**2가지 단안정보를 혼합한 Depth map 생성**

2015103910 김용균

1. **서론**

카메라 사진은 2차원이기 때문에 깊이 정보를 알아내기 어렵다. 사진에서 얻을 수 있는 깊이 정보는 한눈에 의해 수집된 정보와 같아 한쪽 눈으로 얻을 수 있는 정보인 단안 단서로 그 깊이를 판단해야만 한다. 단안 단서에는 어떠한 사물을 가리는 사물이 더 앞의 사물이라는 중첩, 평행한 도로와 같은 직선이 한 점을 향해 좁아지는 선형 조망, 관찰자가 움직일 경우 가까운 물체가 먼 물체보다 더 많이 움직이는 것으로 느껴지는 운동 시차 등이 있다.

이번 프로젝트에서는 객체의 깊이를 알아내고 Depth map을 완성하기 위해 위에서 이야기한 단서 중 중첩과 운동 시차를 이용하여 이미지의 depth 정보를 얻는다.

1. **프로젝트 내용**

**2.1 중첩**

우리는 어떠한 물체가 뒤의 물체를 가린 것인지, 아니면 같은 물체인 것인지 판단하기 위해 물체의 색의 변화, 패턴의 변화 등을 이용하여 그 형태를 파악한다. 형태를 파악하고 나면 각각의 물체가 차지하고 있는 영역을 알 수 있고 이를 이용해 물체의 앞뒤 관계를 파악할 수 있다. occlusion으로 인해 생긴 hole을 채우는 방법에 이 원리를 이용하기 위해 이미지에서 물체의 edge를 찾아낸다.

Edge를 찾아내기 위해 canny edge detector를 사용한다. Canny edge detector는 다음 3단계의 과정을 거쳐 edge를 찾아낸다.

1) Noise smoothing

먼저 Gaussian filter를 사용하여 blurring을 수행한다. 여기서 filter size는 5 X 5를 사용하였다. 이후 이 smoothing된 이미지에 sobel operator를 사용하여 각 픽셀들의 x방향과 y방향의 미분 값을 구한 뒤 이를 이용해 해당 픽셀의 magnitude와 direction을 얻는다. 그림 1이 얻어진 magnitude를 이미지로 표현한 사진으로 기존 사진의 edge들이 찾아진 것을 확인할 수 있다. 그러나 이 이미지는 edge가 선으로 표현되지 않고 두껍게 나타나기 때문에 어디가 가장 edge라고 부를 수 있는 곳인지 알기 어렵고, 물체의 edge뿐 만 아니라 무늬까지 edge로 나타내고 있는 것을 확인할 수 있다. 이제 edge를 얇게 만들어 정확한 edge를 구하기 위해 다음 과정을 수행한다.



그림 . sobel operator를 사용한 결과

2) Nonmax-suppression

1번 단계에서 얻은 magnitude와 direction을 이용해 최대값이 아닌 edge는 없애도록 한다. Direction은 픽셀에서 사용하기 위해 0도에서 22.5도를 기준으로 하여 45도씩 구분하여 해당 픽셀 주위의 8가지 픽셀에 대한 방향으로 구분하였다. 그러나 서로 마주 보는 방향에 대해서는 한 번에 처리할 수 있으므로 이를 합쳐 4가지 방향으로 다시 구분하였다.

다시 맨 처음 픽셀로 이동하여 x값과 y값을 증가시키며 모든 픽셀로 이동한다. 이동하며 픽셀의 direction값을 이용해 설정해둔 방향의 앞, 뒤 픽셀값과 비교하여 해당 픽셀값이 최대값이 아니라면 0으로 바꿔 없앤다. 그림 2에서 변경시킨 direction이 4일 때 앞, 뒤 값과 비교하는 것을 나타내고 있다.

1 2 3

4 4

3 2 1

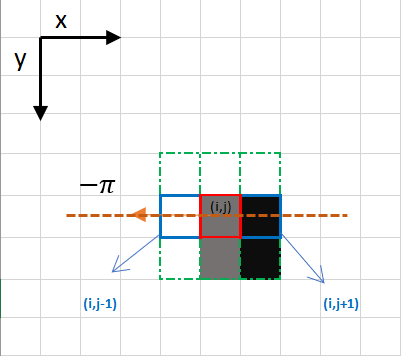


그림 . 방향 설정과 nonmax-suppression 비교 방법(좌), 수행 결과(우)

3) Hysteresis thresholding

Nonmax-suppresion의 결과를 이용하여 검출된 edge의 값에 대한 threshold를 2개로 설정한다. 두 threshold중 높은 값을 High, 낮은 값을 Low라고 할 경우 High threshold보다 커서 검출된 부분은 신뢰도가 높은 부분이라고 할 수 있다. 그러므로 이 부분들은 모두 살려둔다. 이후 이 High threshold와 연결된 부분 중에는 High threshold는 넘지 못했지만 Low threshold는 넘은 값들이 존재한다. 이러한 값들은 신뢰도가 높은 값에 연결된 부분이므로 edge로 판단하고 point들을 살려둔다. 그러나 low threshold는 넘었지만 High threshold를 넘은 local maxima와는 연결되지 않은 부분들과 low threshold를 넘지 못한 point들은 모두 버린다. 이러한 방식으로 두 threshold를 사용한 결과를 합쳐 결과를 얻는다. 이를 수행하기 위해 high threshold 이미지에서 edge인 픽셀을 찾으면 그 주위 3x3 범위에 low threshold 이미지에서 찾아진 edge가 있는지 찾고 없으면 5x5범위까지 찾는다. 찾았을 경우 그 픽셀로 이동하여 그 주위를 찾는 것을 반복하며 계속 연결해 나간다.



그림 . 순서대로 low, high, 결과 이미지 (low = 30, high = 60)

**2.2 운동 시차**

운동 시차를 이용하기 위해서는 다른 시점에서 같은 장면을 찍은 두 사진이 필요하다. 이후 두 사진에서 같은 점을 찾아야 한다. 이를 위해 일단 여기서는 이동 변환만 일어난 simple stereo system이라고 가정한다.

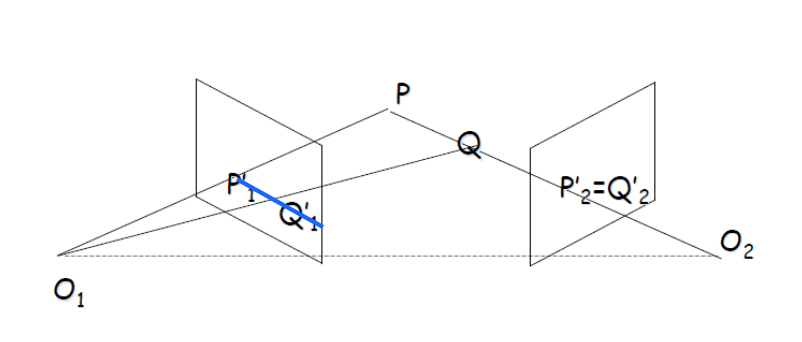
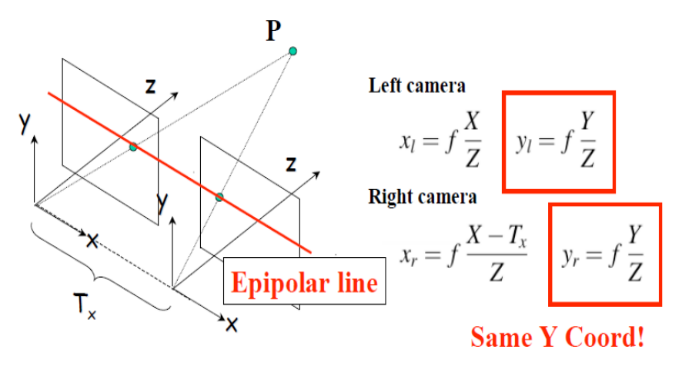


그림 . simple stereo system 그림 . Epipolar constraint

이때 그림4와 같이 사진을 나란히 두었을 때 그림과 같이 epipolar line은 x축에 평행하게 된다. 이 경우에도 그림 5에서처럼 O2에서 한 점으로 가려진 물체들이 O1에서 나타나게 된다. 이렇게 두 사진을 사용하면 한 사진에서 가려진 부분도 찾아낼 수 있고, 그림 4의 점 P의 사진 속 위치 차이는 가까워질수록 커지게 되므로 이를 이용해 점 P의 depth 정보도 알아낼 수 있다. 이를 이용해 depth map을 생성한다.

1) DSI (Disparity Space Image) 생성

epipolar line상의 점들을 매칭해가며 왼쪽에서의 픽셀이 오른쪽에서 어느 픽셀에 해당하는지 찾는다.

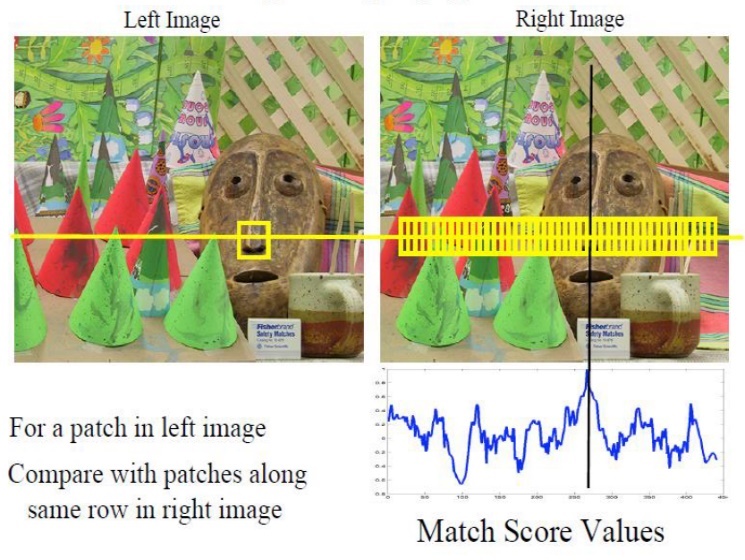


그림 . match score search

DSI 이미지 한 장은 하나의 y값에서 비교한 score들의 집합이 된다. 그림 6과 같이 기준 사진(그림 6은 왼쪽 사진)의 픽셀값과 비교 사진의 x축의 처음부터 끝까지 비교한 값들이 DSI 이미지의 한 줄이 되는 방식으로 비교 사진에서 한 줄과 비교가 끝나면 기준 사진의 그 옆 픽셀과 다시 비교 사진의 처음부터 끝 픽셀에 도달할 때까지 비교하며 한 줄씩 채워나간다. 기준 사진의 픽셀이 x좌표 끝에 도달하면 DSI 이미지 한 장이 완성되는 것이다. Score를 SSD로 계산했다면 결과 이미지는 가장 일치하는 pixel이 검게 나타난다. 이렇게 matching score를 시각화하여 나타낸 이미지가 DSI 이미지이다. 이때 왼쪽 사진의 x축 이동을 y축, 오른쪽 사진의 x축 이동을 x축으로 두어 그 score들을 이미지로 나타낸 것이 DSI이다.

여기서는 5X5 window를 사용하여 해당 픽셀을 중심으로 상하좌우 2만큼의 범위 전체를 비교하였고 SSD를 이용하여 계산하였다.

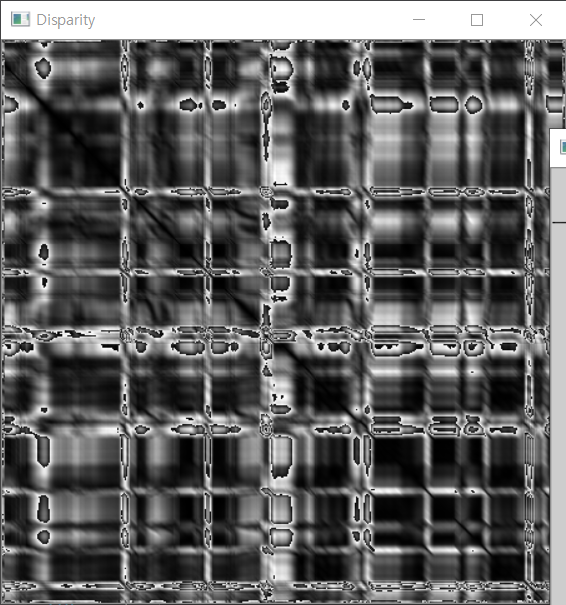


그림 . y = 45일때의 DSI 이미지

그림 7에서 보이듯 match score가 나타나며 대각선과 비슷하게 검은 선이 나타난다. 그러나 y축마다 단순히 가장 낮은 값을 matching point로 선택한다면 정확하지 않은 결과가 나오게 될 것이다. 그래서 기준 사진 pixel 주변의 pixel들은 비교 사진에서도 그 주위에 있다는 가정을 적용하면 matching point는 심각하게 떨어져 있을 수 없다. 이를 이용하면 검은 점들(SSD값이 낮은 점들)이 이어진 선과 같은 형태로 찾는 것이 좋다는 결론을 얻게 된다. 그러나 이 선은 중간중간 끊겨있어 이를 고려한 최적의 경로를 찾아야 하는데 greedy한 알고리즘을 사용하면 local optima를 찾아 잘못 탐색할 수 있으므로 global optima 경로를 찾기 위해 dynamic programming을 이용하게 된다.

2) Dynamic Programming

Global optima 경로를 찾기 위해 DP를 사용한다. 이미지는 좌우 사진의 픽셀 순서가 뒤바뀌는 경우가 거의 없으므로 왼쪽 사진에서만 나타나는 경우, 오른쪽 사진에서만 나타나는 경우, 양쪽 모두에서 나타나는 경우의 3가지 경우를 고려할 수 있다.

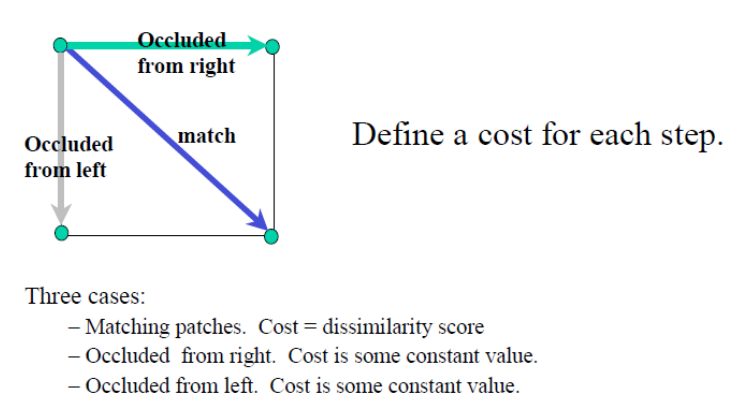


그림 . 3가지 경우의 수

이를 이용해 DP를 사용한다. DP는 각 시점마다 최적의 cost와 path를 탐색하고 저장해둔다. 다음 시점에는 이전 시점을 참고하여 새로운 cost와 path를 추정한다. 이 방법을 끝까지 사용하면 그림 12와 같이 이어지는 최적의 path를 찾을 수 있다. 정사각형으로 이루어져 있는 이 이미지는 좌상단에서 우하단을 향하는 가상의 대각선을 그릴 수 있는데 이때가 바로 우측 사진과 좌측 사진이 정확히 일치할 때이다. 그렇기에 이 대각선으로부터 path까지의 거리가 우리가 측정하고자 하는 disparity, 즉 같은 물체에 대한 좌측 사진과 우측 사진의 위치 차이이다.

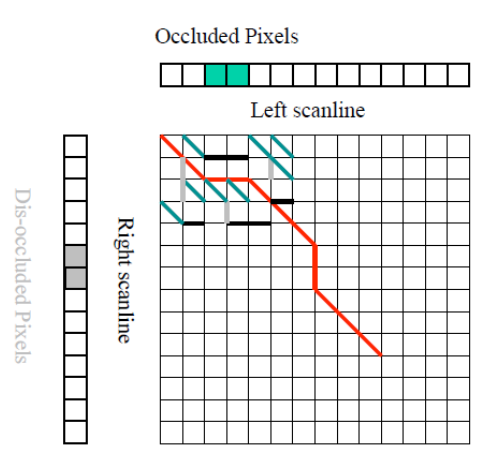
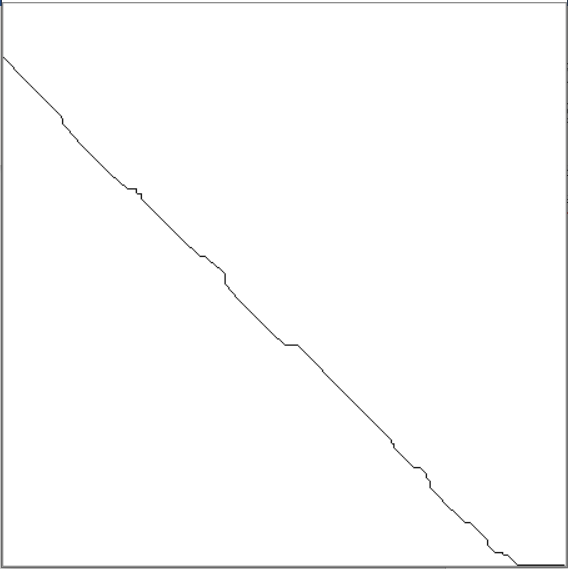
 

그림 . DP를 이용한 결과 구하기(좌), 이를 이용해 나온 y = 25에서의 경로(우)

이렇게 찾아진 경로를 따라가면서 값을 채우면 disparity image를 한 줄씩 그리게 되고 모든 DSI 이미지에 대해 위 방법을 수행하면 그림 10과 같은 결과를 얻게 된다.

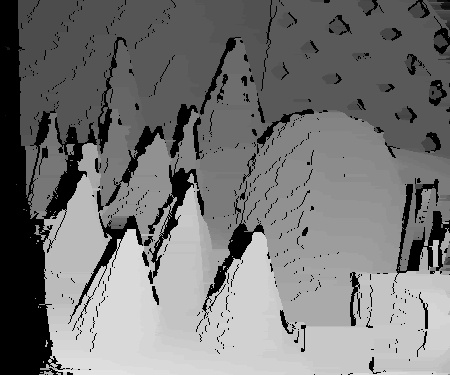


그림 . DP를 이용한 disparity image

**2.3 Hole filling**

대략적인 깊이 정보를 알았으니 2번에서 구한 edge 정보를 사용하여 occlusion된 부분을 채운다.

Edge가 완벽히 물체의 외부를 표현하고 있다면 이 edge를 기준으로 물체가 달라지거나 배경이 나타나므로 disparity 값이 달라질 것이다. 이 가정을 이용해 hole filling에 edge를 사용한다. Disparity 이미지와 edge 이미지의 픽셀을 같이 움직이며 edge나 나타나면 그 픽셀의 gradient direction을 확인하고 이에 맞게 Disparity 이미지를 참고하여 색을 채운다. 그 방식으로는 5x5의 window를 이용하여 변경한 direction이 1일 경우 그림 11과 같이 edge를 따라 빨간 삼각형 영역과 파란 삼각형 영역으로 나눌 수 있다.

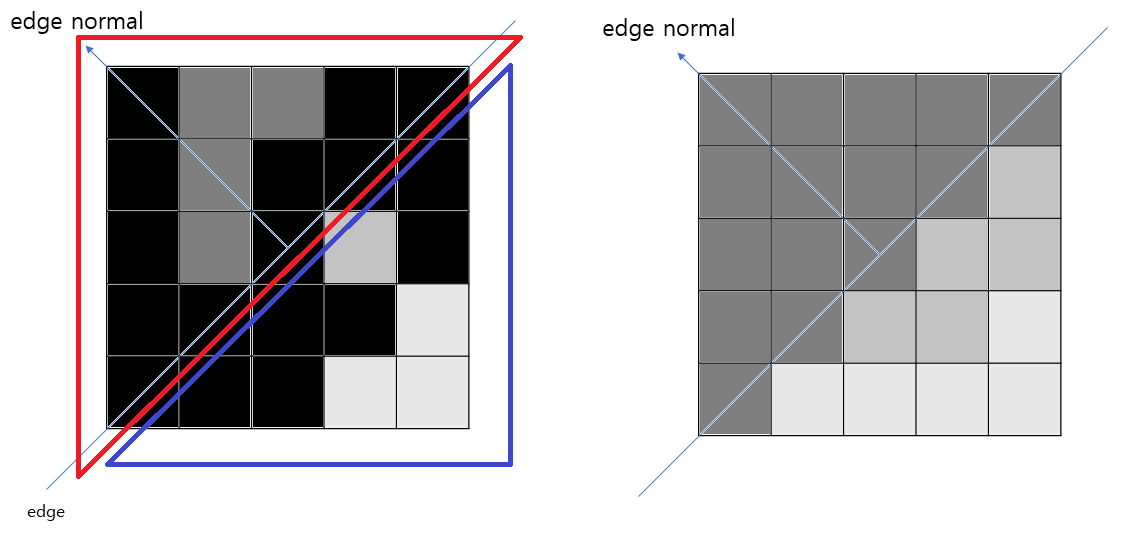


그림 . edge를 이용한 filling 이전, 이후 예

이때 검정 칸들은 값이 없는 부분으로 값을 채워야 하기 때문에 빨간 삼각형 영역에서 값을 찾고 그 값으로 빨간 삼각형의 빈 영역을 채운다. 칠하던 도중 다른 색이 나오면 채울 색을 그 색으로 바꾸고 채우게 된다. 파란 삼각형 영역도 마찬가지로 파란 삼각형 영역에서 값을 찾으면 그 색으로 빈 영역을 채운다. 그림 11의 오른쪽 그림이 이 알고리즘을 사용하 filling 결과 예시이고 이 과정은 화살표로 나타냈다. 이 방법을 사용할 때 표시한 삼각형 내부가 모두 occlusion된 부분이라면 값을 채울 수 없어 여전히 hole으로 남게 된다. 이 부분은 유효한 이전 픽셀의 값으로 채우는 방법으로 채워 최종 결과를 얻는다.

1. **프로젝트 결과**

**3.1 Hole filling**

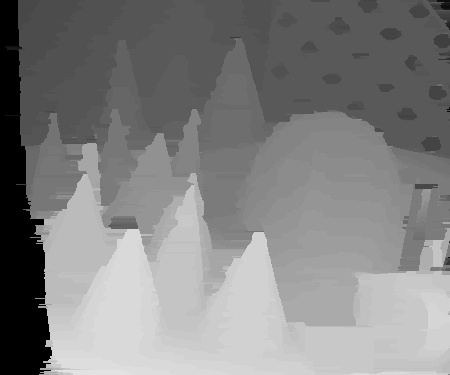
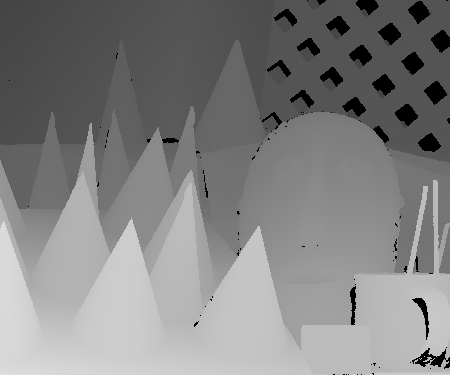
 

그림 . 순서대로 canny 결과, ground truth, 단순 hole filling, 프로젝트 결과

표 . 변수 설정

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Low thresh** | **High thresh** | **Matching window size** |
| **value** | 30 | 60 | 5 x 5 |

그림 12는 표1에 나타난 대로 설정했을 때의 결과이다. 보았을 때 프로젝트 결과 이미지가 표면이 더 울퉁불퉁하여 전체적으로 성능이 나빠 보이지만 왼쪽의 occlusion된 부분을 채우고 물체의 edge에 관련된 부분을 좀 더 살리고 있다. 이 결과를 정량적으로 확인하기 위해 ground truth 이미지와의 MSSIM(Mean Structural Similarity)를 측정하였다.

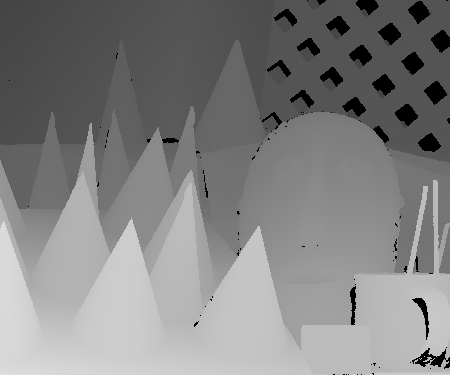


그림 . 비교 부분

표 . MSSIM 측정값

|  |  |  |  |
| --- | --- | --- | --- |
|  | **콘들** | **가면** | **전체** |
| **단순 filling** | 0.789874 | 0.859588 | 0.739214 |
| **프로젝트 결과** | 0.792855 | 0.858217 | 0.75585 |

콘 부분은 edge를 이용해 프로젝트의 결과가 조금 더 콘들의 모양에 가까워졌지만 가면 부분은 occlusion 된 부분이 많이 없었고, 그중 원래 값과 다르게 채워진 pixel로 인해 edge 너머의 부분까지 채워지며 원본과 달라져 단순 filling의 효과가 좀 더 좋게 나타났다. 그러나 전체 이미지로 보았을 때는 프로젝트의 결과가 조금 더 좋게 나온 것을 알 수 있다.

**3.2 다른 이미지**

총 6개의 scene에 대한 stereo image를 사용한 결과 중 2가지 이미지가 좋지 못한 결과를 얻거나, 전체적으로도 단순 filling 방법이 더 좋은 결과를 내었다.

3.2.1 disparity 이미지가 좋지 못한 이미지

표 . Flowerpots. window size 5. Ground truth, filling 전, 아래 순서는 그림 12와 같음

|  |  |
| --- | --- |
|  |  |
|  |  |

이 이미지는 원본 이미지를 확인해 본 결과 카메라 가까이 꽃병들이 쌓여있고 바로 뒤에 큰 물체가 있어 이 물체에 꽃병의 그림자가 생기게 되었다. 또한 뒤에 있던 물건 또한 무늬나 특징 없이 단색의 물체였다. 이 영역들은 색이 대부분 일정하다. 이 경우 주변의 값을 이용해 일치하는 곳을 찾을 때의 문제점이 발견되었는데 무늬가 없는 물체나 그림자 진 부분에 대해서는 색이 일정해 일치하는 픽셀을 잘 찾지 못해 disparity값이 제대로 도출되지 않는다. 이를 해결하기 위해 비교하는 left 이미지와 right 이미지를 비교하는 window size를 더 크게 하면 더 멀리 주위의 정보까지 포함하므로 detail을 좀 잃더라도 더 나은 disparity 값을 얻을 수 있을 것으로 기대했다. 이에 Window size를 5에서 11로 변경하여 실행한 결과 일부는 더 나아졌지만 오히려 아예 사라지는 부분까지 생겨 더 나아졌다고 판단하기 어려웠다. 이러한 경우 주변에 matching point로 선택 가능한 edge나 corner와의 관계를 이용하는 등 추가적인 방법이 필요하다고 생각된다.

3.2.2 edge를 이용한 filling이 더 좋지 못한 이미지

표 . Baby 1 Ground truth, filling 전, 아래 순서는 그림 12와 같음

|  |  |
| --- | --- |
|  |  |
|  |  |

표 . Baby1의 MSSIM

|  |  |  |
| --- | --- | --- |
| **MSSIM** | **단순 filling** | **프로젝트 결과** |
| **전체 영역** | 0.60873 | 0.604191 |
| **왼쪽 팔 영역** | 0.61957 | 0.594621 |

물체가 있는 이미지의 가운데 부분만을 이용하여 ground truth와의 MSSIM값을 얻은 결과값에는 큰 차이는 없어 전체적으로는 비슷한 차이를 가진다고 할 수 있다. 그러나 눈으로 보았을 때 인형의 팔 부분이 번져 보이고 실제로 MSSIM값도 프로젝트 결과값이 더 낮았다. 이 이미지의 경우 아기 인형의 배경이 지도여서 canny edge detector의 결과에 이 지도의 모습까지 edge로 나타나게 되었다. 그 결과 edge를 이용해 hole filling을 수행했을 때 오히려 인형의 팔부분에 지도의 edge가 겹치게 되면서 팔의 범위가 늘어나게 되었다. 이를 해결하기 위해 canny edge detector의 low threshold와 high threshold를 각각 50, 70으로 증가시켜 아기 인형의 edge만 남기려 하였으나 팔 부분의 지도의 edge가 계속 검출되며 이 경우에도 단순한 filling이 더 나은 모습을 보였다. 문제가 되는 부분의 edge가 정확히 검출되지 않아 그 주위에 값이 채워지는 것이 문제인 것으로 생각된다. 이에 대해서는 active contour 등을 사용해 더 정확한 윤곽선을 검출하면 더 나이질 것이라고 생각한다.

1. **결론**

처음 이 방법을 생각했을 때 canny edge detector가 불필요한 edge도 찾겠지만 그만큼 물체의 contour도 확실하게 찾을 수 있을 것으로 생각했고, contour가 정확히 찾아진다면 높은 정확도의 disparity image를 구할 수 있을 것이라 기대했다. 그러나 threshold값을 변경하여도 잘 찾기 힘든 부분이 존재했다. 또한 flowerpot 이미지에서 확인했듯이 contour를 잘 찾더라도 이를 활용하려면 filling 이전의 disparity 이미지 값이 어느정도 잘 나와야만 했다. 6가지 이미지 중 4가지 이미지는 프로젝트의 알고리즘이 더 나은 결과를 얻었지만 위에서 확인한 두 문제를 해결하기 위해 향후 disparity 값을 계산 시 주위에 matching point와의 관계를 사용해보고, contour를 찾을 수 있는 더 좋은 알고리즘을 활용하여 disparity image의 정확도를 높이도록 해본다.

1. **참고문헌**

- <https://vision.middlebury.edu/stereo/data/>

- <https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123>

- https://docs.opencv.org/master/d5/dc4/tutorial\_video\_input\_psnr\_ssim.html

- 컴퓨터 비젼 강의 자료