컴퓨터비전 Quiz 1

학과 : 컴퓨터 학부

학번 : 20192460

이름 : 홍상준

2D 변환

2D 변환 은 두 이미지 사이의 매칭되는 관계를 설명할 수 있다.

- 두 이미지가 서로 형태가 다르더라도,
  매칭되는 쌍의 점들이 있다면 각 쌍의 점들로 선형 변환 식을 유도할 수 있다.
- 선형 변환식만 있다면, 이미지를 잘 **평행 이동**하고, **회전**, **Shear** 등등.. 변환하여
  각 이미지를 잘 정렬할 수 있게 된다.
  왼쪽 눈, 오른쪽 눈, 코의 점이 어떻게 변했는지를 알면 두 점끼리 잘 매칭할 수 있을 것이다.

데카르트 좌표계에서의 선형 변환

- 이 좌표계에서 하나의 변환 행렬만을 사용해서 가능한 변환은
  R(회전), S(확대) 변환 혹은 그 둘을 적절히 섞은 변환이 가능한데
- 단, 불가능한 변환이 있다. 바로 **하나의 변환 행렬로는 평행 이동은 불가능하다.**
- 여기에서는 벡터 덧셈을 추가적으로 수행해 줘야 한다.

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

동차 좌표계

- 점과, 벡터를 하나의 형식으로 표현하기 위해서는 **동차 좌표계**를 사용해야 한다.
- 이를 통해 **평행 이동 변환**도 굳이 벡터 덧셈이 필요 없이 그냥 점의 형태로 **하나의 변환 행렬로 일관되게 사용**할 수 있다.

$$\begin{bmatrix} x_{new} \\ y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & b_1 \\ A_{21} & A_{22} & b_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- 동차 좌표계로 표현된 위 형태에서
  i. $(A_{11}A_{21}, 0)$ 는 벡터 : 마지막 행이 0
  ii. $(A_{21}A_{22}, 0)$ 도 벡터 : 마지막 행이 0
  iii. $(b_1, b_2, 1)$ **는 점! : 마지막 행이 1이면 점** 즉 이 점만큼 **평행 이동했다를 알수 있다.**

최소 3개의 필요한 이유

1). 어파인 행렬을 구성하기 위해서이다.

- **"아핀 변환"에 한에서 3행의 값 $(0, 0, 1)$는 고정** 이기 때문에.
  우리가 알고 싶은 행렬은 사실 다음과 같은 꼴이다.

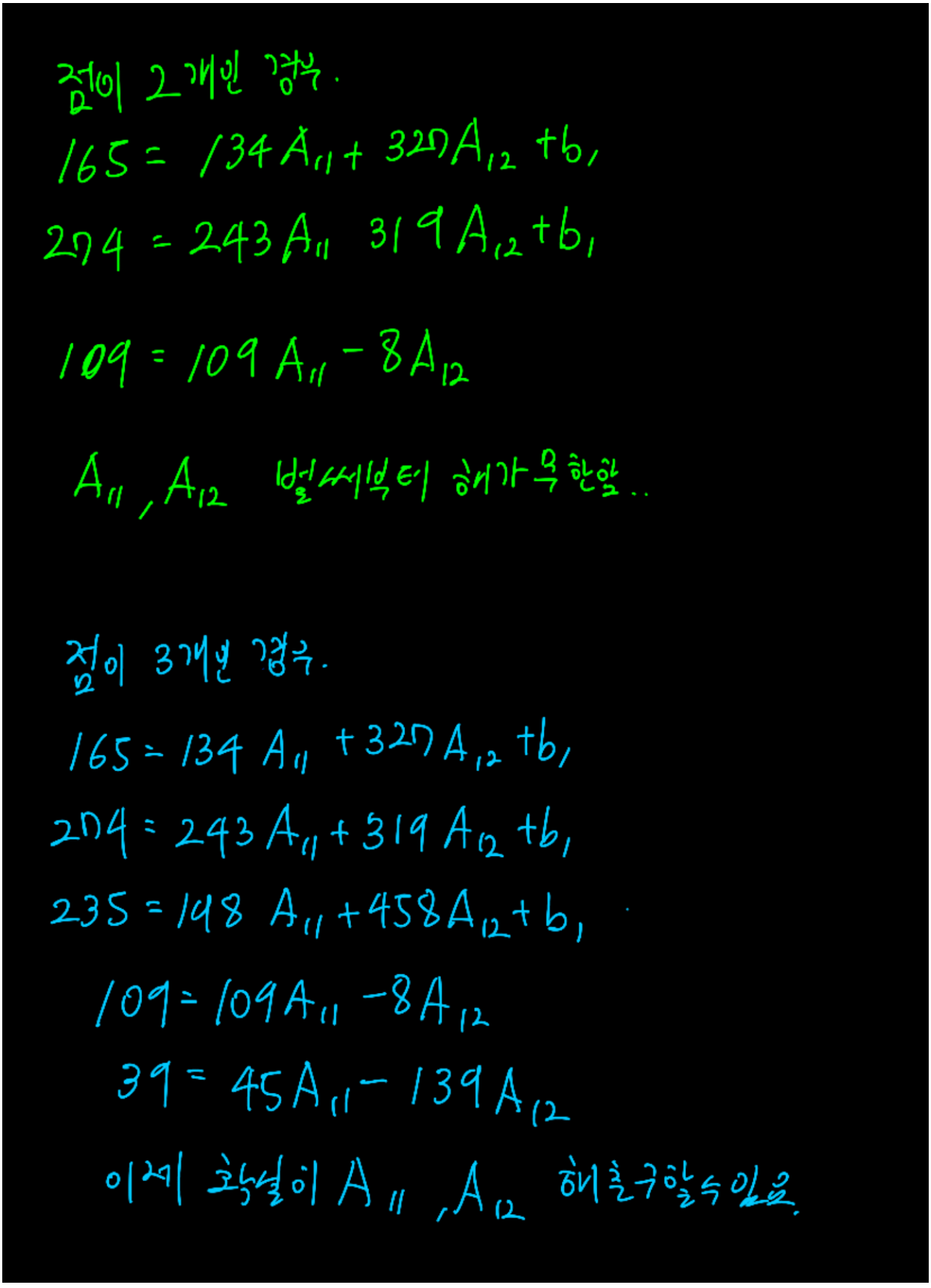
$$T = \begin{bmatrix} A_{11} & A_{12} & b_1 \\ A_{21} & A_{22} & b_2 \end{bmatrix}$$

2). 여기서 알아내야 하는 것들은 A_{11}, A_{12}, A_{21}, A_{22}, b_1, b_2

- 예시로 다음 데이터를 사용한다고 했을 때,

```
"first_image_points": [
  [134,327],
  [243,319],
  [198,458]
],
"second_image_points": [
  [165,320],
  [274,301],
  [235,458]
]
```

1. 만약 점 두 개만 가지고 구해보려고 노력해 봐도. **나머지 2개의 해가 무한한 상태**가 나온다.
2. 하지만, 점이 세 개라면 x_{new}, y_{new} 각각에 대한 3개의 선형 방정식을 만들 수 있고,
   x_{new}의 공통 미지수 3개 A_{11}, A_{12}, b_1 을 구할 수 있게 되었다.
   마찬가지로 y_{new} 또한 같은 방식으로 구해, **총 6개의 미지수를 모두 구함으로 아핀 변환 행열을 유도**할 수 있다



3). "최소 3개??"

- 만약 몇 개의 선형 방정식이 평행이거나 중복이라면 미지수를 확정 지을 수 없게 될 것이다.
  즉 선형 종속인 방정식이 하나라도 껴 있으면 이런 경우 점을 3개보다 더 많이 찍어야 할 수도 있다.

📄 GUI 코드

- 이미지를 클릭하면서 얻어낸 좌표는 각각 다음 멤버에 들어가게 된다.

```
left_image_widget
right_image_widget
```

- 그리고 실제로 저장되는 데이터는 다음과 같고
  각각 `left_image_widget`, `right_image_widget`에 기입될 데이터이다.

```
"first_image_points": [
  [134,327],
  [243,319],
  [198,458]
],
"second_image_points": [
  [165,320],
  [274,301],
  [235,458]
]
```

- 이제 각 포인트 들을 열벡터의 벡터 $left = (\vec{l_1}, \vec{l_2}, \vec{l_3}), right = (\vec{r_1}, \vec{r_2}, \vec{r_3})$ 로 구성한다.

```
/*gui.py*/

def get_mapping(self)
    ...
    left = np.array(
        [[x, y] for y, x in left[:num_points]],
        np.float32
    )
    right = np.array(
        [[x, y] for y, x in right[:num_points]],
        np.float32
    )
```

- OpenCV `getAffineTransform()`를 살펴보자면
  `src`라는 벡터 배열이 `dst`각각의 벡터 배열로
  변환되기 아핀 변환 행렬을 리턴하는 함수라는 것을 알 수 있다.

```
/*cpp 코드*/
Mat cv::getAffineTransform  (
        const Point2f       src[],
        const Point2f       dst[]
)
```

- `return cv2.getAffineTransform(right, left)`
  right 이미지가 left 이미지로 정렬되기 위한 아핀 변환 행렬을 리턴한다.

참고

- https://rfriend.tistory.com/174
- https://angeloyeo.github.io/2024/06/28/Affine_Transformation.html
- https://ballentain.tistory.com/m/38
- https://darkpgmr.tistory.com/79
- https://docs.opencv.org/3.4/d4/d61/tutorial_warp_affine.html
- https://m.blog.naver.com/sees111/222371280008