템플릿 매칭과 부분 워핑을 이용한 효율적인 원근 영상 워핑 기법

정대헌*, 조태훈*

*한국기술교육대학교

Efficient Image Warping Mechanism Using Template Matching and Partial Warping

Dae-Heon Jeong*, Tai-Hoon Cho*

*Korea University of Technology and Education

E-mail* : eses88@koreatech.ac.kr

요 약

이미지의 기하학적 변형은 이미지 보정을 위해 사용되며 컴퓨터 비전 분야에서 강체 변환, 유사 변환 등 많은 방법이 존재한다. 그 중에서도 워핑은 원근감이 있는 이미지에서 많이 활용되는 이미지 보정 방법이다. 일반적으로 워핑을 수행하기 위해서는 워핑할 위치에 대한 특징 점 4개를 추출해 워핑을 수행한다. 그러나 워핑 지점을 정확한 추출이 어려우며, 추출된 4개의 점을 이용해 원근 영상 보정을 할 경우 원본 이미지와 보정 후 영상과의 특정 부분 픽셀이 3~4픽셀 이상으로 오차가 나타나게 된다. 그렇기 때문에 본 논문에서는 정확한 워핑 결과를 가져오기 위해 템플릿 매칭을 이용해 워핑 할 부분의 4개점을 보다 정확하게 추출하고, 추출된 4개점들 중 2개의 점 각각에 대해 주변 3 by 3 영역으로 점을 이동 시켜 총 81번의 반복을 워핑 통해 이미지 보정하는 형태이다. 이와 같이 2개의 점을 주변 3 by 3 위치로 이동 시키면서 오차 픽셀이 1픽셀 이하로 나는 최적의 위치 즉, 최적 결과를 가져오는 4개의 점을 선정한 후 그 점들로 이미지 보정을 진행하여 최적의 결과를 가져올 수 있다.

ABSTRACT

Geometric transform of an image is used to image correction. Ridid-Body, Simlilary transform, etc, many correction methods are exist in computer vision. Image warping is used to correction for image with perspective. To image warping I extracted 4 feature point about warping position. But It is difficult to extract 4 points accurately and warping result with these point is occurs error over 3 or 4 pixel at warping position. So I used template matching to extract 4 points correctly and selected repeatedly 2 points of 4 points because to confirm result correctly. positions of 2 points are changed in near of 3 by 3 pixel and warped each change. So I selected optimal 4 points with a error of less than 1 pixel and finally, warped image using optimal points. For this way is possible to obtain optimum result.

키워드

기하하적 변형, Warping, 원근법, 컴퓨터 비전, Template Matching

Ⅰ. 서 론

이미지의 기하학적 변형이란 컴퓨터 비전 분야에서 사용되는 이미지 변환으로 선형 변환, 유사 변환, 공간 변환, 원근 변환, 워핑 등 다양한 방법들[1]이 존재한다. 그 중에서도 영상 워핑은 화소의 위치를 비선형적으로 특정규칙에 따라 이동하는 기하학적 처리 방법으로 구부러진 영상을 펴거나 임의의 형태로 구부리는 것과 같은 효과를 낼 수 있다. 영상 워핑은 인공위성에서 보내온 일그러진 영상 복원 방법부터 영화 CG방법까지 여러 방면에서 활용되고 있고 최근 명함 스캐너, 필터 카메라 등 스마트 폰용 어플리케이션에도 워핑 기법이 활용되는 것을 보면 영상 워핑이 다양한 분야에 사용된다는 것을 알 수 있다[8]. 일반적인 워핑 방법으로 원근 영상을 보정했을 경우 육안으로는 정확하게 워핑된 것처럼 보이지만 사실상 특정 부근에서는 원본과 비교했을 때 오차가 크게 나타난다. 본 논문에서는 이러한 문제점을 해결하고자 템플릿 매칭과 부분 워핑을

이용해 동, 서, 남, 북 방향의 측면에서 기울여 촬영된 영상들을 위에서 똑바로 촬영한 영상처럼 변환하는 효율적인 기법 대해 소개할 것이다. 본 논문의 본론에서는 제안하는 알고리즘에 대한 구체적인 방법에 대해서 설명할 것이며, 실험 결과에서는 제안하는 알고리즘으로 테스트를 한 결과에 대해서 소개할 것이다.

II. 본 론

본 논문의 본론에서 소개될 워핑 방법은 간략하게 다음과 같은 순서로 이루어진다.

(1) 워핑을 할 위치를 추출한다. (4개의 코너 점)
(2) 추출된 4개의 점들 중 좌 상단 및 우 상단에 위치하는 2개의 점에 대해서 iteration을 적용한다. iteration을 적용하는 이유는 워핑 후 원본과의 오차를 줄이기 위해서 이다.
(3) iteration 과정을 통해 선택되는 4개의 코너 점을 이용해 워핑을 진행한다. 이 때 워핑은 선택된 워핑 점을 통한 전체 이미지의 워핑이 아닌 오차를 확인할 부근에 대해서만 워핑을 진행한다.
(4) 최종적으로 원본 영상과 워핑 후 영상의 특정 부근에서 오차가 가장 적을 때의 코너 점을 이용해 전체 영상을 워핑한다.

2.1절부터 2.3절까지는 제안하는 알고리즘의 구체적인 방법을 제시할 것이다.

2.1 4개의 코너 점 추출.

본 논문에서 제시하는 방법은 워핑 하려는 위치의 4개의 코너 점을 추출하기 위한 템플릿 매칭의 사용이다. 템플릿은 워핑 되기 전 영상들 중 하나의 이미지에서 워핑 하려는 위치를 템플릿으로 제작한다. 이 때, 템플릿 이미지의 중앙에 워핑 하려는 위치가 오도록 제작한다. 예를 들어 영상 크기를 151 by 151로 제작하게 되면 워핑 하려는 점의 위치는 템플릿 이미지의 76행, 76열에 위치하도록 제작한다. 이처럼 4개의 점에 대해 4개의 템플릿을 생성한다. (Fig 3)

워핑 하려는 위치의 4개 코너 점을 찾기 위해 위 과정에서 생성된 템플릿을 이용해 Opencv 내의 matchTemplate[2] 함수로 템플릿 매칭을 진행한다. matchTemplate에서 사용하는 method들은 search 영상과 template 영상의 비교 방법을 지정하는데 본 논문에서 사용한 CV_TM_CCOR_NORMED method[2]는 상관계수 방법에서 정규화 계수를 나누는 방법이며 해당 method에 관한 수식은 아래의 수식과 같다.

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

매칭 후 최대 유사도를 가지는 픽셀 위치에서 행, 열 모두 템플릿 사이즈의 절반 크기에서 1을 더한 위치를 코너 점으로 선택 및 추출한다. 위의 과정을 통해 추출된 4개의 코너 점에 대한 서브 픽셀을 적용하여 좀 더 정확한 코너 위치를 찾아낸다. 템플릿 매칭 후 결과인 gray scale의 영상이 각 픽셀은 해당 픽셀의 인접 영역이 템플릿과 얼마나 일치하는지의 정보를 담고 있다. 이 결과 영상을 이용해 sub-pixel 연산으로 보다 정확한 코너 점을 추출한다. sub-pixel 계산식은 아래의 (2), (3)식과 같다.

$$X = \frac{\sum_{i=0}^{8}(point_x[i] \cdot result\_value[i])}{\sum_{i=0}^{8}(result\_value[i])} \quad (2)$$

$$Y = \frac{\sum_{i=0}^{8}(point_y[i] \cdot result\_value[i])}{\sum_{i=0}^{8}(result\_value[i])} \quad (3)$$

수식 (2)은 sub-pixel의 x좌표를 계산하는 식이며 수식 (3)은 y좌표를 계산하는 식이다. $point_x$는 코너 점 주변 3 by 3 픽셀의 x좌표이며, $point_y$는 y좌표이다. $result_value[i]$는 template matching 이후 나온 결과 영상에서 코너 점 주변 3 by 3 위치에서의 픽셀 값이다.

2.2 Iteration warping 및 워핑 오차 확인

2.2.1 Iteration warping

2.1 과정을 통해 추출된 4개의 코너 점을 이용해 워핑을 진행하는데, 전체 이미지를 워핑 하는 것이 아닌 워핑 후 오차를 확인할 일부분만을 워핑 한다. 이 과정을 진행하기 위해 우선 코너 점 iteration 선택을 진행하는데, 2.1에서 코너 점으로 선택된 점을 중앙에 두고 주변 3 by 3 픽셀로 코너 값을 이동시키면서 워핑을 하는 것이다. 다만, 4개 코너 점 모두에 반복을 적용하는 것이 아닌 오차 값이 큰 2개의 코너에 대해 적용하는 것이다. Fig 2 영상을 예로 들면 촬영된 위치를 기준으로 왼쪽 상단과 오른쪽 상단의 점이 왼쪽 하단과 오른쪽 하단의 점들보다 멀리 있다는 것을 알 수 있다. 이처럼 먼 쪽에 위치한 두 개의 코너 점에 대해서만 iteration을 진행하므로 총 81번의 코너 점 선택과 워핑이 이루어진다. 전체 영상에서 부분 워핑을 진행하기 위해 원본 영상의 오차를 확인할 부근에서 일정한 크기만큼 window를 생성하고 해당 window의 코너 위치를 확인 한다. (Fig 4) 이 후 워핑이 되는 기준 점을 맞춰 주기 위해 워핑 전 이미지에서 추출된 코너 위치와 원본 이미지의 코너 위치에서 window의 코너 위치를 빼준다. 이를 통해 워핑 시 상대적 위치를 구할 수 있고 이 상대적 위치로 Opencv 내의 getPerspectiveTransform[3] 함수를 이용해

변환 행렬을 생성하고 warpPerspective[3] 함수를 이용해 워핑 전 영상의 특정 위치만 워핑을 진행한다.

2.2.2 워핑 오차 확인

2.2.1의 과정을 마치고 일부분만 워핑된 영상을 원본 영상에서의 해당 영역과 오차를 확인하기 위해 다시 한 번 템플릿 매칭을 이용한다. 템플릿 영상은 원본 영상에서 워핑 된 영역 내의 오차를 확인할 수 있는 점을 포함하는 부분으로 제작한다. 이렇게 제작된 템플릿과 워핑 된 영상의 템플릿 매칭을 진행하여 특징 점의 오차를 확인한다. 즉, 1차 iteration에서 81번의 부분 워핑 후 오차가 가장 작을 때의 코너 점을 최적의 코너 점으로 선정한다. 오차가 가장 작을 때의 선정 기준은 원본에서의 특징 점의 위치와 워핑 후 특징 점의 오차 거리인데 워핑 후 오차 거리가 1픽셀 이내일 때의 코너 점을 최적의 점으로 선정한다. 만약 81번의 부분 워핑 후 오차 거리가 1픽셀 이내가 아닐 경우 81번의 iteration 과정을 다시 한 번 진행하게 되는데 이 때 iteration의 기준이 되는 코너 점은 첫 번째 iteration 과정에서 오차 거리가 가장 작은 코너 점을 최적의 코너 점으로 지정한다. 이 과정을 오차 거리가 1픽셀 이내가 될 때까지 반복 진행한다.

2.3 최종 워핑

n번째 반복 진행한 결과 오차 거리가 1픽셀 이내로 나타날 경우의 코너 점을 최종 코너 점으로 선정하고 전체 영상 워핑 단계로 넘어간다. 최적의 코너로 선정된 4개의 점을 토대로 getPerspectiveTransform[3] 함수로 전체 이미지에 해당하는 변환 행렬을 만들고 그 변환 행렬 값을 토대로 warpPespective[3] 함수를 이용해 최종 워핑한다.

Ⅲ. 실험 결과

실험 환경은 i5 인텔 CPU, 8GB 램이며, 프로그램은 비주얼 스튜디오 2013으로 구현하였다. 영상 내 코너 점 추출 및 오차 계산과 워핑을 위해 OpenCV 2.4.9 라이브러리를 사용하였으며, 실험 영상의 크기는 3904x3904 크기의 Gray scale 영상을 사용하였다. 실험 영상은 **Fig 1** 영상과 같이 위에서 촬영된 원본 영상으로 비교될 수 있는 영상과 **Fig 2**의 영상들 과 같이 원본 영상을 동, 서, 남, 북 방향에서 촬영한 이미지 총 5개를 사용하였다. **Fig 2** 4개의 영상들은 각 방향에서 기울여 촬영된 영상이며 이를 **Fig 1**과 같이 변환 하는 것이 목적이다.

이를 위해선 정확한 4개의 코너 점 검출이 필요한데 많이 알려진 harris corner detector[4]나 feature detector[5]로 코너 점을 검출 시 추출되기 원하는 코너 점만을 정확하게 추출해낼 수 없었다. 이 문제를 해결하기 위해 Fig 2의 영상과 같이 템플릿을 제작하여 각 방향 영상에 템플릿 매칭 방법을 적용해 코너 점을 정확하게 찾아 낼 수 있었다. 코너 템플릿 영상은 **Fig 2**의 남쪽 방향 영상에서 제작하였고 오차 계산을 위한 템플릿은 main 이미지에서 제작하였다.

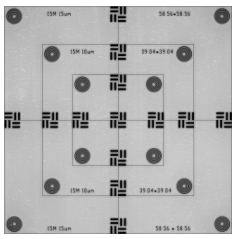

Fig 1. main 영상

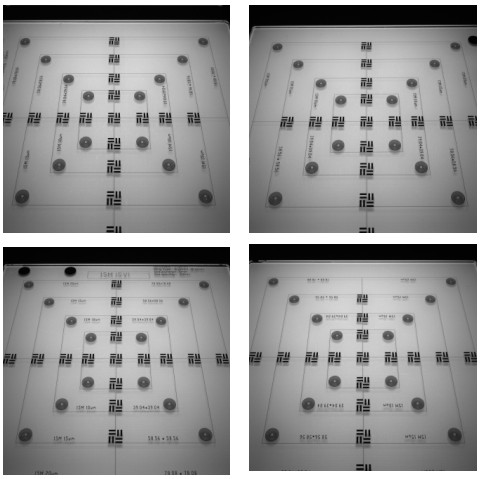

Fig 2. 왼쪽 위부터 동, 서, 남, 북 방향에서 촬영된 영상

이렇게 찾아낸 코너 점을 그대로 워핑 시 중앙 부근의 오차가 컸으며 3904x3904 크기 전부를 워핑할 경우 시간 또한 오래 걸리는 것을 확인하였는데 iteration warping을 이용해 오차는 1픽셀 미만으로 단축 시켰으며 부분 워핑으로 전체 영상을 워핑 하는데 소요되는 수행 시간보다 단축 할 수 있었다.

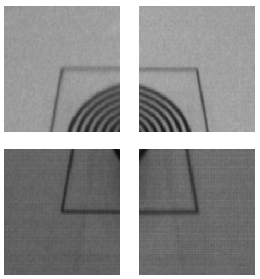

Fig 3. Fig 2의 남쪽 방향 이미지에서 추출한 4개의 코너 Template

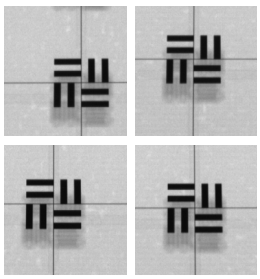

Fig 4. 오차를 확인할 중앙 부근의 450x450 크기 sub 영상

표 1은 3904x3904 크기의 이미지를 워핑 시켰을 경우와 450x450 크기의 부분 이미지를 워핑 시켰을 때의 시간 차이를 비교한 것이다. 이를 통해 영상의 전체 영역에 대해 워핑 할 경우 수행 시간이 더 오래 걸리는 것을 알 수 있었다.

표 2는 워핑 후 중앙 점 좌표를 원본 좌표와 비교한 것이다.

표 3은 워핑 후 영상과 main 영상의 중앙 점 오차와 각 방향에서 촬영된 영상의 iteration 실행 횟수와 그에 따른 수행 시간을 비교해본 결과이다. Iteration 실행 횟수는 2.2절의 과정을 몇 번 수행 했는가를 나타낸다.

Fig 5는 최적의 코너 점을 찾아 최종적으로 워핑한 결과이다.

	3904x3904	450x450
Time	0.082 ~ 0.094 s	0.001 ~ 0.002 s

Table 1. 워핑 시 크기에 따른 실행 시간 비교

	Center_Point
Main	x : 1954.00, y : 1954.00
East	x : 1953.23, y : 1953.64
West	x : 1954.09, y : 1954.95
North	x : 1954.33, y : 1954.86
South	x : 1954.26, y : 1954.05

Table 2. 워핑 후 중앙 점 좌표를 비교해본 결과

	Center_Point 오차	iteration 실행 횟수	수행 시간
East	x : 0.77, y : 0.36	3회	7.884 s
West	x : -0.09, y : -0.95	1회	2.744 s
North	x : -0.33, y : -0.86	3회	8.131 s
South	x : -0.26, y : -0.05	2회	5.465 s

Table 3. 각 영상별 오차, 실행 횟수, 수행 시간 비교

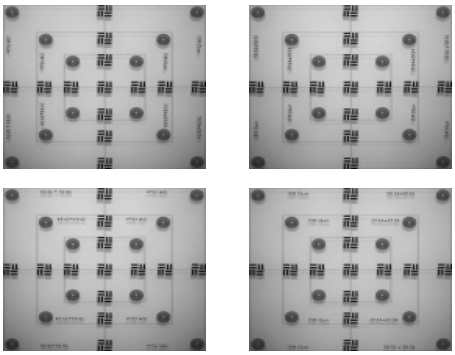

Fig 5. 각 방향에서 촬영된 영상의 최종 워핑 결과

Ⅳ. 결 론

본 논문은 template matching과 부분 워핑을 이용해 기존의 워핑 방법[6], [7] 보다는 보다 정확하게 워핑 지점을 추출하고 빠르게 워핑할 수 있는 기법을 제안하였다. 본 논문에서 제시된 영상처럼 부분 워핑을 할 위치가 굳이 중앙 부근이 아니어도 오차 확인이 필요한 부분을 추출해 낸다면 원근 변환이 필요한 모든 영상에서 해당 기법을 적용할 수 있다. 실험을 통해 확인한 결과로는 OpenCV 라이브러리에서 제공하는 matchTemplate 함수는 [1] 템플릿 크기에 따라 수행되는 시간의 차이가 없다는 한계가 있다는 것이다. 템플릿의 크기가 작을수록 매칭 속도의 빨라진다면 수행 시간을 더 줄일 수 있을 것으로 판단된다. 추후 연구로는 작은 크기의 template를 이용해 템플릿 매칭 시 속도 향상과 정확도를 높이는 연구가 필요하다.

참고문헌

[1] Geometric Transform of Image [Internet]. Available : http://opencv-python.readthedocs.io/en/latest/doc/10.imageTransformation/imageTransformation.html

[2] OpenCV 2014.04.11. [Internet]. Available: http://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html

[3] OpenCV 2014.04.11. [Internet]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html?highlight=getperspectivetransform#cv.GetPerspectiveTransform

[4] Harris Corner Detection [Internet]. Available : http://www.cse.psu.edu/~rtc12/CSE486/lecture06.pdf

[5] Feature Detection [Internet]. Available : http://m.blog.naver.com/laonple/220928491791

[6] J.P. Lewis, “Fast Template Matching”, Vision Interface 95, Canadian Image Processing and Pattern Recognition Society, Quebec City, Canada, May 15-19, 1995, p. 120-123.

[7] Luigi Di Stefano, Stefano Mattoccia, Federico Tombari, “Speeding-up NCC-based template matching using parallel multimedia instructions”

[8] D. W. Kim (2012,April) “Image Warping” [Internet]. Available: http://paeton.tistory.com/entry/%EC%9D%B4%EB%AF%B8%EC%A7%80-%EC%9B%8C%ED%95%91-image-Warping