

컴퓨터비전프로그래밍 및 응용

Term project Report



담당교수	권 남 규 교수님
과목번호	3514
학번	21911944
이름	권 혁 민
제출일	2023.06.07

목차

1. level_A1a

1) [위치 관계 파악을 위한 IDEA 및 구현 방법]

- ① IDEA
- ② 구현 방법
- ③ 핵심부분 코드설명

2) [알고리즘 흐름도]

3) [한계점]

4) [여러 예시에 대한 적용 결과]

2. level_A1b

1) [HW#7에서 추가로 고려해야 할 점에 대해 분석 및 구현 방법]

- ① 분석
- ② 구현 방법
- ③ 코드 추가설명

2) [알고리즘 흐름도]

3) [한계점]

4) [여러 예시에 대한 적용 결과]

3. level_A2

- 1) [영상의 순서를 파악하기 위해 사용한 방법]
- 2) [알고리즘 흐름도]
- 3) [주요코드 설명]
- 4) [여러 예시에 대한 적용 결과]

4. level_B1

- 1) [구현을 위해 생각한 IDEA 및 참고사진 활용방안]
- 2) [주요코드 설명]
- 3) [알고리즘 흐름도]
- 4) [여러 예시에 대한 적용 결과]

5. 결론

Level 1-a

[위치 관계 파악을 위한 IDEA 및 구현 방법]

- IDEA

위치 관계 파악을 하기 위해서는 Stitching 과정에서 도출되는 M 행렬을 이용해야 합니다. 아래는 findHomography 함수를 이용하여 M 행렬을 구하는 코드입니다.

```
④ mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)  
#M, mask = cv2.findHomography(src_pts, dst_pts)
```

<출처 강의자료>

아래의 행렬식에서 표현된 행렬 M은 3by3 행렬로서 a,b,d,e는 scale과 rotation에 대한 성분이고 c,f는 translation에 대한 성분입니다. 따라서 위치 관계를 파악하기 위해서는 c와 f를 자세히 분석해야 합니다.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

M 행렬

<출처 강의자료>

우선 c는 매칭된 feature의 x방향으로의 차, f는 매칭된 feature의 y방향으로의 차를 의미합니다. 따라서 x방향으로의 translation 변화가 크다면 c의 절댓값이 클 것이고, y방향으로의 translation 변화가 크다면 f의 절댓값이 클 것입니다.

따라서 두 image의 좌우를 판단할 때는 행렬 M의 c값으로, 상하를 판단할 때는 행렬 M의 f값으로 해결해야 합니다.

- 구현방법

전체적인 흐름으로는 우선 Threshold를 주어 x,y 방향 각각 translation변화가 의미 있는 수치인지를 파악합니다. 그리고 x방향으로의 c값이 양수이면 img1이 우측, 음수이면 img1이 좌측으로, y방향으로의 f값이 양수이면 img1이 아래, 음수이면 img1이 상 image로 판단합니다.

미리 img1에 대한 방향을 나타내는 변수 answer1 그리고 img2에 대한 방향을 나타내는 변수 answer2를 선언합니다. 그리고 각 조건이 성립되면 그에 대응되는 방향을 변수에 추가되게 하였습니다.

- 핵심부분 코드설명

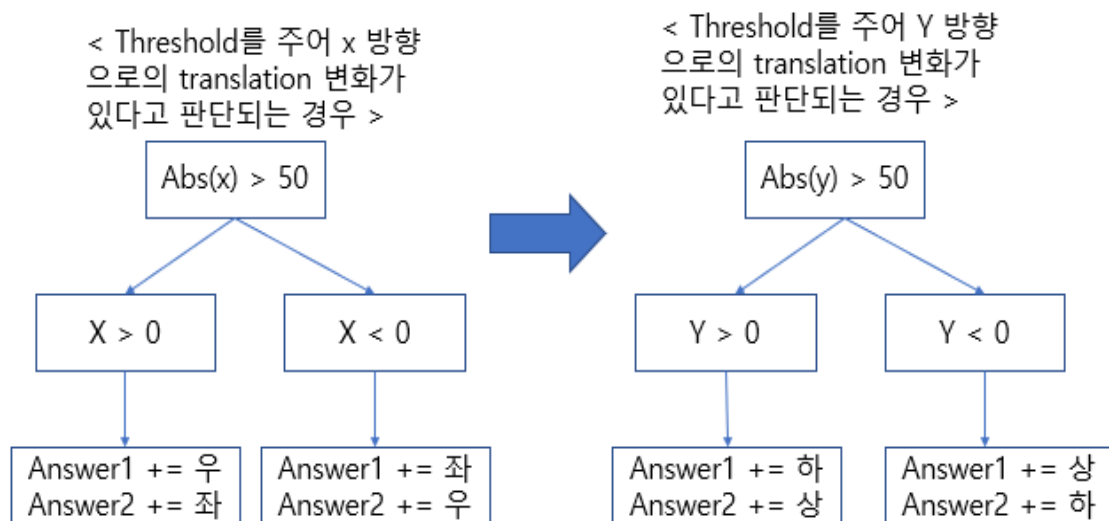
```
66 print(M)
67 answer1= ''
68 answer2= ''
69
70 x=M[0,2]
71 y=M[1,2]
72
73 if abs(x)>50:
74     if x>0:
75         answer1+='우'
76         answer2+='좌'
77     else:
78         answer1+='좌'
79         answer2 += '우'
80 if abs(y) > 50:
81     if y>0:
82         answer1+='하'
83         answer2 += '상'
84     else:
85         answer1+='상'
86         answer2 += '하'
87
88 print('img1 = ',answer1)
89 print('img2 = ',answer2)
```

우선 img1에 대한 방향을 나타내는 변수 answer1 그리고 img2에 대한 방향을 나타내는 변수 answer2를 선언하고, x방향으로의 값을 의미하는 C를 나타내는 M[0,2]를 x로 선언, Y방향으로의 값을 의미하는 F를 나타내는 M[1,2]를 y로 선언합니다.

우선 thresholding을 하여 의미 없는 값들은 버린 후 진행합니다.(자세한 설명은 뒤의 [한계점]에서 설명할 예정입니다.) 선택문(if문)을 이용하여 x,y 각각 방향을 판단합니다. x값이 양수이면 answer1에는 '우' 방향을, answer2에는 '좌'방향을 추가합니다. x값이 음수인 반대의 상황에서는 answer1에는 '좌' 방향을, answer2에는 '우'방향을 추가합니다. 위의 과정을 y 방향으로도 동일하게 수행합니다.

예를들어 x값이 음수라 answer1에 '좌'가 추가되고, y값이 양수라 answer1에 '하'가 추가된다면, img1은 '좌하'가 됩니다.

[알고리즘 흐름도]



중간에 표시된 화살표는 좌측, 우측 그림 둘 다 실행되어야 한다는 의미입니다. (if 문처럼 둘 중에 하나가 실행되어야 한다는 의미가 아닙니다.)

[한계점]

제가 작성한 코드의 한계점이자 가장 고민을 많이 했던 부분은 Threshold을 얼마로 설정하느냐 입니다. 예시로, 1-1과 1-2 image일때 X방향으로의 변화(좌우 판단의 상황)만 있는 경우로서 X방향 변화를 나타내는 c값을 봐야 합니다. 하지만, 이






경우 Y방향 변화를 나타내는 f값이 무조건 0이 되는 것은 아닙니다. 그 이유는 아무리 RANSAC algorithm을 사용할지라도 regression을 사용하는 Feature Matching 과정에서 발생하는 약간의 오차 또는 rotation, scale 등과 같은 image 왜곡 때문이라고 생각합니다. 따라서 교수님께서 제공해주신 sample dataset의 c,f 값의 경우의 수를 표로 파악해보았습니다.

순서	Img1	Img2	C	f
1	1-1	1-2	276	0.08
2	2-1	2-2	285	-4
3	1-1	2-1	-0.07	147
4	1-2	2-2	-4	-140
5	1-1	2-2	-284	-140
6	2-1	1-2	-280	140

X방향으로의 변화(좌우 판단의 상황)만 해야 하는 1번,2번의 경우 f값이 0.08, -4입니다. 또한, Y방향으로의 변화(상하 판단의 상황)만 해야 하는 3번,4번의 경우 C값이 -0.07, -4입니다.

뿐만 아니라 제가 임의의 사진들로 코드를 실행해 본 결과 최소값은 0.08, 최대값은 40입니다. 그래서 저는 Threshold를 50으로 설정하여 절댓값이 50이하인 경우는 버리는 값으로 설정하였으며 threshold를 임의로 설정하였기에 이 때문에 발생하는 오류가 있을 수 있습니다.

[여러 예시에 대한 적용 결과]

Img1	Img2	output
		<pre>img1 = 우 img2 = 좌</pre>
		<pre>img1 = 좌상 img2 = 우하</pre>
		<pre>img1 = 하 img2 = 상</pre>

두 장의 Image에 대한 output(방향성을 나타냄)을 도출하였습니다. 4가지 경우 모두 입력한 순서대로 올바른 방향을 나타내는 것을 볼 수 있습니다. 하지만, 순서를 올바르게 바꾸지 않고 그대로 스티칭한다면 올바른 스티칭 이미지가 나오지 않을 것입니다. 이러한 문제(순서를 올바르게 바꿔서 올바른 stitching해결하기)는 Level 2에서 수행하였습니다.

Level 1-b

[HW#7에서 추가로 고려해야 할 점에 대해 분석 및 이에 대한 구현 방법]

- 분석

우선 가장 큰 차이점은 HW7에서는 단순히 두 장의 이미지를 스티칭 하는 과제였지만, 이번 과제는 사분할인 네 장의 이미지를 스티칭 하는 과제입니다. 이번 1_b 과제에서 주요하게 봐야할 부분은 두가지라고 생각합니다.

첫번째는 '스티칭 순서' 입니다. 이미 이미지의 방향을 찾는 1_a 과제에서 img1에는 우측이미지 그리고 아래 이미지가 들어가야 한다는 것을 알게 되었습니다. 그래서 저는 우선순위로 첫번째로는 우측이미지, 두번째로는 아래이미지로 두었습니다. 이 부분에 대한 추가 설명은 아래의 구현방법 순서1번에 있는 그림과 함께 설명할 예정입니다.

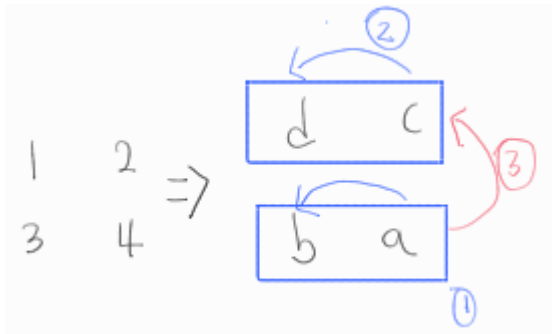
두번째는 '슬라이싱'입니다. 지금까지 생각한 내용으로만 하단부(img12)와 상단부(img34)를 스티칭 한다면 최종 스티칭 결과에서는 아래부분이 검은색 화면으로 나타나게 됩니다. 따라서 상단부 스티칭(img c와d) 과정에서 그냥 보내는 것이 아니라 slicing을 통해서 유효한 정보들만 남기고 최종 스티칭 과정으로 보냈습니다. 이 부분에 대한 추가 설명은 아래의 구현방법 순서3번에 있는 중요 표시 및 코드 설명에서 보충할 예정입니다.

- 상세한 구현방법

전체적인 흐름을 4단계로 나누었고, 각 파트별로 기술하였습니다.

1. image를 입력 받고 순서를 조정합니다. (code 11~40)

- 4개의 image를 각각 입력받아 순서대로 img1,2,3,4로 선언합니다. 이후, img a,b,c,d로 이후의 작업이 편리해지도록 아래의 그림처럼 순서를 변환합니다.



이렇게 변환하는 것이 필수적인 것은 아니지만 이후에 있을 작업이 편리해지도록 하기 위해서 제가 임의로 설정한 것입니다. 우선 stitching을 할때에는 첫번째 image로 우측 또는 아래쪽 image가 오도록 하는 것이 올바른 방법입니다. 따라서 1,2번 처럼

우측 image가 각각 선행 image가 되도록 설정하였고, 3번처럼 아래쪽 image가 선행 image가 되도록 설정하였습니다.

2. 하단부 image (image a와 b) stitching (code 44~111)

- 초반에 있는 코드 44줄부터 97줄까지는 hw7과 동일합니다. 따라서 이 부분에 대한 설명은 뒤쪽에 있는 코드의 주요설명 부분에서 설명하겠습니다.

우선 image a와b를 스티칭하기위해서 우측 image인 a를 b에맞게 변환하여 imga_new로 선언합니다. 이후 새롭게 선언된 imga_new와 imgb를 합성하여 스티칭 결과인 **imgn1**을 생성합니다.

3. 상단부 image (image c와 d) stitching (code 115~193)

- 상단부 image도 코드 115줄부터 169줄까지는 위의 2번과 동일하게 우측, 좌측 image를 스티칭 하는 작업이기에 위 내용과 동일합니다.

우선 image c와d를 스티칭하기위해서 우측 image인 c를 d에맞게 변환하여 imgc_new로 선언합니다. 이후 새롭게 선언된 imgc_new와 imgd를 합성하여 스티칭 결과인 **imgn2**을 생성합니다.

<중요> 물론 다른 코드들도 수정을 약간 하였지만, 제가 생각하는 level1_b 과제의 핵심은 코드 176줄부터 189줄이라고 생각합니다.

우선 if-else문을 사용하여 imgc와 imgd 중 더 작은 height를 height threshold인 h_end로 선언하였습니다. 그리고 imgc_new와 imgd를 합성한 stitching image인 imgn2를 바로 출력시키는 것이 아니라 indexing을 해준 뒤 출력하였습니다. 이때 우선 좌,우측 이미지를 작업하는 중이기 때문에 width는 기존의 코드와 동일하게 두었고, 위에서 선언해두었던 두 이미지의 높이 중 더 낮은 값인 h_end를 이용하여 슬라이싱하였습니다. 아래의 코드설명 부분에서 코드와 함께 추가적인 설명을 할 예정입니다.

4. 상단부, 하단부 image (imgn1와 n2) stitching (code 196~262)

- 상단부, 하단부 image도 코드 196줄부터 244줄까지는 위의 2번과 동일하게 아래, 위 image를 스티칭 하는 작업이기에 위 내용과 동일합니다.

우선 하단부 이미지 스티칭 결과인 image n1와 상단부 이미지 스티칭 결과인 n2를 스티칭하기 위해서 아래 image인 imgn1을 imgn2에 맞게 변환하여 imgn1_new로 선언합니다. 이후 새롭게 선언된 imgn1_new와 imgn2를 합성하여 최종 스티칭 결과인 **imgn3**을 생성합니다.

- 코드 추가설명

```
176 h0,w0,ii=imgc.shape #우선 상단부 이미지 (imgn2)를 구할 때 imgc, imgd, imgc_new의 각 height, width가
177 h1,w1,ii=imgd.shape # 중요시 사용될 것이기에 각 parameter들을 선언하였습니다.
178 h2,w2,ii=dst.shape
179
180 h_end=0
181
182 if h0<=h1:
183     h_end=h0
184 else:
185     h_end = h1
186
187 dst[0:h1,0:w1]=imgd[0:h1,0:w1] # 변환된 두 image을 합성
188
189 dst=dst[0:h_end,0:w2]
190
191 cv2.imshow('stiching 2',dst),plt.show() #출력
192
193 imgn2=dst
```

우선 상단부 스티칭 이미지 (imgn2)를 구할 때 imgc, imgd, imgc_new의 각 height, width가 중요시 사용될 것이기에 각 parameter들을 선언하였습니다. if-else문을 사용하여 imgc와 imgd 중 더 작은 height를 height threshold인 h_end로 선언하였습니다.

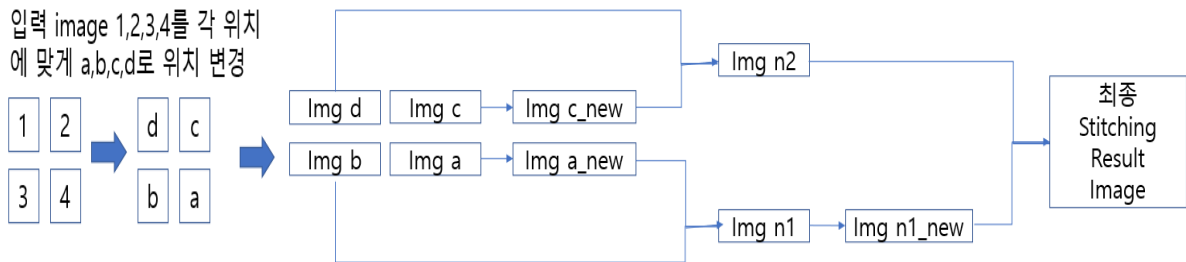
그리고 imgc_new와 imgd를 합성한 stitching image인 imgn2를 바로 출력시키는 것이 아니라 indexing을 해준 뒤 출력하였습니다. 이때 우선 좌, 우측 이미지를 작업하는 중이기 때문에 width는 큰 상관이 없습니다. 따라서 width에 대한 범위는 기존의 코드와 동일하게 두었고, 위에서 선언해두었던 두 이미지의 높이 중 더 낮은 값인 h_end를 이용하여 슬라이싱하였습니다.

처음에는 단순히 검정화면을 제거하고자 imgc의 height인 h0으로 슬라이싱을 해봤지만, 중간 부분에 여전히 검정색 선이 생겼습니다. 그래서 조금 더 고민을 해보니 imgc와 imgd중에 높이가 더 작은 것을 미리 알 수 없기에 if-else문을 사용하여 더 작은 높이인 h_end를 구하고 이 값을 이용해 slicing을 진행하면 된다는 것을 알게 되었습니다.

```
255 #resize사용해서 크기 낮춤
256 scale_down = 0.5
257 scaled_dst = cv2.resize(dst, None, fx=scale_down, fy=scale_down, interpolation=cv2.INTER_LINEAR)
```

최종 스티칭 결과의 크기를 resize함수를 통해서 줄이는 코드입니다.

[알고리즘 흐름도]



문제에서 주어진 조건에 맞게 입력되는 image 순서는 1,2,3,4 입니다. 이러한 순서의 이미지들을 작업하기 편리하도록 a,b,c,d로 변경합니다. 이후 하단부 상단부 각각 스티칭 작업을 합니다. 하단부의 경우 img a를 b에 맞게 변환한 img a_new를 생성하여 b와 합성한 후 스티칭 결과로 img n1을 선언합니다. 상단부의 경우도 동일한 원리로 img c를 d에 맞게 변환한 img d_new를 생성하여 d와 합성한 후 스티칭 결과로 img n2를 선언합니다.





마지막으로 img n1을 n2에 맞게 변환한 img n1_new를 생성하여 n2와 합성한 후 최종 스티칭 결과로 stitching result를 출력합니다.

[한계점]

1. 만약 image의 사분할을 특정한 규칙없이 임의로(ex. 상당한 크기 차이, 겹치는 구간이 부족 등)한다면 오류가 발생합니다.
2. slicing을 하였기 때문에 원본 이미지의 손실이 발생할 수 있습니다.

[여러 예시에 대한 적용 결과]

- 수정된 샘플 이미지

Img1	Img2	Img3	Img4
			
기존 : 413 x 227 변경 : 390 x 227	기존 : 414 x 228 변경 : 400 x 210	기존 : 415 x 227 변경 : 415 x 227	기존 : 414 x 227 변경 : 414 x 200

최종 스티칭 이미지

 stitching result



- 위 사진은 교수님께서 올려주신 sample 사진을 그대로 넣은게 아니라 각 image마다 크기를 임의대로 변형시켜서 넣은 image입니다.

- 카페 외관 이미지





Img1	Img2	Img3	Img4
			

최종 스티칭 이미지

 stitching result



- 주택 외관 이미지

Img1	Img2	Img3	Img4
			

최종 스티칭 이미지

 stitching result



Level 2

[영상의 순서를 파악하기 위해 사용한 방법]

우선 img4개를 각각 입력받고(code 10~31) level_A1a의 코드를 그대로 사용하여 img1,2의 방향을 판단하고(code 34~98) img3,4의 방향을 판단합니다(code 101~164). 그 후, case 3개로 나누어서 진행합니다. Case1: img1이랑 img2가 좌우 인경우, Case2 : img1이랑 img2가 상하 인경우, Case3 : img1이랑 img2가 대각선 관계인 경우 입니다. 제가 이와 같이 Case 3개로 나눌 수 있는 이유는 예를들어서 img1,2가 좌,우측 이미지이면 무조건 img 3,4도 좌,우측 이미지일 수 밖에 없는 원리를 적용하였습니다.

아래와 같이 case를 3개로 나누어서 기술하였습니다.

Case1 : img1이랑 img2가 좌우 인경우 (당연히 3이랑4도 좌우임) code: 168~270

① 우선 level A 1a의 코드를 그대로 가져와서 이미지 1,2 세트와 이미지 3,4 세트 중에 어떤 세트가 아래쪽 이미지인지 판단합니다. 대표로 이미지1과 이미지3을 비교하여 이미지1,2 세트의 방향을 나타내는 변수 img12에 할당하고 이미지3,4 세트의 방향을 나타내는 변수 img34에 할당합니다. 예를 들어, img12는 '상', img34는 '하' 로 할당됩니다.

② 위 과정에서 구한 네개의 이미지의 방향을 이용해서 아래측, 위측 양측 다 각각 좌우측 순서를 맞춰줍니다. 우선 이미지 1,2세트부터 순서를 맞추면, if문과 temp변수를 이용하여 img1이 우측, img2가 좌측 이미지 되게끔 변경합니다. 3,4 세트도 동일하게, if문과 temp변수를 이용하여 img3이 우측, img4가 좌측 이미지 되게끔 변경합니다.

③ 위 과정에서 구한 이미지1,2 세트의 방향을 나타내는 변수 img12을 이용하여 img 1,2,3,4를 각 순서에 맞게 img a,b,c,d에 할당합니다. 예를들어, img12가 위측

이라고 한다면 이미지 1,2세트와 이미지 3,4세트를 교환해준 뒤 a,b,c,d에 할당합니다. (물론 여기서도 level A 1b 과제에서 정의한 a,b,c,d 순서에 맞게 할당합니다. 이런 식으로 일정한 규칙으로 할당해줘야 뒤쪽에서 A_1b 코드를 넣었을 때 오류가 발생하지 않기 때문입니다.)

Case2 : img1이랑 img2가 상하 인경우 (당연히 3이랑4도 상하임) code: 275~377

① 우선 level A 1a의 코드를 그대로 가져와서 이미지 1,2 세트와 이미지 3,4 세트 중에 어떤 세트가 우측 이미지인지 판단합니다. 대표로 이미지1과 이미지3을 비교하여 이미지1,2 세트의 방향을 나타내는 변수 img12에 할당하고 이미지3,4 세트의 방향을 나타내는 변수 img34에 할당합니다. 예를 들어, img12는 '좌', img34는 '우' 로 할당됩니다.

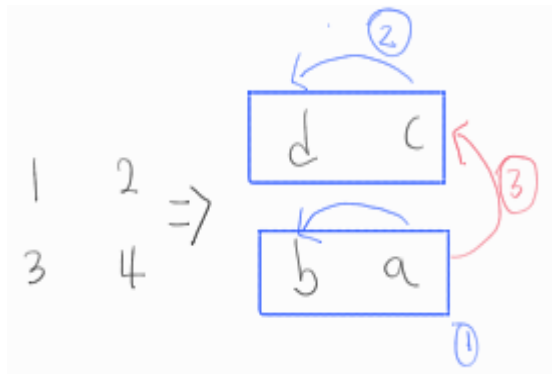
② 위 과정에서 구한 네개 이미지의 방향을 이용해서 우측, 좌측 양측 다 각각 위아래 순서를 맞춰줍니다. 우선 이미지 1,2세트부터 순서를 맞추면, if문과 temp 변수를 이용하여 img1이 아래, img2가 위 이미지 되게끔 변경합니다. 3,4세트도 동일하게, if문과 temp변수를 이용하여 img3이 아래, img4가 위 이미지 되게끔 변경합니다.

③ 위 과정에서 구한 이미지1,2 세트의 방향을 나타내는 변수 img12을 이용하여 img 1,2,3,4를 각 순서에 맞게 img a,b,c,d에 할당합니다. 예를들어, img12가 위측 이라고 한다면 이미지 1,2세트와 이미지 3,4세트를 교환해준 뒤 a,b,c,d에 할당합니다. (물론 여기서도 level A 1b 과제에서 정의한 a,b,c,d 순서에 맞게 할당합니다. 이런 식으로 일정한 규칙으로 할당해줘야 뒤쪽에서 A_1b 코드를 넣었을 때 오류가 발생하지 않기 때문입니다.)

Case3 : img1이랑 img2가 대각선 관계인 경우 (당연히 3이랑4도 대각선임) code : 382~401

2차원 배열, list개념, for문, if문을 사용하여 올바른 순서에 맞게 img a,b,c,d에 할당 하였습니다. (물론 여기서도 level A 1b 과제에서 정의한 a,b,c,d 순서에 맞게 할당 합니다. 이런 방식으로 일정한 규칙으로 할당해줘야 뒤쪽에서 A_1b 코드를 넣었을 때 오류가 발생하지 않기 때문입니다.) case3의 자세한 설명은 뒤쪽 주요 코드 설명에서 할 예정입니다.

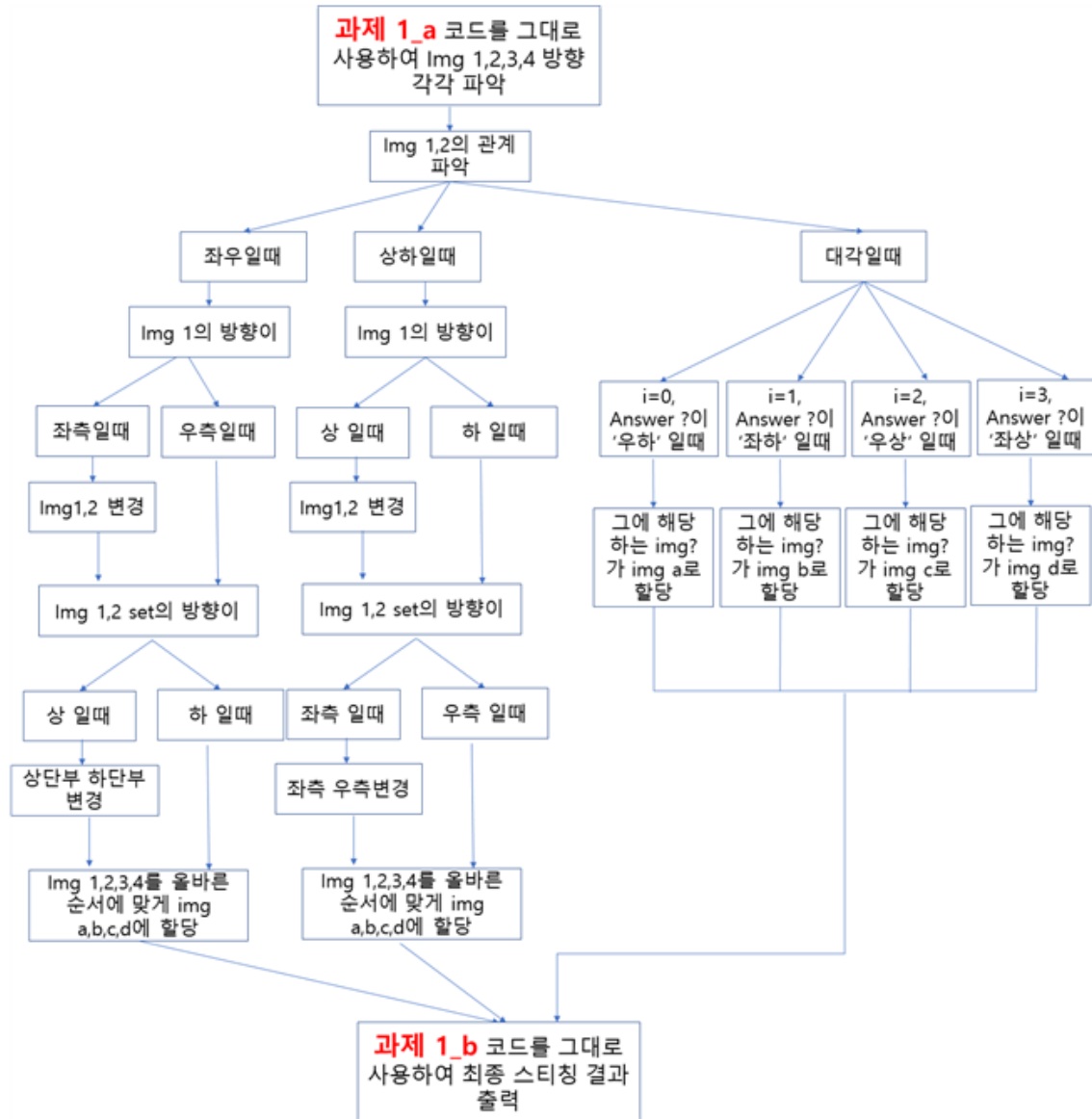
각 case별로 if문을 사용하였기에 결국 모든 경우의수 48가지(4X3X2X1)를 모두 수행할 수 있게 됩니다. Case를 통과하고 나면 제가 level 1_b 과제에서 임의로 정의했던 순서에 맞게 img a,b,c,d에 올바른 값이 할당됩니다.



<level1_b 보고서 p.7>

Case를 통과한 이후에 1-1, 1-2, 2-1, 2-2 순서대로 스티칭 작업하는 것은 level 1_b 과제의 코드를 그대로 이용하였습니다. 따라서 올바른 최종 스티칭 이미지가 출력됩니다. (code: 401~637)

[알고리즘 흐름도 (level 1-a,b 활용)]



위에서 설명하였듯이 처음에 1_a 코드를 사용하고, 위의 알고리즘을 거쳐서, 마지막으로 1-b 코드를 사용하는 흐름입니다.

흐름도를 표현함에 있어서 기존의 흐름도처럼 코드에 집중하기 보다는 제가 하고자 하는 알고리즘에 대한 설명을 중요시하였습니다. 모든 경우의 수에 대해 설명하기 보다는 경우의 수 중 하나를 특정하여 설명하였습니다. 예를들면, case 1에서 img1의 방향만 바꿔주는게 아니라 img3의 방향이 좌측이면 3,4도 변경해줘야하는데 이러한 부분들을 생략하였습니다.

[주요 코드 설명]

```
241      # 순서 2번 좌우측 맞춰주기
242
243      if answer1 == '좌': # 순서 바꿔주기 !! (만약에 img1이 좌측이면 바꿔줌 img1이 우측, img2가 좌측되게끔)
244          temp = img1
245          img1 = img2
246          img2 = temp
247
248
249      if answer3 == '좌': # 순서 바꿔주기 !! (만약에 img3이 좌측이면 바꿔줌 img3이 우측, img4가 좌측되게끔)
250          temp = img3
251          img3 = img4
252          img4 = temp
```

Case1 순서2번 과정에서 좌,우측 방향을 올바르게 맞춰주기, case2 순서2번 과정에서 위,아래측 방향을 올바르게 맞춰주기에서 모두 사용된 원리입니다. 우선 if문으로 바꿔줘야 할 상황을 선언합니다(예시로 answer1이 '좌'일 때). img1과 2를 변경할때에는 temp라는 변수를 이용하여 바꿔줍니다.

```
256      # 순서 3번 위아래정보 (img12)를 고려하여 abcd에 할당
257
258      if answer12 == '상': # 순서 바꿔주기 !! (만약에 img12가 위면 바꿔줌 img12가 아래로 오게끔)
259          temp = img1
260          img1 = img3
261          img3 = temp
262
263          temp = img2
264          img2 = img4
265          img4 = temp
```

Case1 순서3번 과정에서 위,아래 방향을 올바르게 맞춰주기, case2 순서3번 과정에서 좌,우측 방향을 올바르게 맞춰주기에서 모두 사용된 원리입니다. 우선 if문으로 바꿔줘야 할 상황을 선언합니다(예시로 answer12가 '상'일 때). Temp 변수를 사용하여 img1과 3을, img2와 4를 각각 변경해줍니다.

```

382  ## case 3
383  if answer1=='우하' or answer1=='좌하' or answer1=='우상' or answer1=='좌상':
384
385      aa = [[img1, answer1], [img2, answer2], [img3, answer3], [img4, answer4]]
386
387      answers = ['우하', '좌하', '우상', '좌상']
388
389      for i in range(4):
390          answer = answers[i]
391          for element in aa:
392              if element[1] == answer:
393                  if i == 0:
394                      imga = element[0]
395                  elif i == 1:
396                      imgb = element[0]
397                  elif i == 2:
398                      imgc = element[0]
399                  elif i == 3:
400                      imgd = element[0]
401                  break
402





```

2차원 배열, list개념, for문, if문을 사용하여 img 1,2,3,4를 올바른 순서에 맞게 img a,b,c,d에 할당하였습니다.

우선 aa변수에 img1에 대응되는 방향인 answer1, img2에 대응되는 방향인 answer2, img3에 대응되는 방향인 answer3, img4에 대응되는 방향인 answer4를 2차원 배열로 선언하고, answers에는 방향4개를 list로 선언합니다. 이후 중첩 for 문, if문을 통해 순서가 랜덤한 방향들이(예를들어, 우상,좌하,좌상,우하) 주어져도 그에 대응되는 img가 할당되게 하였습니다. 예를들어, 첫번째로 주어진 방향이 '우하'이라면 i=0일 때, answer=answers[0](즉, '우하' 일때) 조건이 만족하기에 element[0](즉, img1)이 imga에 할당됩니다.

[여러 예시에 대한 적용 결과]

- 수정된 샘플 이미지 (case3 대각선인 경우)

Img1	Img2	Img3	Img4
			
기존 : 415 x 227 변경 : 415 x 227	기존 : 414 x 228 변경 : 400 x 210	기존 : 413 x 227 변경 : 390 x 227	기존 : 414 x 227 변경 : 414 x 200

최종 스티칭 이미지

 stitching result



- 위 사진은 교수님께서 올려주신 sample 사진을 그대로 넣은게 아니라 각 image마다 크기를 임의대로 변형시켜서 넣은 image입니다.

순서를 랜덤하게 넣어도 올바른 스티칭이 완료되는 모습입니다.

- 카페 외관 이미지 (case1 좌우인 경우)

Img1	Img2	Img3	Img4
			

최종 스티칭 이미지



순서를 랜덤하게 넣어도 올바른 스티칭이 완료되는 모습입니다.

- 주택 외관 이미지 (case2 상하인 경우)

Img1	Img2	Img3	Img4
			

최종 스티칭 이미지

 stitching result



순서를 랜덤하게 넣어도 올바른 스티칭이 완료되는 모습입니다.

Level 1-B

[구현을 위해 생각한 IDEA 및 참고사진 활용방안]

수업시간때 배웠던 affine transformation의 rotation성분을 이용하였습니다.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2D in-plane rotation

옆 그림에 있는 호모그래피 행렬 M의 성분들을 자세히 다시 살펴보면, 1행에 있는 세개의 성분은 x축 방향으로의 이미지 변환 및 왜곡에 대한 정보를, 2행에 있는 세개의 성분은 y축 방향으로의 이미지 변환 및 왜곡에 대한 정보를

<출처 강의자료>

담고 있습니다. 또한, 위 그림에서 각도를 나타내는 4개의 성분들은 모두 회전에 대한 성분입니다. 따라서 각도를 나타내기 위해서는 arctangent 개념으로 1행1열 cos성분과, 2행1열의 sin성분을 계산하여 theta를 계산해야합니다.

하지만, 분침이 가리키는 각도만 나오는 상황에서 이 각도를 분으로 변환해야합니다. 현재 -180도부터 +180까지 각도가 나오는 상황에서 제가 생각한 아이디어는 양수 음수로 분리하여 분을 계산하는 방법입니다.

시간은 총 60분이고 각도는 총 360도입니다. 따라서 시간을 구하기 위해서는 각도에서 6을 나눠줘야 합니다.

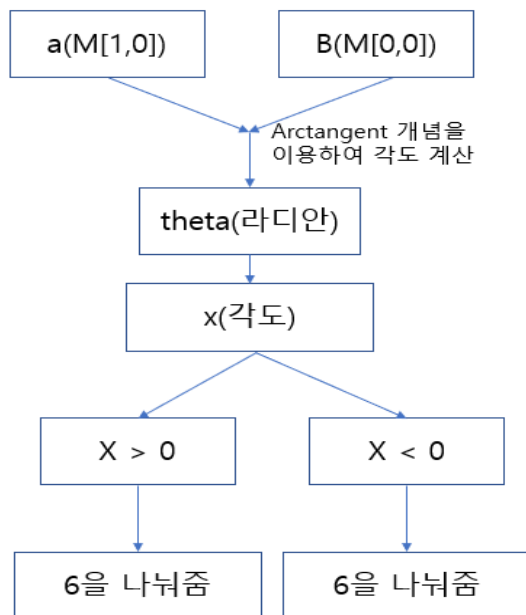
- 핵심부분 코드설명

```
69     a=M[1,0]
70     b=M[0,0]
71
72     theta=math.atan2(a,b)
73     x=math.degrees(theta)
74     #print(x)
75
76     if x>=0:
77         x=int(x/360*60)
78         print(x,'분')
79
80     else:
81         x = int(x / 360 * 60)
82         print(60+x,'분')
```

우선 각도 계산에 필요한 변수 M[1,0]과 M[0,0]을 각각 a, b에 할당합니다. Arctangent를 의미하는 함수 atan2를 이용하여 각도를 구합니다. 하지만, 여기서 구한 각도는 라디안이기에 math library의 degrees함수를 이용하여 도로 변환합니다.

If 문을 이용하여 각도가 양수일때, 음수일때로 구분합니다. 그리고 분으로 변환시키기 위해서 동일하게 6으로 나눠주고 분은 정수만 취급하기에 int를 씌워주고 출력합니다.

[알고리즘 흐름도]



[여러 예시에 대한 적용 결과]

Img1	Img2	output
		57 분
		45 분
		17 분

Img1에 ref image를 넣고, img2에 몇분인지 알고 싶은 imgae를 넣었습니다. Output을 확인하였을 때 분침이 가리키는 정확한 시각이 나온 것을 확인할 수 있습니다.

[결론]

우선 1-1에서는 문제를 해결함에 있어서 행렬 M의 성질을 잘 이해했다면 큰 어려움이 없었지만 threshold를 줘야만 하는 상황이 다소 어려웠습니다. 이론상으로는 1-1와 1-2 image는 x축 변화만 존재해야하는데 실제로는 그렇지 않아서 이러한 문제를 스스로 해결해봄으로써 조금 더 이해하게 되었습니다.

1-1을 해결할 때 우측이미지, 아래이미지를 img1로 뒤야 한다는 것을 이미 알았기에 1-2를 해결하는데 큰 어려움이 없었습니다. 하지만 이대로 실행시키니 아래 절반에는 검정색 화면이 나왔고 이러한 문제를 slicing을 통해 해결해봄으로써 조금 더 이해하게 되었습니다.

1-3을 해결할 때에는 처음부터 감이 잡히지 않아 모든 경우의 수를 노트에 정리해보았습니다. 그러던 중 3가지 case로 나뉘지는 것을 깨닫고 문제를 해결하였습니다. hw7에서 사용했던 코드를 바탕으로 여러가지 아이디어를 구현해보며 Switching a 과제를 해결하였는데 코드1_a, 코드1_b 등 모든 부분들이 톱니바퀴처럼 잘 맞물리게 하는 것이 상당히 어려운 작업이라는 것을 느꼈습니다. 또한, 처음에는 1-1 1-2 1-3 문제들을 따로 접근을 하였는데, 최종적으로 생각해보니 모든 소문제들이 다 연관이 있었다는 것을 깨달았고 처음부터 큰 그림을 그리고 진행했으면 시간을 조금 더 단축 시킬수 있었을텐데 하는 아쉬움이 남습니다.

또한, 1-3 과제의 코드길이가 637줄로 굉장히 길어 저도 작업을 하면서 수시로 헛갈렸던 적이 많았습니다. 만약에 1-1, 1-2 코드를 함수화하여 (예를들면 input image를 넣으면 출력으로 방향이 나오는 1_1a함수) 1-3 코드를 간소화했다면 코드가 더욱 간단해졌을텐데, 과제를 마무리할 즈음에 아이디어가 떠올라서 함수화를 구현하지 못한 것에 아쉬움이 남습니다.

하지만, 제가 직접 구현한 앞선 코드들을 사용하여 1-3에서 최종적인 스티칭 이미지가 출력되는 것을 확인하면서 앞선 두 코드가 각각 올바르게 짜였다는 것을 증명할 수 있었고 종합적으로 잘 맞물렸다고 생각합니다.

비록 스티칭 내장함수가 있기도 하고 내장함수 구현함에 있어서 제 코드가 상당히 둘러간 느낌이 있을지라도 이번 텀프로젝트를 하면서 예상 외의 부분에서 알게된 새로운 점들도 많고 문제 해결 능력을 향상시킬 수 있었다고 생각합니다.