

컴퓨터 비전 기말 과제 리포트

2021.12.21

2019102210 컴퓨터 공학과

이 유 제

Stereo Matching Improvement using Dynamic Programming

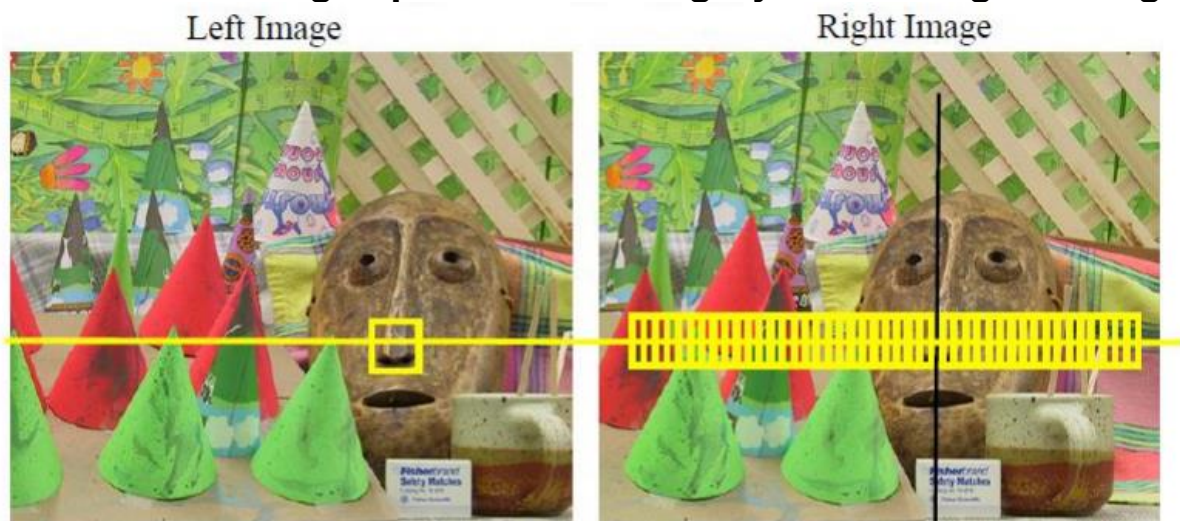


그림 1) 왼쪽부터 left image, right image

Stereo Matching을 통해 depth image를 구할 수 있습니다. 그림과 같이 왼쪽 이미지와 오른쪽 이미지를 입력으로 받아, 두 이미지를 사용해 Disparity Space Image(DSI)를 구한다. 그 후, DSI를 통해 각 path마다 cost를 부여한 후, Dynamic Programming을 통해서 최적의 path를 찾음으로써 Depth Image를 얻을 수 있다.

따라서 Depth Image를 얻기 위해서는 다음과 같은 과정이 필요하다.

1. 이미지 불러오기 (Loading Image)
2. 이미지 전처리 (Preprocessing Image)
3. DSI 구하기 (Calculating DSI)
4. DSI를 사용해 CostMap 구하기(Calculating CostMap)
5. CostMap을 사용해 최적의 Path 구하기(Getting Optimal Path)
6. Depth Image 구하기 (Getting Depth Image)

구현

코드를 구현한 환경에 대한 설명, 사용한 라이브러리, 각 단계별 코드 상세 내용을 작성했다. 단계별 코드에 대한 더욱 자세한 내용은 주석에 작성했으며, 본 보고서에서 코드를 통해 간단히 이해가 가는 부분은 생략했다.

환경

Google Colab 환경에서 코드를 구현하였다.

사용한 CPU와 메모리의 사양은 아래와 같다.

```
cpu family      : 6
model name     : Intel(R) Xeon(R) CPU
@ 2.20GHz
cpu MHz        : 2200.218
cache size     : 56320 KB
MemTotal:      13302916 kB
```

사용한 파이썬 버전은 3.7.12 이다.

사용한 라이브러리

사용한 라이브러리와 그 이유에 대한 설명은 아래와 같다.

```
google.colab
드라이브와 연동 및 colab 전용 cv2 imshow (시각화)
cv2
이미지 로드 및 이미지 크기 변경
numpy:
배열 저장 및 계산
```

이 외에는 파이썬 기본 내장 함수를 사용하였다.

1. Loading Image

함수: `img2np(path,resize=None)`

이미지를 로드(`cv2.imread`)하고 크기를 변경(`cv2.resize`)하기 위해 `cv2` 라이브러리를 사용하였다.

2. Preprocessing Image



그림 2) 흑백 이미지

함수: `grayscale(img)`

1982 년, ITU-R BT.601 에서 정의된 식(1)을 사용하여 기존 `rgb` 이미지를 `grayscale` 로 변경한다.

수식 1) `gray`

$$Y = 0.299 \times R + 0.587 \times G + 0.114 \times B \quad (1)$$

3. Calculating DSI

함수: `dsi(imL,imR)`

왼쪽 이미지(`H x W`)와 오른쪽 이미지(`H x W`)를 입력으로 받아, 두 이미지에 대한 `dsi(H x W x W)` 를 반환해준다.

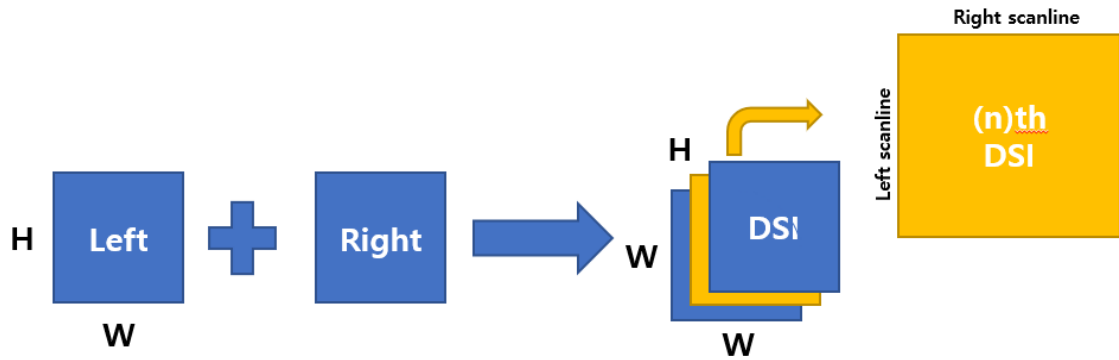


그림 3) DSI 계산 과정도

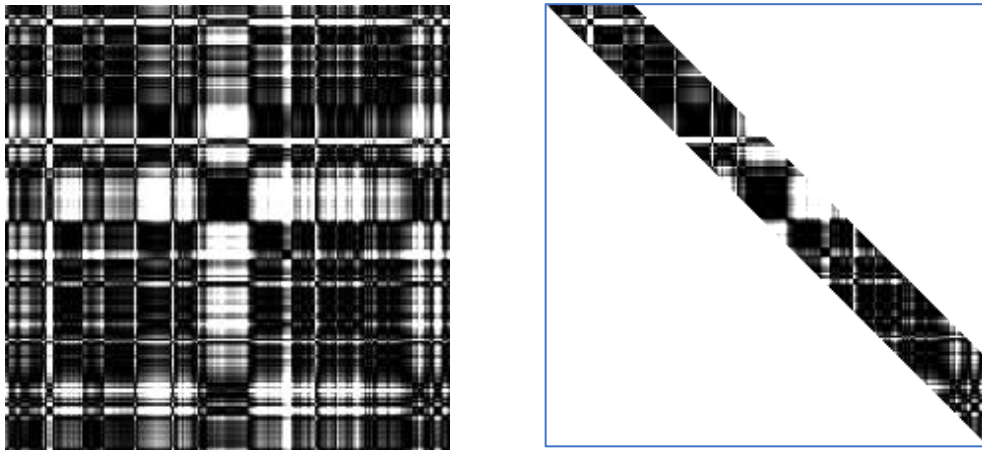


그림 4)왼쪽부터 DSI 예시, 경량화된 DSI 예시

depth 이미지를 구하기 위해서 가장 오랜 시간이 걸리는 DSI 부분이다. DSI 의 전체 부분에서 중심과 그 위쪽 부분만이 실제 CostMap 생성에 필요한 부분으로, 해당 위치를 제외하곤 계산하지 않도록 작성했다. 또한 계산하지 않는 지역들의 DSI 값을 500 으로 높게 세팅함으로써, 이후 CostMap 을 작성할 때 올바르게 않은 맵 생성을 방지했다.

이때, 각 dsi 의 x 축이 right image, y 축이 left image 이기 때문에 right image point - left image point > 0 인 부분만 계산할 수 있도록 했다.

right image point - left image point < 0 인 부분 모두를 계산하지 않고, 더욱 효율적인 계산을 하기 위해 휴리스틱한 방법으로 적절한 범위를 찾았고, 그 결과, right image point - left image point < - image width * 0.2 까지 이미지의 차이가 있지 않아 이 값으로 설정해주었다.

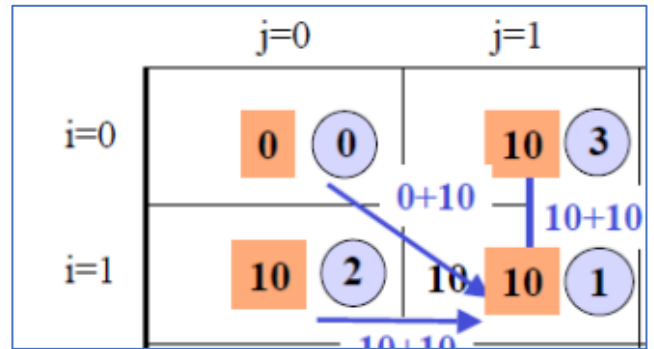
4. Calculating CostMap

함수: `costMap(dsi)`

dsi 맵을 입력으로 받아, 이를 토대로 CostMap 을 작성한다.

각 costmap 의 (0 행 i 열), (i 행 0 열) 들의 값을 $i * \text{Occlusion}$ 으로 설정해줌으로써, occlusion 을 고려해주었다.

```
m1 = cost[row,r-1,c-1]+dsi[row,r,c]
m2 = cost[row,r-1,c]+occ #오른쪽으로 감
m3 = cost[row,r,c-1]+occ #아래로 내려감
move = min(m1,m2,m3)
cost[row,r,c] = move
```



코드 1) dsi 코드 일부분

그림 5) COSTMAP과정

그림과 같이 현재 위치의 cost 가 이전 위치에서 최소 값을 가지는 값을 찾아 넣어준다.

여기서 사용되는 Occlusion 값은 하이퍼 파라미터로 적절한 값을 찾아야 한다. 이때 occlusion 값을 휴리스틱한 방법으로 적절한 값을 찾았다.

5. Getting Optimal Path + Getting Depth Image

함수: `findRoute(cost)`

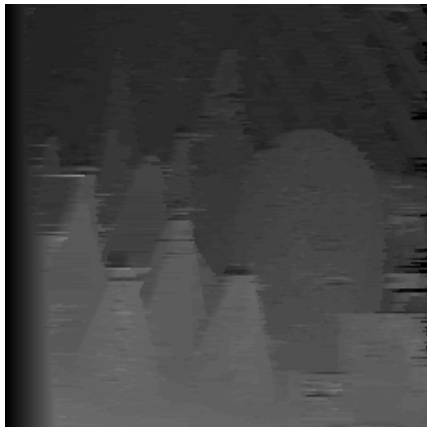


그림 6) depth image 결과

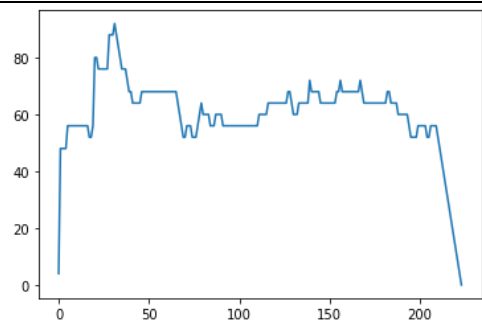
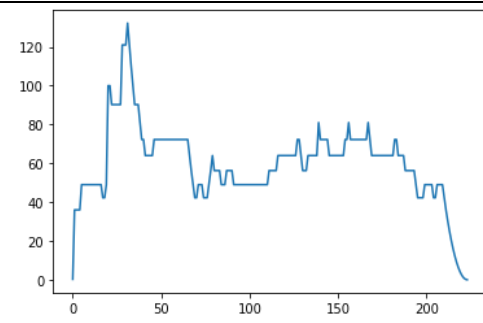
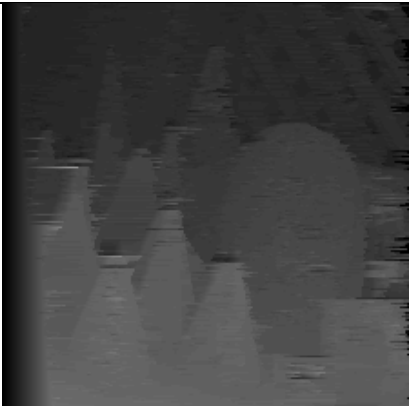
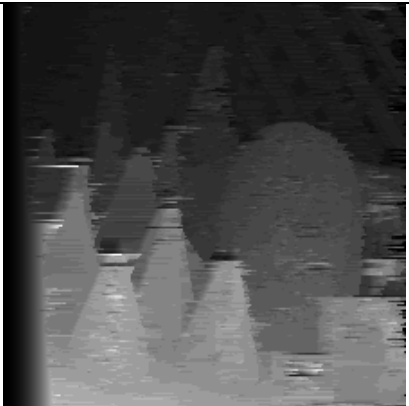
costMap 을 입력으로 받아, `costMap[W-1,W-1]`부터 최소의 cost 로 가는 route 를 찾고, route 를 통해 depth map 을 구한다. depth map 을 얻기 위해 아래와 같은 방식으로 각 포인트의 값을 구한다.

이때 사용하는 depth 공식은 절대값을 사용했다. 그러나 절댓값을 사용했을 경우, 값의 차이가 크지 않아 depth 의 차이를 육안으로 확인하기 어려운 단점이 존재한다. 이런 문제를 해결하기 위해 [성능 개선 - 실험 1]에서 새로운 depth 식을 제안한다.

성능 개선 Improvement

1. 실험 1

depth 이미지를 생성했을 때, depth 식에 따라 noise - depth 시각화 tradeoff가 발생함을 발견했다. 표와 같이 절댓값을 사용했을 때에는 노이즈가 적지만, depth를 시각적으로 인지가 잘 되지 않고 있다. 반대로 제곱을 사용하면, depth는 인지가 잘 됐지만 노이즈가 크게 발생함을 알 수 있다.

실험 1	$depth = abs(row-col)$	$depth = (row-col)^2$
100번째 line에 대한 depth 값		
depth image		
특징	Noise가 적다. 값의 차이가 적어 depth가 잘 보이지 않는 부분이 존재한다.	Noise가 크다. 값의 차이가 커 depth가 인식이 잘 된다.

(작성한 식에서 scale 변수는 제외하고 작성하였다.)

이러한 특징은 그림처럼 (r-c)의 값이 커짐에 따라 제곱의 증가분이 절대값의 증가분보다 크기 때문이다. 본 리포트는 이 두 식을 선형 결합하여 depth 값을 구함으로써, 두 효과를 smoothing 해주어 성능을 향상시켰다. 제안하는 식은 아래와 같다.

수식 2 depth 수식

$$depth = p \times abs(row - col) + (1 - p) \times (row - col)^2$$

다양한 p 에 대해서 계산해보았고, 실험 결과 선형 결합을 사용했을 때, 두 장점이 모두 드러나는 것을 볼 수 있다.











(실험 1) p	depth image		Focus	
0.00 ($=\text{abs}(r-c)$)				
0.25				
0.50				
0.75				
1.00 ($= (r-c)^2$)				

Figure 1) 실험 1 결과

2. 실험 2,3

그러나 선형 결합으로도 발생하는 노이즈가 있음을 확인했다. 이 현상을 최소화 하기 위해, 높은 값들을 억제해주는 방법을 제안한다.

ReLU 활성화 함수는 음수 값들을 0 으로, 양수 값들은 그대로 반환해주는 간단한 함수이지만, 매우 딥러닝 분야에서 효과적인 성능을 보여준다. ReLU 와 같이 threshold 전까지는 값 그대로 반환하고, threshold 이후부터는 threshold 값으로 모두 고정시키는 함수(Depth ReLU)를 제안한다. 기존 depth 이미지를 이 함수로 후처리 할 시, 노이즈가 줄어든 이미지를 얻게 된다.

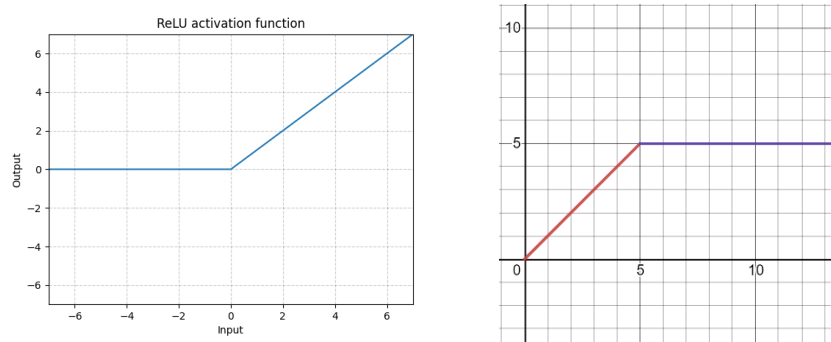


그림 7 왼쪽부터 ReLU 활성화 함수, 제안하는 함수 예시

수식 3 Depth ReLU

$$y = \begin{cases} x & x \leq threshold \\ threshold & x > threshold \end{cases}$$

(실험 2) Threshold	Image
No preprocess	
Threshold= 125	

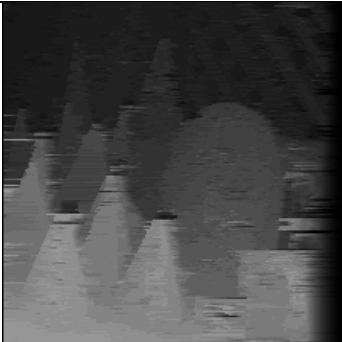
Threshold = 150	
-----------------	--

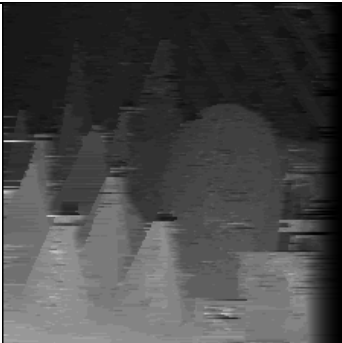

Figure 2) 실험 2 결과

그러나 실험 결과, depth 관련 정보가 크게 손실되거나, 노이즈를 쉽게 잡지 못했다. 이를 개선하기 위해, threshold 후에 기울기가 감소시켜 높은 값을 완만하게 억제하는 함수를 제안했다.

수식 4) Depth LeakyReLU

$$y = \begin{cases} x & x \leq threshold \\ \frac{1}{2}threshold + \frac{1}{2}x & x > threshold \end{cases}$$

또한 threshold 를 휴리스틱한 방법으로 설정하게 되면, 각 이미지마다 설정해줘야 하는 값이 달라지기 때문에 전체 이미지의 P% 상위에 해당 하는 값을 threshold 로 설정함으로써 함수를 일반화했다.

(실험 3) Threshold	Image	
No process		
P=0.2		


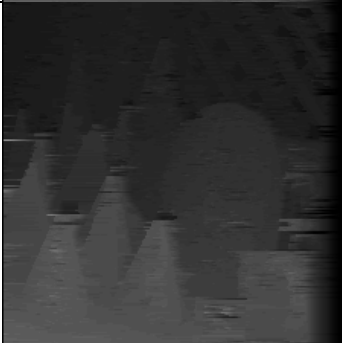
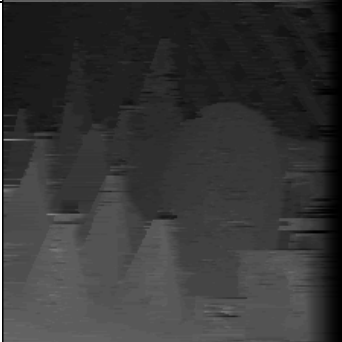
P=0.5			
P=0.7			
P=0.9			







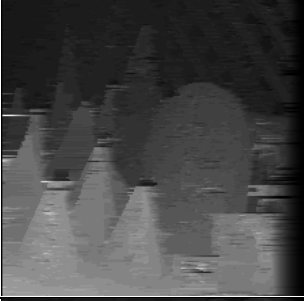

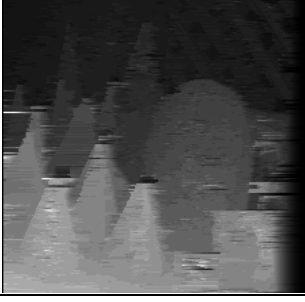

Figure 3) 실험 3 결과

실험 1,2,3 에 대한 표는 부록에 다시 첨부하여, 한 화면에 첨부하였다.

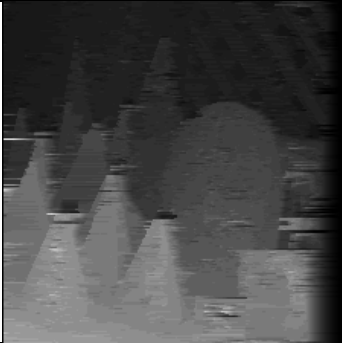
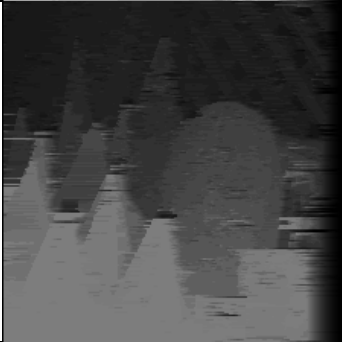
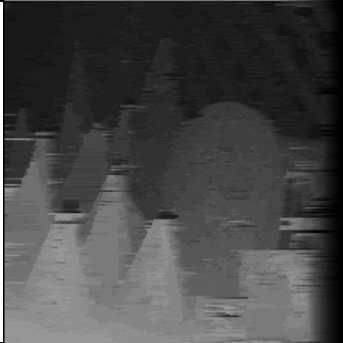
결론

stereo matching 을 위한 두 이미지를 사용해 depth image 를 구했다. 이 과정에서 DSI 및 CostMap 을 구해 Optimal Path 를 찾았다. DSI 를 추출할 때, 효율적인 계산을 위해 필요한 위치의 계산만 진행하도록 했다. 성능 개선에서 새로운 depth 수식을 제안하여 기존 depth image 들보다 noise 를 줄이고 depth 선명도를 개선했으며, 후처리 공식을 제안해 depth image 의 noise 를 감소시켰다.

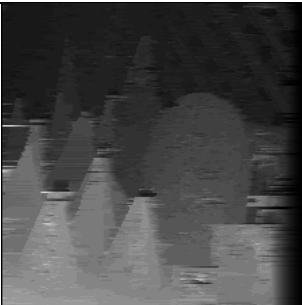



부록 A

(실험 1) p	depth image		Focus	
0.00 (=abs(r-c))				
0.25				
0.50				
0.75				
1.00(= (r-c)^2)				

부록 B

(실험 2) Threshold	Image	
No preprocess		
Threshold= 125		
Threshold = 150		

부록 C

(실험 3) Threshold	Image	
No process		
P=0.2		
P=0.5		
P=0.7		
P=0.9		