[Stereo Matching 보고서]

2018100727 이경준

(1) 초기 환경 설정

```
!pip install open3d
```

```
import numpy as np
import open3d as o3d
import plotly.graph_objects as go
from google.colab import drive
import cv2
from google.colab.patches import cv2_imshow
from PIL import Image
drive.mount('/content/drive')
```

```
%cd /content/drive/MyDrive/'Colab Notebooks'/3DDP/HW2/
!ls
```

이때, Open3D는 3D 데이터 처리와 시각화를 위한 오픈 소스 라이브러리이다. 3D 포인트 클라우드, 3D Mesh 등의 다양한 형식의 3D 데이터를 읽고, 변환하고 시각화 하는 기능을 제공한다.

(2) 이미지 불러오기 & Camera Calibration 값 대입

```
left_img = cv2.imread('left.png')
right_img = cv2.imread('right.png')

intrinsic_matrix = np.array([[1.09029096e+03, 0.00000000e+00, 7.07542608e+02], [0.00000000e+00, 1.08922050e+03, 5.45461473e+02],
                             [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]],np.float64)
dist = np.array([[ 1.65437913e-01, -5.79061012e-01,  2.96852001e-04, -5.97762518e-04,  8.06805488e-01]],np.float64)
```

아이폰 12pro의 카메라로 왼쪽과 오른쪽이 x 방향으로 1cm(0.01m) 차이 나도록 사진을 촬영했으며, 해당 사진을 각각 'left.png', 'right.png'로 저장하였다.

그 뒤, OpenCV의 imraed 함수를 이용해 구글 드라이브 폴더에 저장돼 있는 'left.png', 'right.png' 이미지 불러온 뒤, 해당 이미지 파일 데이터를 left_img, right_img 변수에 저장했다.

그리고 intrinsic parameter와 distortion coefficient 같은 경우에는 이전에 수행했던 camera calibration 실습 자료를 이용해 다시 계산하였으며, 좀 더 정확한 값을 이번 실습에 이용하고자 했다.

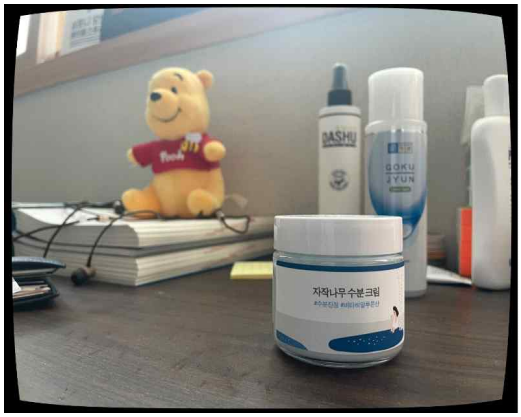
(3) Image Rectification

```
R = np.array([[1,0,0], [0,1,0],[0,0,1]],np.float64)
T = np.array([0.01,0,0],np.float64)

image_size = (left_img.shape[1], left_img.shape[0])
R1, R2, P1, P2, Q, roi1, roi2 = cv2.stereoRectify(intrinsic_matrix, dist, intrinsic_matrix, dist, image_size, R, T)

map1, map2 = cv2.initUndistortRectifyMap(intrinsic_matrix, dist, R1, P1,image_size, cv2.CV_32FC1)
dst1 = cv2.remap(left_img, map1,map2, cv2.INTER_LINEAR,cv2.BORDER_CONSTANT)

map3,map4 = cv2.initUndistortRectifyMap(intrinsic_matrix, dist, R2,P2,image_size, cv2.CV_32FC1)
dst2 = cv2.remap(right_img, map3,map4, cv2.INTER_LINEAR,cv2.BORDER_CONSTANT)

cv2_imshow(dst1)
cv2_imshow(dst2)
```

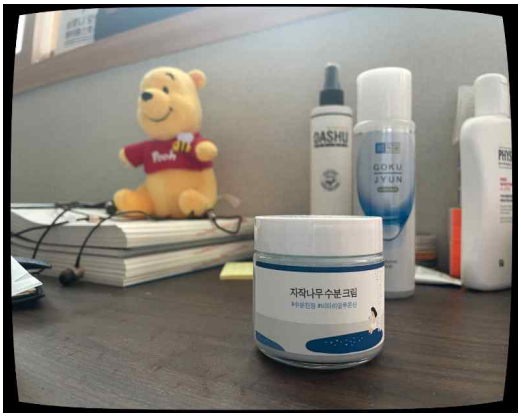
나는 앞서 말한 바와 같이 똑같은 카메라를 이용해 2차례 사진을 촬영했으며, 'right.png'는 'left.png'보다 카메라의 x방향으로 0.01m 더 이동해서 찍은 사진 파일이다. 이러한 사실에 기반하여, 사진을 찍을 때의 각 카메라 간에는 rotation이 일어나지 않았으며, x방향으로만 0.01m 만큼 traslation이 일어났음을 알 수 있다. 위 그림의 1,2번째 줄은 이를 반영한 코드이다.

코드의 중반부에서는 OpenCV 라이브러리의 stereoRectify 함수를 이용하는 것을 볼 수 있는데, 똑같은 카메라를 이용해 두 번 촬영했으므로 함수의 매개변수로 똑같은 intrinsic_matrix와 distortion coefficient가 적용된 것을 확인할 수 있다. 우리가 stereoRectify 함수를 이용하는 목적은 두 사진의 epipolar line이 서로 수평이 되도록 만들고, distortion이 사라지도록 만들기 위함인데, epipolar line이 서로 수평이 되도록 만들면, corresponding point를 더 쉽게 찾을 수 있으며, 동시에 disparity 또한 쉽게 계산할 수 있다. 이때 주의해야 할 점은 stereoRectify 함수가 곧바로 두 이미지를 rectify 해주는 것이 아니라, rectification rotation matrix와 같은 rectify 매개변수를 결과로 반환한다는 점이다.

두 이미지를 rectifiy 하는 작업은 initUndistortRectifyMap 와 remap 함수를 통해 이루어진다. initUndistortRectifyMap 매개변수로 카메라의 intrinsic matrix, distortion coefficient, rectification rotation matrix, rectified 된 좌표계에서의 3*4 projection matrix 등을 대입했으며 이를 통해 rectify에 필요한 매핑 정보가 반환된다. 그 뒤, remap 함수에서 매핑 정보를 사용해 리매핑된 이미지 데이터가 만들어지고, 이를 화면에 띄우면 최종적으로 rectify된 이미지가 나타난다.



left image를 Rectify한 결과



right image를 Rectify한 결과

(4) Disparity map 생성하기

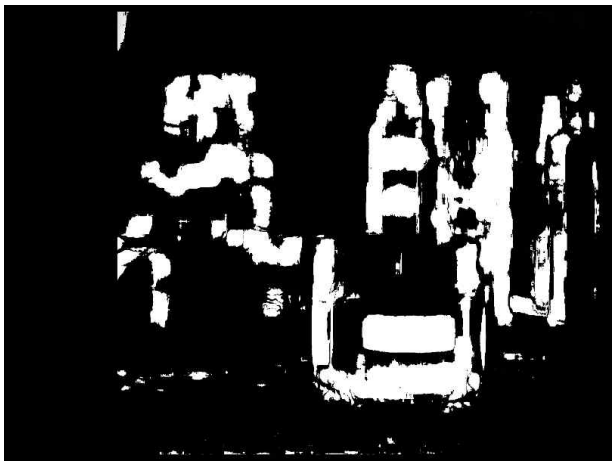
```
gray1 = cv2.cvtColor(dst1,cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(dst2,cv2.COLOR_BGR2GRAY)

stereo = cv2.StereoBM_create(numDisparities=256, blockSize = 39)
disparity = stereo.compute(gray1,gray2)
#disparity = cv2.normalize(disparity,disparity,alpha=0,beta=255,norm_type=cv2.NORM_MINMAX,
 #           dtype=cv2.CV_8U)

cv2_imshow(disparity)
```

앞서 나는 'left.png'와 'right.png'를 rectify 하는 작업을 거쳤다. 이제 rectify 된 두 이미지를 가지고 disparity map을 생성하는 코드에 대해 설명해 보겠다.

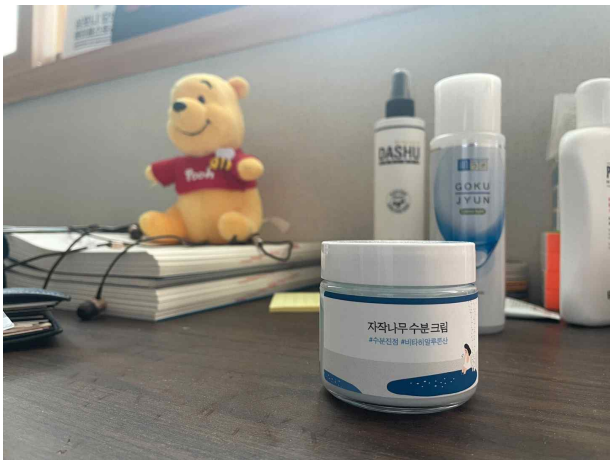
첫 번째로, rectified image들을 OpenCV의 cvtColor 함수를 이용해 흑백으로 바꾸는 작업을 한다. 이로써 gray1과 gray2 변수에는 rectified image들의 흑백 이미지 데이터가 저장된다.

다음으로 OpenCV의 StereoBM_create를 통해 stereo라는 객체를 생성했는데, 보는 바와 같이 StereoBM_create는 알고리즘 객체를 생성하는 함수라고 한다. 이 함수의 매개변수로는 numDisparities와 blockSize 라는 값이 대입되며, 'numDisparities'는 탐지할 수 있는 최대 변위의 레벨을 나타낸다. 더 자세히 말하자면, 'numDisparities' 값이 높을수록 탐지할 수 있는 이미지 간의 최대 disparity 범위가 늘어나 matching 성능이 높아지지만, 계산용량이 커지고, 해당 값이 낮을수록 반대로 matching 성능이 낮아지고 계산 속도가 빨리진다는 특징이 있다. blocksize는 매칭을 수행할 때 이용되는 블록의 크기를 나타내고 홀수 값을 대입해야 한다. 이 두 매개변수의 값을 바꾸어가며 disparity map을 생성한 결과, 값이 너무 작으면 Disparity를 잘 계산하지 못할뿐더러 Disparity map이 너무 거칠게 나오는 경향이 있었다. 하지만 값이 과도하게 큰 경우, Disparity map이 더 잘 계산되고 부드럽지만, 너무 뭉뚱그려지게 나타나는 것을 확인할 수 있었다. 따라서 나는 최종적으로 numDisparities는 256, blockSize는 39로 설정하였다.

마지막으로 StereoBM_create로 객체를 생성한 뒤, 해당 객체의 compute 함수를 이용해 disaprity map 데이터를 계산하고 'disparity' 변수에 데이터를 저장하는 작업을 하였다. 그리고 dispairty map 데이터가 담긴 'disparity'를 OpenCV 함수로 화면에 나타낸 결과가 다음과 같다.



Disparity map



left.jpg

(5) 3D Visualization

```
import math

h, w, ch = left_img.shape
colors = []
points = []

f_x = intrinsic_matrix[0,0]
f_y = intrinsic_matrix[1,1]
c_x = intrinsic_matrix[0,2]
c_y = intrinsic_matrix[1,2]
# target distance unit: meter
Z = 1.0

sampling_rate = 10

for v in range(0,h,sampling_rate):
  for u in range(0,w,sampling_rate):
    if disparity[int(v),int(u)]>0 :
      color = left_img[int(v),int(u),:]
      if color[0]==0 and color[1]==0 and color[2]==0:
        continue
      else:
        Z = 0.01 * 1.08922050e+03 / disparity[int(v),int(u)]
        if Z<0.5:
          X = (u - c_x) * Z / f_x
          Y = (v - c_y) * Z / f_y
          points.append([X,Y,Z])
          colors.append([color[2]/255.0,color[1]/255.0,color[0]/255.0])

points=np.array(points)
colors=np.array(colors)
```

3D Visualization은 camera calibration 실습 당시 이용했던 코드를 통해 구현했으며, 위 그림은 내가 일부 수정한 부분의 내용이다.

이 코드를 처음 이용할 때는 if문을 통한 disparity의 선별작업을 하지 않았었는데, 그 결과 point clouds가 정확하게 나오지 않는 것을 확인할 수 있었다. 그래서 그 원인을 찾아보았더니, 깊이를 나타내는 Z값이 음수로 계산된다는 사실을 알게 되었으며 근본적으로 disparity가

음수로 계산되어 그렇다는 것을 깨달았다. 그래서 나는 위 코드에서 볼 수 있듯이 disparity가 0보다 큰 경우에만 X, Y, Z, color를 계산하여 배열에 추가되도록 하였다. 또한, left_image의 BGR 값이 모두 0이어서 검정색인 경우 작업을 건너뛰도록 코드를 설계하였다.

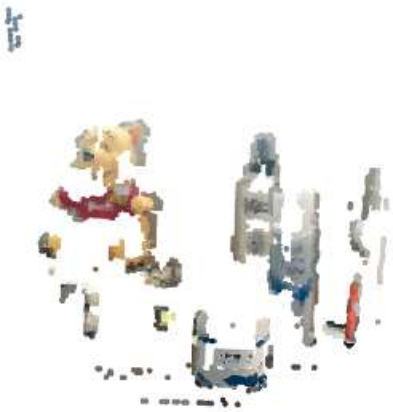
이때, Z 값은 아래 그림의 식과 같이 계산되도록 코드 구성을 하였다. D(u, v)는 disparity를 나타내며, B는 left_image와 right_image 사이의 x방향 변위차인 0.01m를 나타낸다. 마지막으로 f는 camera의 focal length로, 앞서 camera calibration 실습 코드에서 구한 값을 대입하였다. 이렇게 Z 값을 계산한 뒤, 나는 Z가 0.5 즉 50cm보다 작은 경우에만 나머지 작업이 이루어지도록 if문을 구성했는데, 왜냐하면 카메라로 사진 촬영을 했을 당시 물체와 카메라 사이의 실제 거리가 대부분 50cm 이내로 들어왔기 때문이다.

$$Z = \frac{Bf}{D(u,v)}$$
$$X = \frac{(u-c_x)Z}{f_x}$$
$$Y = \frac{(v-c_y)Z}{fy}$$

Disparity를 이용한
X, Y, Z 계산법

이처럼 나는 앞서 계산한 Disparity를 통해 point의 깊이(Z), X, Y를 계산하였으며, 이 값들을 바탕으로 point를 (X, Y) 지점에서 Z만큼 깊이 위치하도록 하여 point clouds를 구성하였다. 결과는 아래의 그림과 같다.

3D Visualization 1



3D Visualization 2