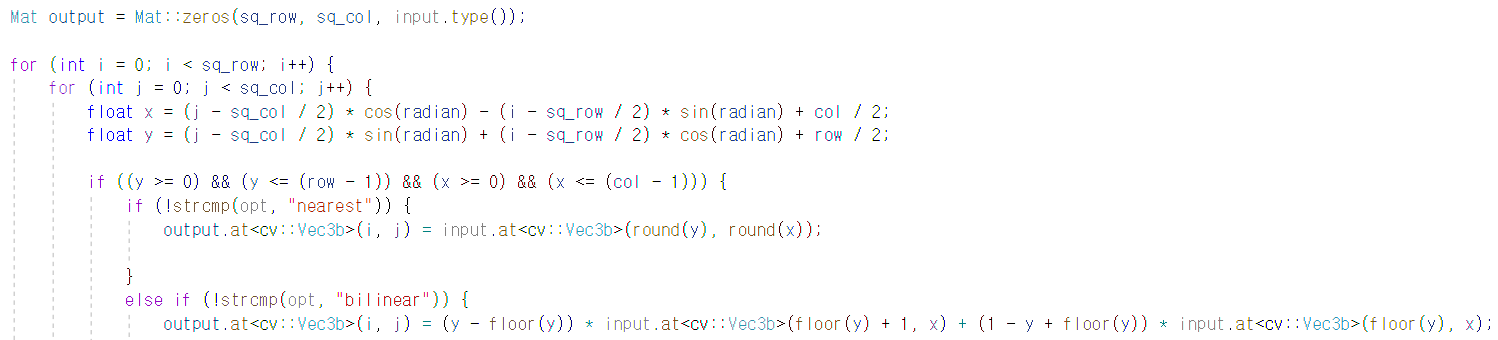
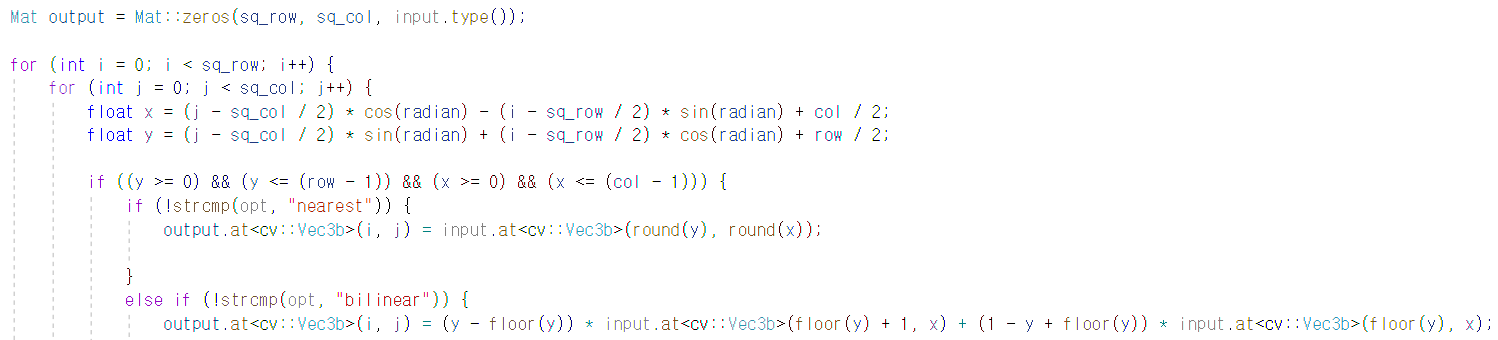
OSP\_assignment01\_technical report 1716009 백지희

**Practice 1 image rotation**

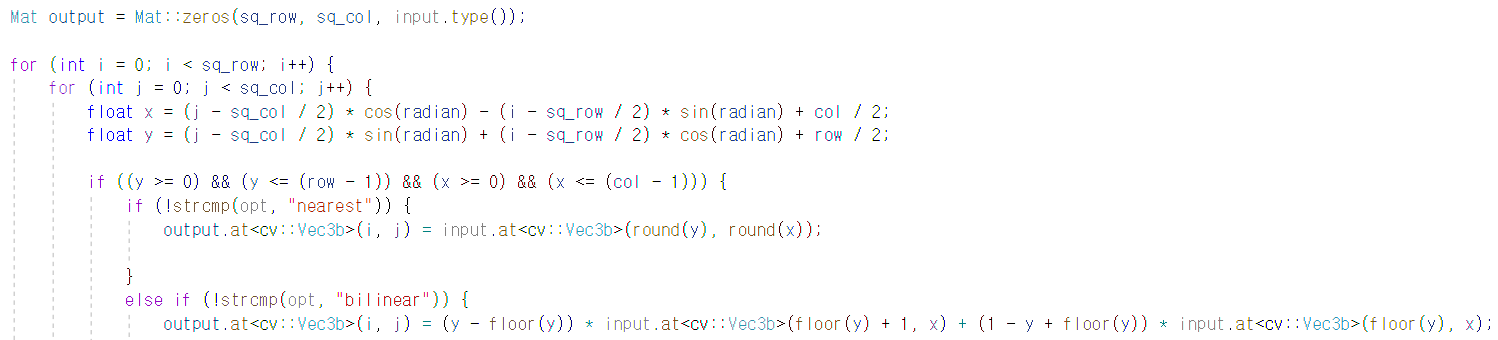
* Code 설명



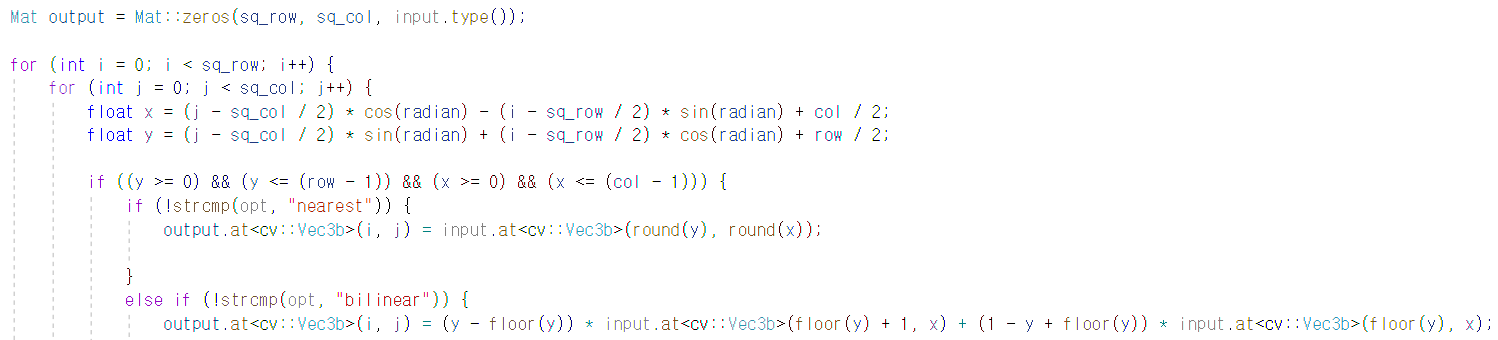
이 코드는 image rotation을 위한 것이다. 식을 이용하여 코딩한 것이다. (i, j)는 integer 값이고 (y, x)는 float 값일 것이다.



(i, j)에서 (y, x)로 inverse warping한 값이 input image 안에 있는 valid 한 pixel인지 확인한 후에 interpolation을 한다.

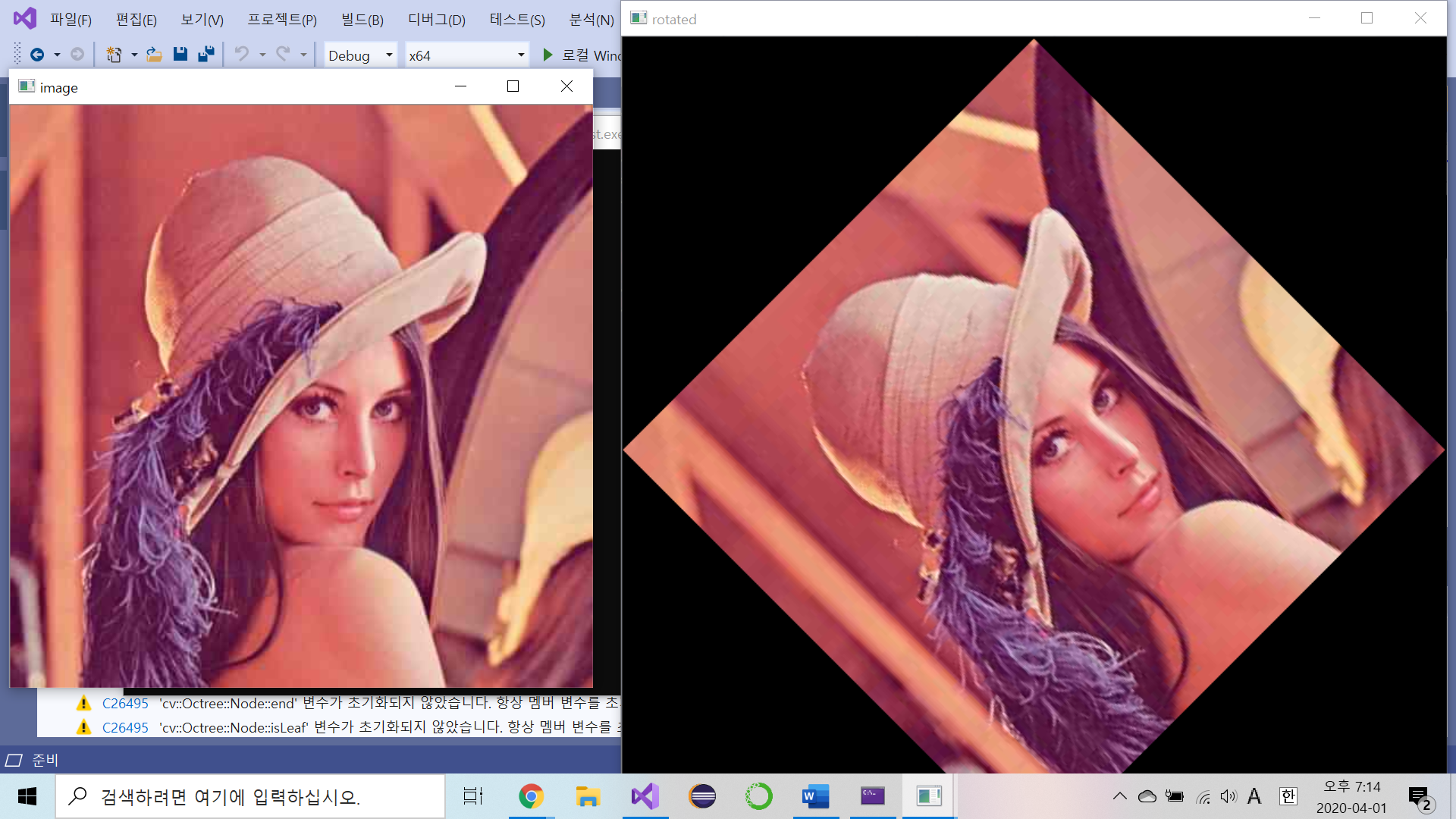


Nearest neighbor interpolation을 위한 코드로 y1<y<y2가 있을 때 y에서 더 가까운 값, 즉 반올림한 값에 re-sampling을 해준다. 따라서 반올림 함수인 round()를 이용하여, input matrix의 (round(y), round(x))에서 intensity 값을 output(rotated image)의 (i, j) pixel의 위치에 copy한다.



Bilinear interpolation을 위한 코드로 y1 < y < y2이고 x1 < x < x2 (y1, y2, x1, x2는 모두 정수) 이며 λ = y - y1일 때, 식 f ( y, x ) = ( 1 – λ ) f( y1, x ) + λ f( y2, x )를 만족한다. y1은 y를 내림 한 값이므로 내림 함수 floor()를 이용하여 작성하고, y2는 y1 + 1이다. 따라서 λ = y – y1 = y – floor(y)가 되기 때문에, 위와 같이 코드를 작성할 수 있다.

* 결과 화면



[image] [rotated]

* 분석

이 문제는 image geometric transformation 중 rigid(Euclidean) transformation으로 image rotation을 코딩해야 한다. Image를 rotate하면 pixel들이 꼭 integer 값에 위치하지 않기 때문에 re-sampling이 필요하다. 이 코드에는 두가지 interpolation이 있다. 하나는 nearest neighbor interpolation이고, 다른 하나는 bilinear interpolation이다. Nearest neighbor interpolation은 float pixel에서 가장 가까운 pixel의 intensity 값을 주는 개념이다. Bilinear interpolation은 float pixel 근처의 네개 픽셀을 통해 삼각비의 원리를 이용하여 intensity 값을 준다. 이 개념을 생각하며 위의 코딩을 하였다.

**Practice 2 image stitching using affine transformation**

* Code 설명



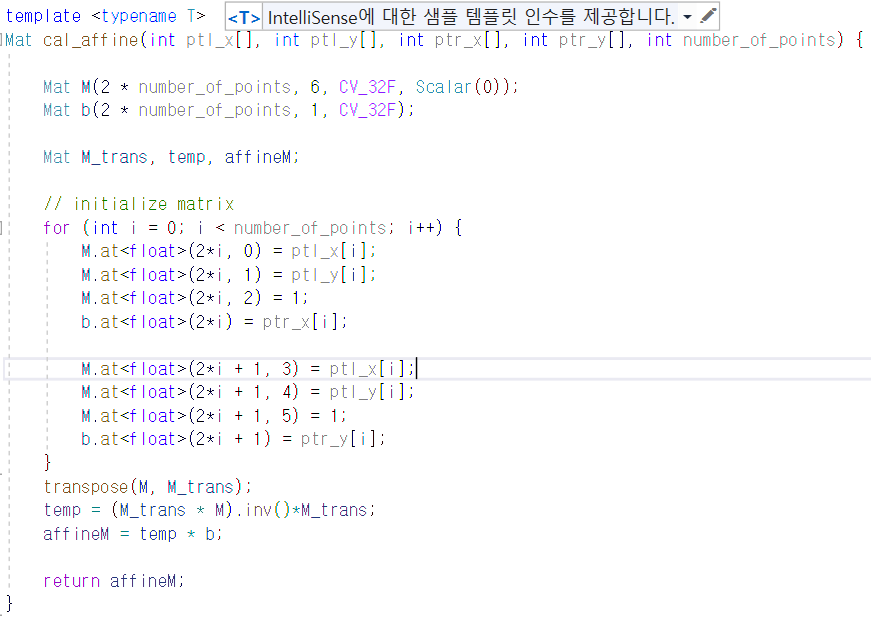
Affine transformation을 할 때 inverse warping을 하기 위해서 ) 식을 따라서 계산한다.



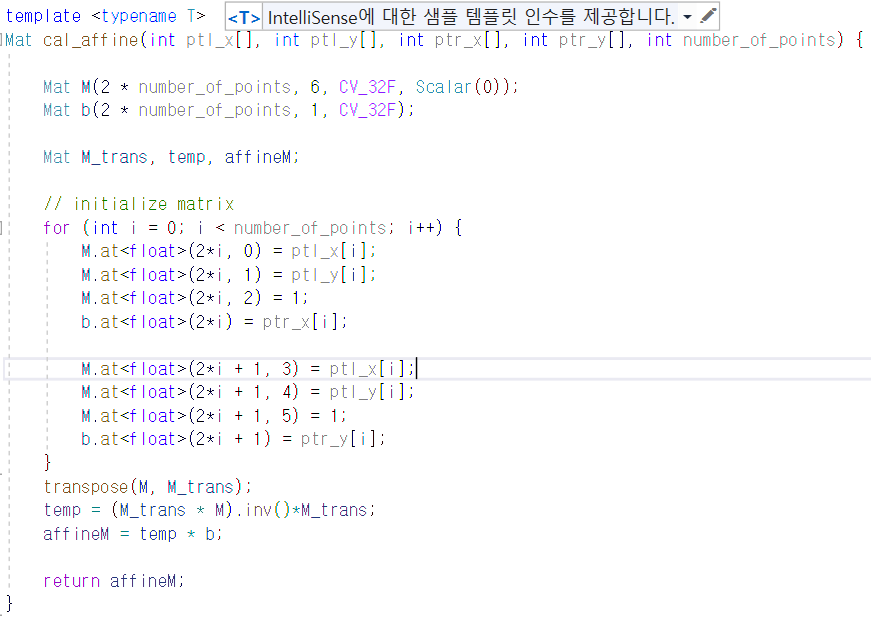
아래에 bilinear interpolation을 계산하는 코드를 보다 깔끔하게 작성하기 위하여 추가적으로 변수를 선언한다.



i2를 affine transformation한 이미지를 최종 output image인 I\_f에 저장해주는 코드이다. x1 < x < x2 이며 y1 < y < y2이고 이고 λ = x - x1 (x1, x2, y1, y2는 모두 정수) 일 때, bilinear interpolation을 하려면 ( 1 – λ ) f( x1, y ) + λ f( x2, y ) 식을 계산해야 한다. y1은 y를 내림 한 값이므로 내림 함수 floor()를 이용하여 작성하고, y2는 y를 올림 한 값이므로 올림 함수 ceil()을 이용한다. 따라서 λ = y – y1 = y – floor(y)이고 코드를 작성할 때 위에서 선언한 변수를 사용하였다.

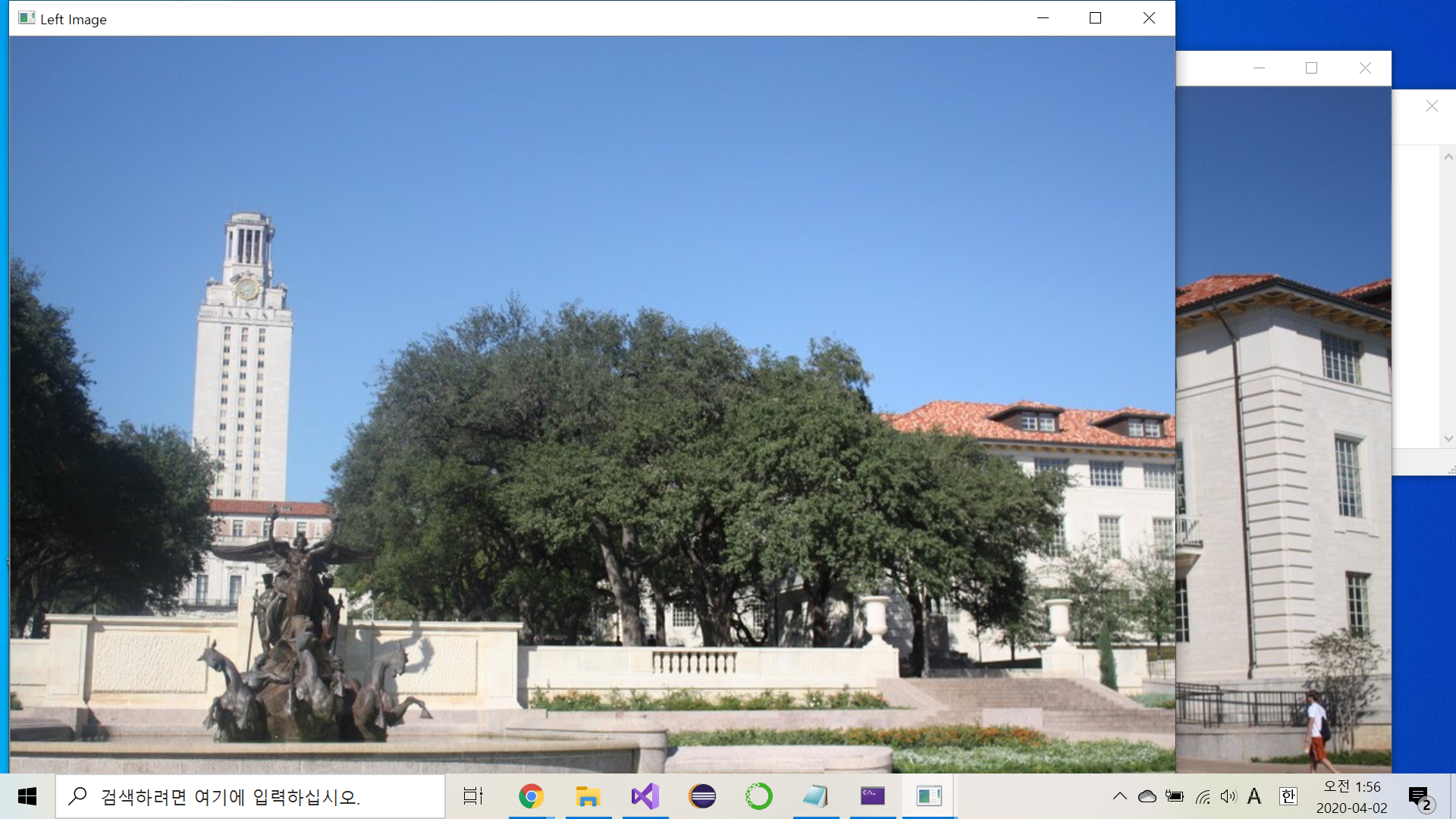


Corresponding pixel이 있을 때 affine matrix를 구하려면 2N x 6인 M=과 6x1인 x=의 행렬곱은 2N x 1인 b=을 이용하면 된다. b가 affine transformation되는 이미지 이므로 오른쪽 이미지의 corresponding pixel값을 b에 넣어주고, M에는 왼쪽 이미지의 corresponding pixel값을 넣어준다. M을 선언할 때 0으로 초기화 하였으므로 xn, yn, 1만 값을 넣어주면 된다. 반복문에서 i를 number\_of\_points 만큼으로 설정했기 때문에 한번 반복문이 실행될 때, M과 b 행렬의 2개 행에 값을 저장해야 한다. 따라서 2개 행 중 첫번째 행의 위치는 2\*i로 하고 2개 행 중 두번째 행의 위치는 2\*i +1 로 해준다.



위에서 선언한 M\_trans에 M을 transpose한 행렬을 넣어주고자 transpose(origin, destination) 함수를 사용했다. Mx = b에서 x값을 구하려면 (MTM)-1MTb를 계산해준다. 이 식에서 행렬 곱을 (MTM)-1MT 하나, 앞의 결과와 b의 행렬 곱 하나 이렇게 계산을 해주면 된다. 그 결과 affine matrix인 affineM 값을 얻게 된다.

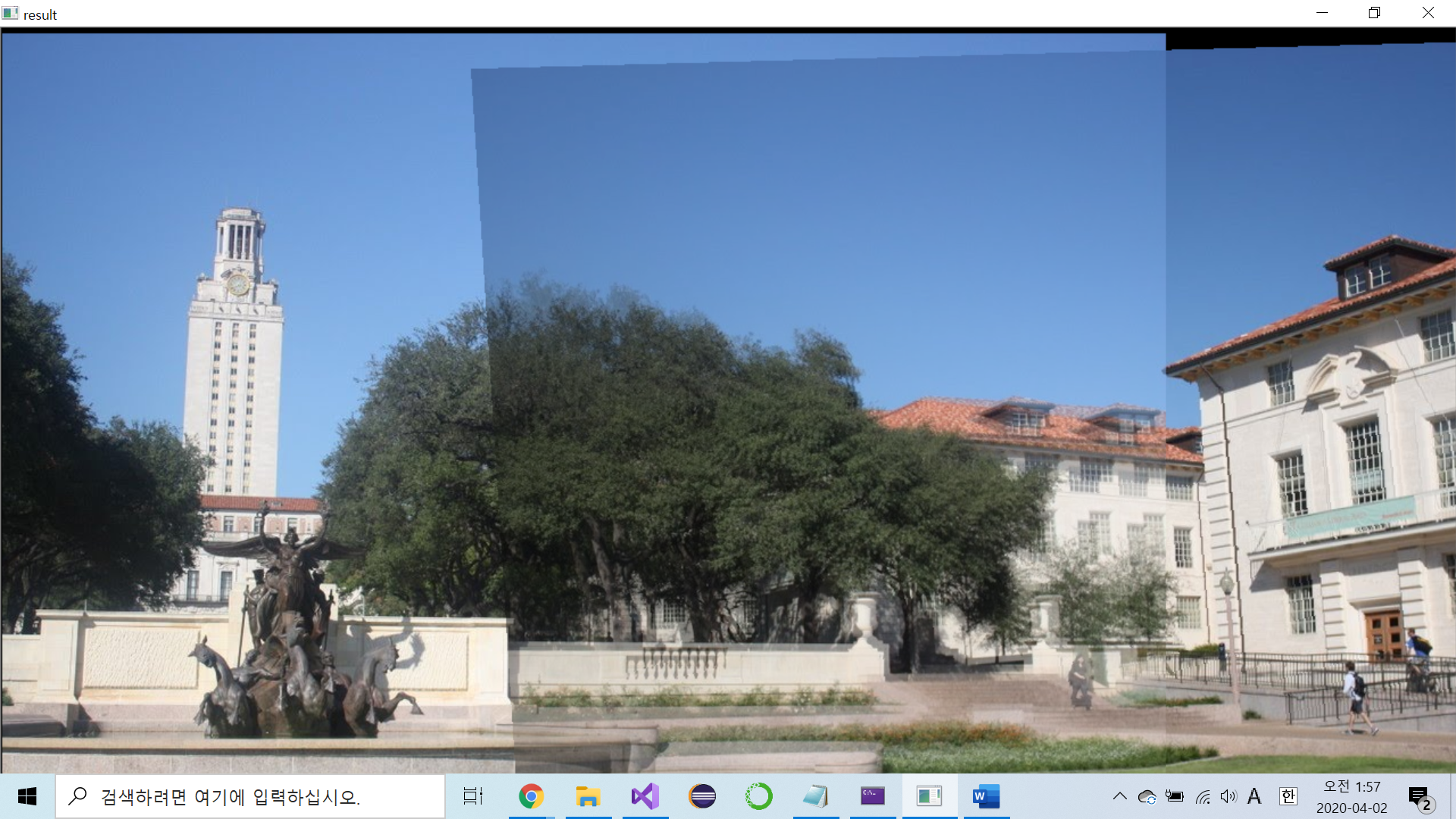
* 결과 화면



Left Image



Right Image



result

* 분석

이 문제는 image stitching using affine transformation으로 두개의 다른 앵글로 찍은 사진이 있을 때 일정 부분 겹치는 부분이 존재하는 경우에 두개의 사진을 이어 붙여서 더 넓은 앵글의 사진을 얻을 수 있다. 이를 수행하기 위해서는 corresponding pixel 값을 가지고 하나의 image를 affine transformation 해야 하는데 이 문제에서는 오른쪽 이미지를 transform한다. 이미지를 transform 하게 되면 pixel값들이 integer하지 않을 수 있기 때문에 inverse warping을 이용하여 오른쪽 이미지를 bilinear interpolation을 해준다. 마지막으로 왼쪽 이미지와 transform된 오른쪽 이미지를 blend stitch하면 output을 얻을 수 있다.