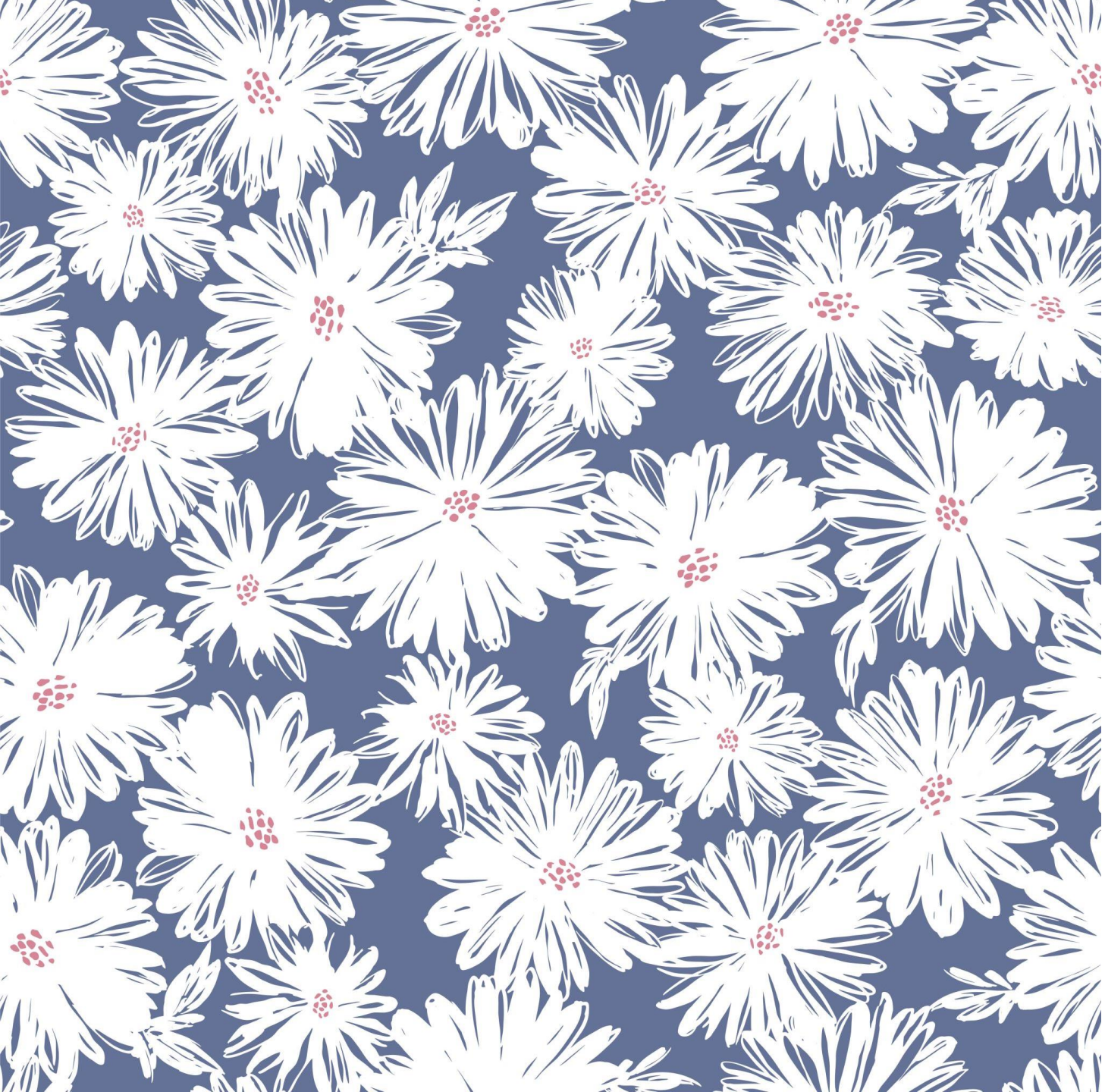
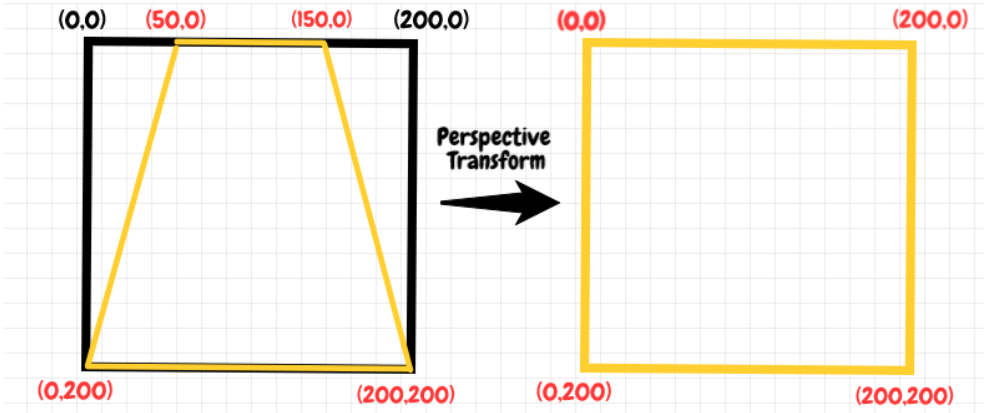


영상처리

기하 변환
perspective transform(원근변환)

원근 변환 개념

- [개념]원근 변환이란 보는 사람의 시각에 따라 같은 물체도 먼 것은 작게, 가까운 것은 크게 보이는 현상인 원근감을 주는 변환임.
  - 이 원근 변환을 통해서 영상이 투영되는 면을 변화 시킬 수 있음.
- [용도] 그림과 같이, 영상내에 있는 물체가 영상의 수평축 기준으로 틀어진 경우 이를 축에 맞도록 변형 할 수 있음.
  - Perspective Transform is a feature that is very useful if you want to align the image properly.
  - It transform the image in a straight manner after Perspective Transformation is applied to it.



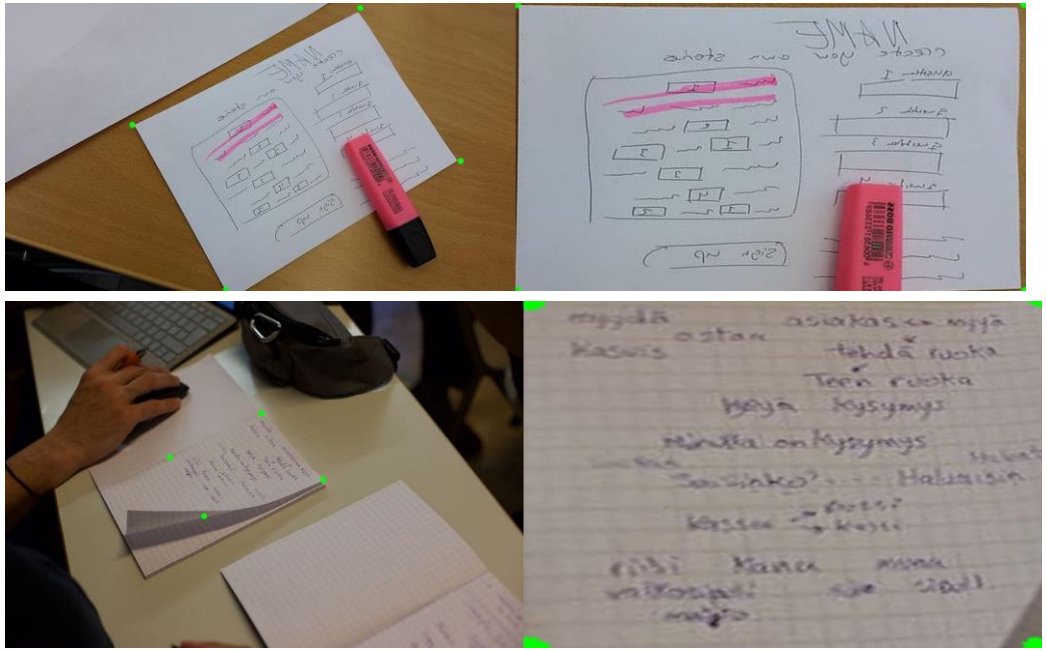


그림.

원근 변환의 작동 방식

- [핵심 아이디어] 영상내 점들을 선형변환 해주는 변환행렬(“행렬은 선형 변환이다.”) 을 구하는 것으로 문제를 정의 할 수 있음.

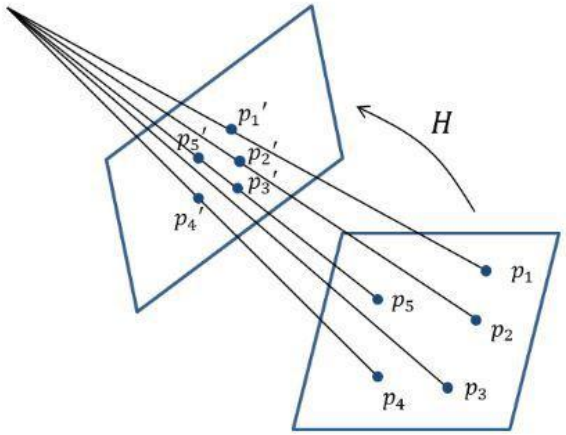


그림. Homography.

Perspective transform 의 일반 식은 homography 와 같으며 이는 homogeneous 좌표계에서 정의되어 그 일반식은 아래와 같음.

$$w\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homogeneous 좌표계에서 정의됨.

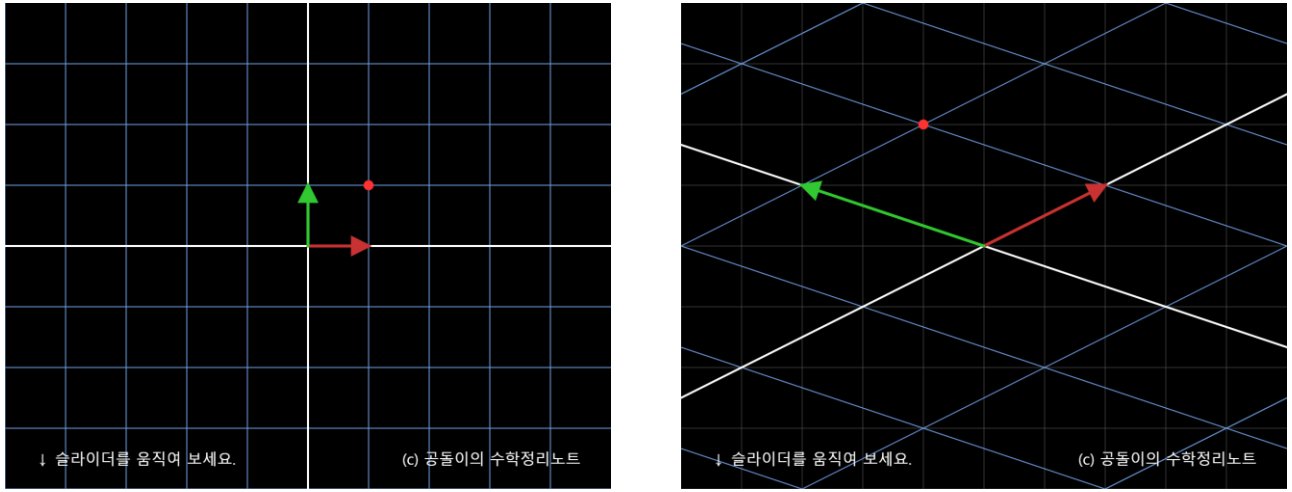


그림. $A = \begin{bmatrix} 2 & -3 \\ 1 & 1 \end{bmatrix}$ 에 대한 벡터 공간의 변환 결과.

- 알고자 하는 변수 개수 8 개. => 수식이 8개 필요.
  - 따라서, 변환 전과 후를 짝짓는 4개(4개+4개=8개)의 매핑 좌표만 지정해 주면 원근 변환에 필요한 3x3 변환행렬에 필요한 값을 찾을 수 있음.

https://darkpgmr.tistory.com/79
https://b.mytears.org/2007/10/599/
https://ballentain.tistory.com/40

원근 변환의 작동 방식

- 구체적으로, 변환행렬은 변수는 다음과 같이 구할 수 있음.

$$w\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Perspective transform 을 위한 식.

homogenious coordinate 를 사용하고 있으니 x' 와 y' 에 관한 식은 아래와 같이 바꿔 쓸 수 있음.

$$x' = \frac{ax+by+c}{gx+hy+1}, \quad y' = \frac{dx+ey+f}{gx+hy+1}$$

$$x' = ax+by+c-gxx'-hx'y$$
$$y' = dx+ey+f-gxy'-hyy'$$

$$\begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -x_1'y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -x_2'y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -x_3'y_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -x_4'y_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' \end{bmatrix}\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix}$$

- 8x8 matrix 의 inverse matrix 를 구한 뒤 이를 양변에 곱해 주면 우리가 값을 알고 싶은 변수들은 a, b, c, d, e, f, g, h 을 구할 수 있음.
  - matrix multiplication 의 경우 서로 곱할 수 있는 형식인지를 체크한 뒤 단순한 계산을 하면 되고, inverse 는 gauss elimination 을 이용 reduced row echelon form 으로 만들어주는 것을 통해 쉽게 구해낼 수 있음.

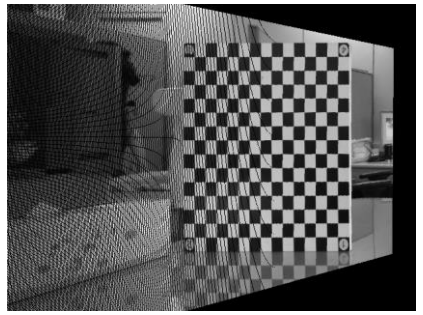
https://b.mytears.org/2007/10/599/

원근 변환의 작동 방식

- 위의 두 가지까지 구현했다면, 이제 warping 만을 구현하면 되겠습니다. 이 warping 은 크게 두가지 방법을 통해 구현할 수 있음.

a. forward mapping
   - 변환 전 좌표에 영상에 대하여 변환 후 좌표의 좌표 값을 계산한 뒤 값을 채워주는 방식.
   - 변환 전 영상보다 변환 후 영상의 크기가 커질 경우 채워줘야 하는 픽셀 좌표의 개수가 증가하기 때문에 forward mapping 방식을 활용하면 결과 영상에 구멍이 생기는 문제 발생. (one to many)

```
for( y = 0 ; y < height ; y++ ){
    for( x = 0 ; x < width ; x++ ){
        x' = (ax+by+c) / (gx+hy+1);
        y' = (dx+ey+f) / (gx+hy+1);
        dst[y'][x'] = src[y][x];
    }
}
```



의사 코드 및 결과 영상.

b. backward mapping
   a. Forward warping 에서 변환 전 좌표를 기준으로 변환 후 좌표를 계산했다면, backward warping 방식에서는 변환 후 좌표를 기준으로 변환 전 좌표를 계산함.
   b. 따라서, 변환 후 모든 좌표계가 변환 전 좌표계에 대응되기 때문에 구멍이 발생하지 않고 변환 후 좌표계에서도 모든 픽셀 값들을 변환 전 좌표계에서 구할 수 있음. (many to one)

```
for( y' = 0 ; y' < height ; y'++ ){
    for( x' = 0 ; x' < width ; x'++ ){
        x = (ax'+by'+c) / (gx'+hy'+1);
        y = (dx'+ey'+f) / (gx'+hy'+1);
        dst[y'][x'] = src[y][x];
    }
}
```

의사 코드 및 결과 영상.

https://b.mytears.org/2007/10/599/

원근 변환의 작동 방식

- [예시]

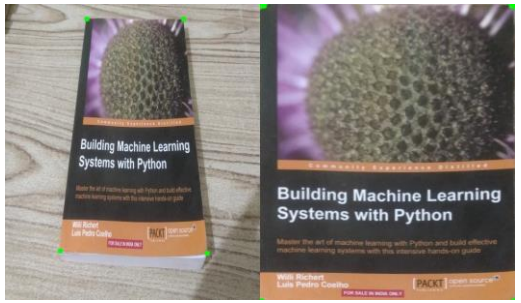
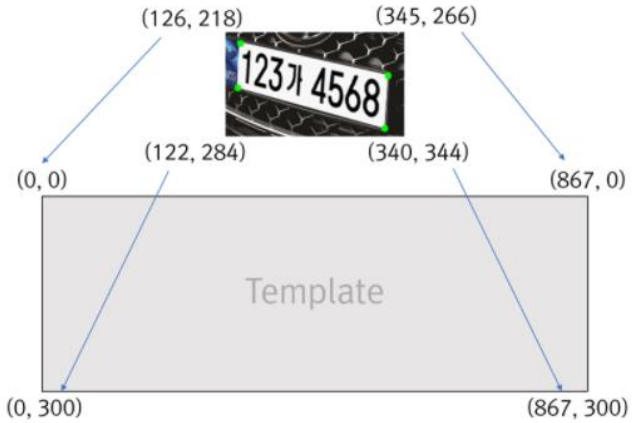


그림. 변환 전 좌표와 그에 대응되는 4개의 좌표쌍이 주어진 경우.(green point)




```
original points :
 [[126. 218.]
 [345. 266.]
 [340. 344.]
 [122. 284.]]

destination points :
 [[  0.   0.]
 [867.   0.]
 [867. 300.]
 [  0. 300.]]
```

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} 126 \\ 218 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad \begin{aligned} 126a + 218b + c &= 0 \\ 126d + 218e + f &= 0 \\ 126g + 218h + 1 &= 1 \end{aligned}$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} 345 \\ 266 \\ 1 \end{bmatrix} = \begin{bmatrix} 867 \\ 0 \\ 1 \end{bmatrix} \qquad \begin{aligned} 345a + 266b + c &= 867 \\ 345d + 266e + f &= 0 \\ 345g + 266h + 1 &= 1 \end{aligned}$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} 340 \\ 344 \\ 1 \end{bmatrix} = \begin{bmatrix} 867 \\ 300 \\ 1 \end{bmatrix} \qquad \begin{aligned} 340a + 344b + c &= 867 \\ 340d + 344e + f &= 300 \\ 340g + 344h + 1 &= 1 \end{aligned}$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} 122 \\ 284 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 300 \\ 1 \end{bmatrix} \qquad \begin{aligned} 122a + 284b + c &= 0 \\ 122d + 284e + f &= 300 \\ 122g + 284h + 1 &= 1 \end{aligned}$$

- 변환 전 좌표와 그에 대응되는 4개의 좌표쌍을 입력

- 3x3 변환행렬에 대하여 변수 8개가 주어지며 입력된 좌표계에 의하여 수식 8개가 도출되는 것을 확인.

https://minimin2.tistory.com/135
https://snakehips.tistory.com/entry/Python-06%EC%A3%BC%EC%B0%A8-%EA%B3%BC%EC%A0%9C-Perspective-transform

원근 변환의 작동 방식

- [코드구현]
  - OpenCV library
    1. cv2.getPerspectiveTransform() 을 통해서 3x3 변환행렬을 계산.
    2. 1. 에서 구해진 투영 행렬을 cv2.warpPerspective() 을 통해서 원근 변환을 적용하여 결과 영상 출력.

```python
# 변환 전 4개 좌표
pts1 = np.float32([topLeft, topRight, bottomRight, bottomLeft])

# 변환 후 영상에 사용할 서류의 폭과 높이 계산
w1 = abs(bottomRight[0] - bottomLeft[0])
w2 = abs(topRight[0] - topLeft[0])
h1 = abs(topRight[1] - bottomRight[1])
h2 = abs(topLeft[1] - bottomLeft[1])
width = max([w1, w2])  # 두 좌우 거리간의 최대값이 서류의 폭
height = max([h1, h2])  # 두 상하 거리간의 최대값이 서류의 높이

# 변환 후 4개 좌표
pts2 = np.float32([[0, 0], [width - 1, 0],
                   [width - 1, height - 1], [0, height - 1]])

# 변환 행렬 계산
mtrx = cv2.getPerspectiveTransform(pts1, pts2)
# 원근 변환 적용
result = cv2.warpPerspective(img, mtrx, (width, height))
cv2.imshow('scanned', result)
```

```python
M = cv2.getPerspectiveTransform(src=src_np, dst=dst_np)
result = cv2.warpPerspective(ori_img, M=M, dsize=(int(width
print("Perspective Transformation :\n", M)
```

```
Perspective Transformation :
[[ 5.62450436e+00  4.32654182e-01 -8.03006161e+02]
[-1.21483331e+00  5.49205892e+00 -1.04419985e+03]
[ 1.05016650e-03  2.93721465e-04  1.00000000e+00]]
```

https://minimin2.tistory.com/135
https://snakehips.tistory.com/entry/Python-06%EC%A3%BC%EC%B0%A8-%EA%B3%BC%EC%A0%9C-Perspective-transform