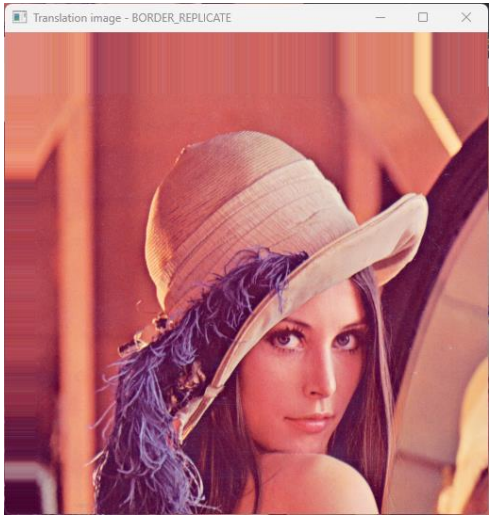
original



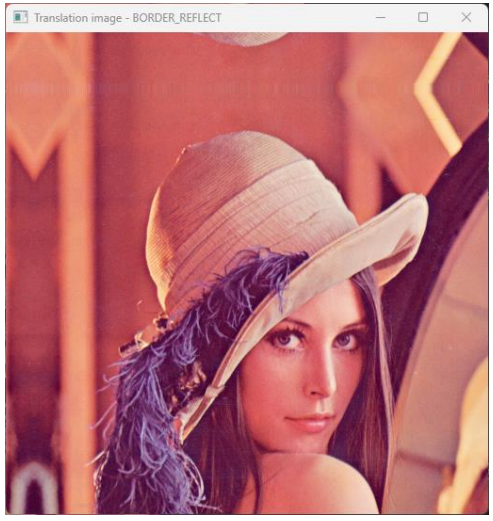
default



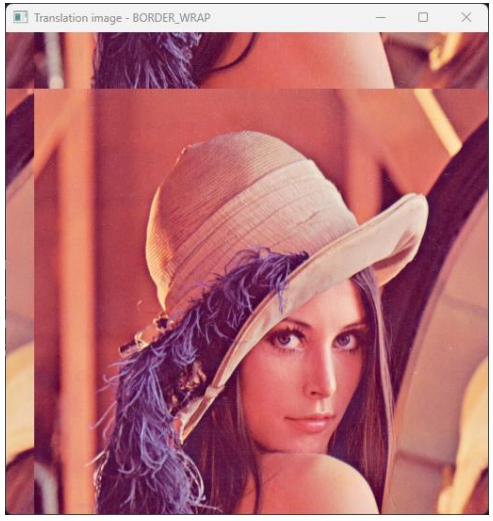
boder_constant



replicate



reflect



wrap

3.3 기본 변환 - Rotation

- 영상 어파인(Affine) 변환 함수: warpAffine() 함수
  - 입력된 영상 데이터에 대하여 지정한 어파인 변환 행렬에 따른 변환을 수행
  - 변환 행렬이 이동 행렬일 경우 이동 변환을 수행

함수명	cv2.warpAffine(src, dsize, fx, fy, interpolation)
매개변수	- src (numpy.ndarray) 변환시킬 입력영상 데이터 - M (2x3) 변환 행렬 - dsize 출력 영상 데이터 크기 - dst 출력 영상 데이터 (dsize 크기로 조정됨) - flags 보간 방법과 역변환 방법 설정 - borderMode 픽셀 외삽(extracpolation) 방법 BorderTypes - borderValue BORDER_CONSTANT 경우 사용할 상수값 (기본값은 0)
리턴값	출력 영상 데이터 (numpy.ndarray)

3.3 기본 변환 - Rotation

- 픽셀 외삽(extracpolation)
  - borderMode

borderMode	- cv2.BORDER_CONSTANT iiiiii|abcdefgh|iiiiii 여기서 i는 지정한 borderValue - cv2.BORDER_REPLICATE aaaaaa|abcdefgh|hhhhhh 경계 복사 - cv2.BORDER_REFLECT fedcba|abcdefgh|hgfedcb 경계부터 반사 - cv2.BORDER_WRAP cdefgh|abcdefgh|abcdefg 소실된 영역을 이동 변환

3.3 기본 변환 - Rotation

- 회전 변환 (Rotation)
  - 영상을 회전시키는 처리
  - OpenCV에서는 cv2.warpAffine() 함수를 사용하여 회전 변환

회전 변환 R	$P'(x,y) = R \cdot P(x,y)$ $R = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$	원점을 중심으로 시계방향으로 θ도 회전

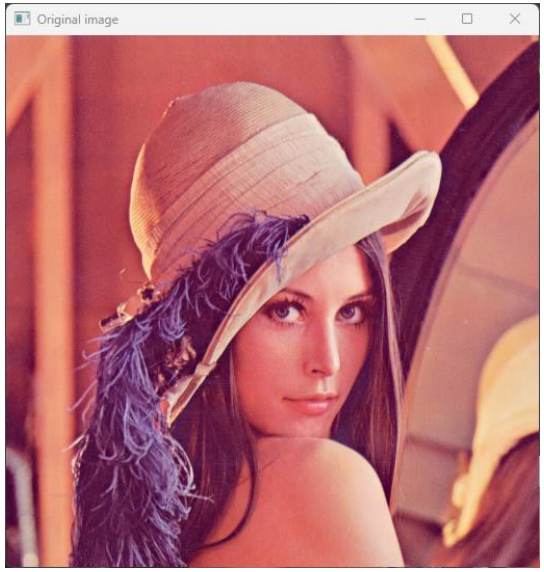
3.3 기본 변환 - Rotation

- 512x512 크기 컬러 영상에 대하여 다양한 회전 변환을 수행하는 코드 (ex3-3)

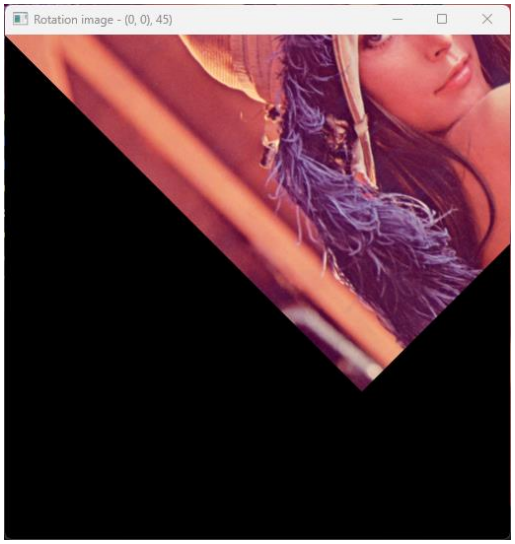
```
import cv2

if __name__ == '__main__':
    ori_img = cv2.imread("./images/Lenna.jpg", cv2.IMREAD_UNCHANGED)
    cv2.imshow('Original image', ori_img)

    rows, cols = ori_img.shape[:2] # channel 여부 무시
    Mat1 = cv2.getRotationMatrix2D((0, 0), 45, 1.0)
    Mat2 = cv2.getRotationMatrix2D((cols / 2, rows / 2), 45, 1.0)
    Mat3 = cv2.getRotationMatrix2D((cols / 2, rows / 2), 90, 1.0)

    r_image1 = cv2.warpAffine(ori_img, Mat1, (cols, rows))
    r_image2 = cv2.warpAffine(ori_img, Mat2, (cols, rows),
                              borderMode=cv2.BORDER_REPLICATE)
    r_image3 = cv2.warpAffine(ori_img, Mat2, (cols, rows),
                              borderMode=cv2.BORDER_DEFAULT)
    r_image4 = cv2.warpAffine(ori_img, Mat3, (cols, rows))

    cv2.imshow('Rotation image - (0, 0), 45)', r_image1)
    cv2.imshow('Rotation image - (w/2, h/2), 45 - replicate)', r_image2)
    cv2.imshow('Rotation image - (w/2, h/2), 45 - default)', r_image3)
    cv2.imshow('Rotation image - (w/2, h/2), 90', r_image4)

    cv2.waitKey(0)
```

3.3 기본 변환 - Rotation

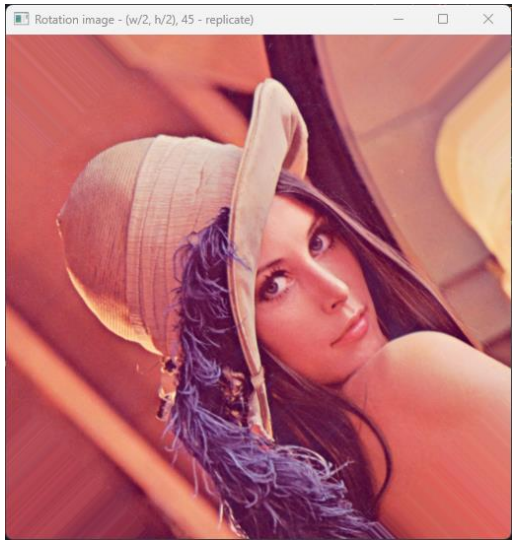
- 실행결과



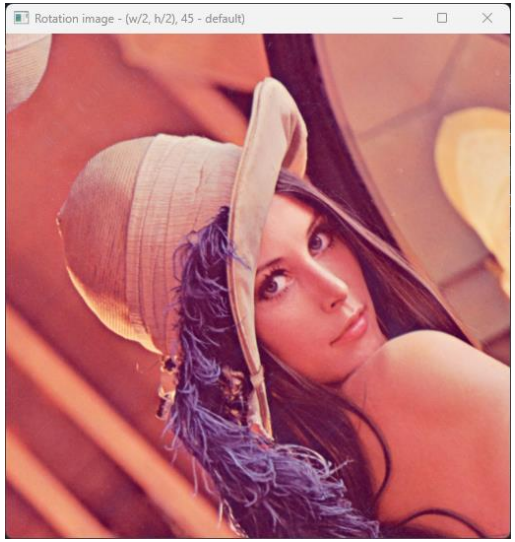
original



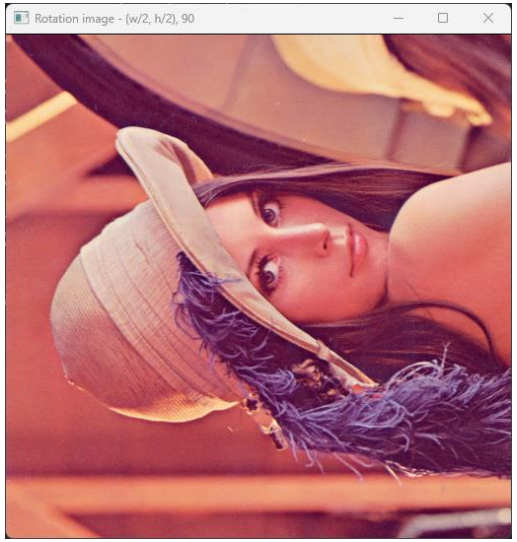
(0,0) 45˚



(w/2,h/20) 45˚ replicate



(w/2,h/20) 45˚ default



(w/2,h/20) 90˚

3.3 기본 변환 - Flip

- 대칭(flip) 변환 또는 반사(reflection) 변환
  - 기하학적으로 반사(reflection) 변환 형태로 특정한 축 기준의 대칭 변환을 수행
  - OpenCV에서는 cv2.flip() 함수를 사용하여 대칭 변환을 수행
- 영상 대칭 변환 함수: flip() 함수
  - 입력된 영상 데이터에 대하여 대칭 변환을 수행

$$
\begin{aligned}
&dst(x,y) = src(rows-i-1,\ j) && \text{if flilpCode} = 0\\
&dst(x,y) = src(i,\ cols-j-1) && \text{if filpCode} > 0\\
&dst(x,y) = src(rows-i-1\ cols-j-1) && \text{if filpCode} < 0
\end{aligned}
$$

함수명	cv2.flip(src, flipCode)
매개변수	- src (numpy.ndarray) 변환시킬 입력영상 데이터 - flipCode 플립 방법을 지정하는 코드 x축 대칭 = 0 y축 대칭 = 양수 - x축 및 y축 대칭 = 음수
리턴값	출력 영상 데이터 (numpy.ndarray)

3.3 기본 변환 - Flip

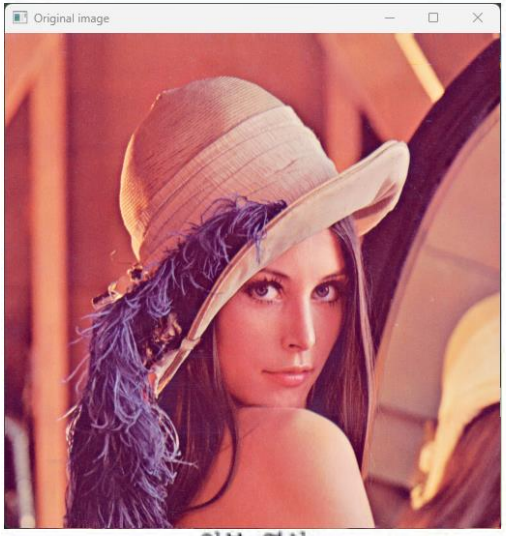
(ex3-4)

- 512x512 크기 컬러 영상을 대칭 변환하는 코드

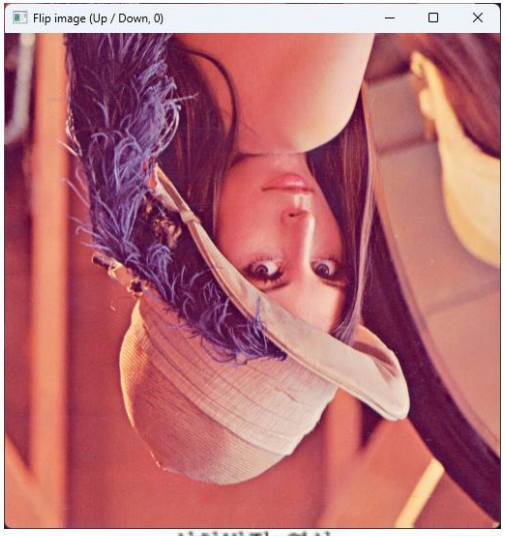
```
import cv2

if __name__ == '__main__':
    ori_img = cv2.imread("./images/Lenna.jpg", cv2.IMREAD_UNCHANGED)

    f_image0 = cv2.flip(ori_img, 0)
    f_image_p1 = cv2.flip(ori_img, 1)
    f_image_m1 = cv2.flip(ori_img, -1)

    cv2.imshow('Original image', ori_img)
    cv2.imshow('Flip image (Up / Down, 0)', f_image0)
    cv2.imshow('Flip image (Left / Right, 1)', f_image_p1)
    cv2.imshow('Flip image (Y=X, -1)', f_image_m1)

    cv2.waitKey(0)
```

3.3 기본 변환 - Flip

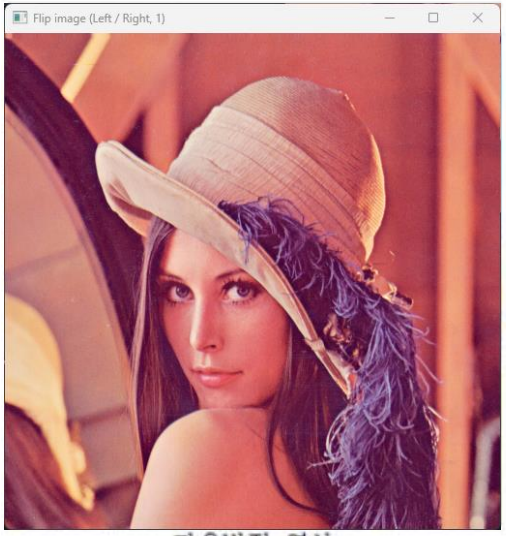
- 실행 결과



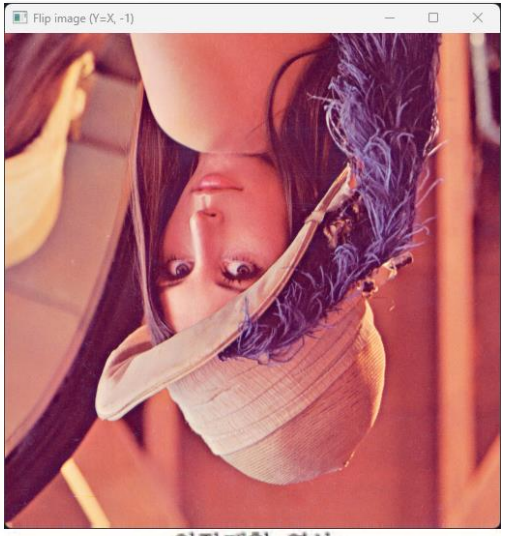
원본 영상



상하반전 영상



좌우반전 영상



원점대칭 영상

[Pyside6] 콤보박스(Combobox)

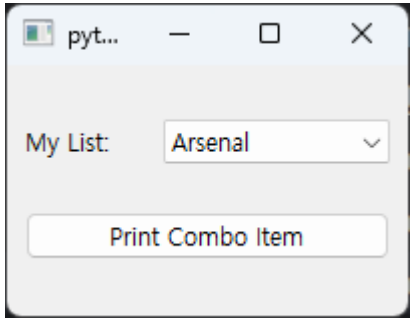
QComboBox

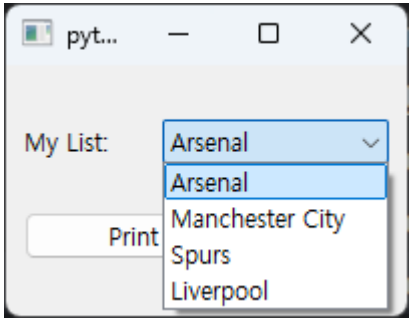
- 콤보 박스는 현재 항목을 표시하고 클릭하면 선택 가능한 항목 목록을 팝업으로 표시하는 선택 위젯
- addItem 매서드를 이용하여 ComboBox 리스트에 표시할 아이템을 추가

```
self.my_combo_box = QComboBox()
self.my_combo_box.addItem("Arsenal")
self.my_combo_box.addItem("Manchester City")
```

- currentText() 매서드: 현재 콤보박스에 선택된 문자열 불러오기
- currentIndex() 매서드: 현재 콤보박스에 선택된 List의 인덱스 번호를 불러오기

```
self.my_combo_box.currentText()
self.my_combo_box.currentIndex()
```





[Pyside6] 콤보박스(Combobox) 만들기

(ex3-5)

```
import sys
from PySide6.QtWidgets import (QApplication, QComboBox, QHBoxLayout,
                               QLabel, QMainWindow, QPushButton, QVBoxLayout, QWidget)
class Window(QMainWindow):
    def __init__(self):
        super().__init__()
        # Title and dimensions
        self.button = QPushButton("Print Combo Item")
        self.my_combo_box = QComboBox()
        self.my_combo_box.addItem("Arsenal")
        self.my_combo_box.addItem("Manchester City")
        self.my_combo_box.addItem("Spurs")
        self.my_combo_box.addItem("Liverpool")

        self.button.clicked.connect(self.print_item)
        self.combo_label = QLabel("My List:")

        h_layout = QHBoxLayout()
        h_layout.addWidget(self.combo_label)
        h_layout.addWidget(self.my_combo_box)
        # 다음 슬라이드를 보고 이어서 작성하세요.
```

[Pyside6] 콤보박스(Combobox) 만들기

(ex3-5)

```
        h_layout.addWidget(self.my_combo_box)
        # 다음 슬라이드를 보고 이어서 작성하세요.
        v_layout = QVBoxLayout()
        v_layout.addLayout(h_layout)
        v_layout.addWidget(self.button)

        # Central widget
        widget = QWidget(self)
        widget.setLayout(v_layout)
        self.setCentralWidget(widget)

    def print_item(self):
        print(f"currentText: {self.my_combo_box.currentText()}")
        print(f"currentIndex: {self.my_combo_box.currentIndex()}")

if __name__ == "__main__":
    app = QApplication()
    w = Window()
    w.show()
    sys.exit(app.exec())
```