

Monocular Depth Estimation and Evaluation with LiDAR

한은기

GitHub: EunGiHan

Email: 717lumos@gmail.com

이현진

GitHub: IDIOTSAD

Email: 2sguswls2s@naver.com

유희평

GitHub: Henryy-ss

Email: hpplayer30@gmail.com

Abstract—이 보고서는 프로그래머스 자율주행 데브코스 3기 쓰리스랩 팀의 최종 프로젝트 Monocular Depth Estimation and Evaluation with LiDAR (단일 이미지로부터의 깊이 추정과 LiDAR 데이터를 이용한 평가)의 수행 과정 및 결과를 기술하고 있다. 강남 일대에서 수집된 ACE Lab Dataset을 평가 데이터로 이용하여 프로젝트를 수행하였다. Camera 이미지 데이터를 monocular depth estimation 모델인 DepthFormer에 입력하여 이미지 내 깊이를 추정하고, LiDAR point cloud data를 2차원 이미지 평면으로 투영하여, 앞서 얻은 깊이 추정치와 픽셀별로 (pixel-wise) 비교하고 정확도 및 오차 평가를 진행한다. 입력 데이터의 전처리와 깊이 추정 모델 DepthFormer의 개량을 진행하였으며, 추정 및 평가 단계의 각 산출물(point cloud data의 투영 결과와 깊이 추정치, 추정 결과 평가 등)을 저장하여 중간 및 최종 결과 산출물의 재사용성을 높였다.

1. Introduction: Project Overview

본 프로젝트는 자율주행 통합 솔루션 기업 ACE Lab의 지원을 받아 프로그래머스 자율주행 데브코스 3기 수강생들이 각 팀을 이뤄 2022년 06월 27일부터 2022년 07월 15일까지 총 3주간 진행하였다. 프로젝트의 목표는 단일 이미지로부터 깊이를 추정하고 LiDAR 데이터를 이용하여 추정치의 정확도를 평가하는 것이다. 이를 위하여 깊이 추정 알고리즘을 고안하고, LiDAR 데이터를 ground truth 지표로 이용하여 추정 결과를 평가하는 방식을 개발한다. 프로젝트의 마지막 수행일에는 약 3주간 진행한 내용과 결과물을 발표하고 보고서를 제출한다. 본 쓰리스랩 팀은 3인(한은기, 이현진, 유희평)으로 구성되었으며, 온오프라인 환경에서 협업하여 과제를 수행하였다. 과제 수행에 사용한 코드는 GitHub Repository에서 이용할 수 있다¹.

단일 이미지 깊이 추정(Monocular depth estimation)은 하나의 이미지에서 각 픽셀 위치에서의 깊이(거리, depth)를 정확히 추정하는 문제이다. 컴퓨터 비전에서 오랜 기간 연구되어 왔으며, 자율주행 자동차와 로보틱스 등에 적용되고 있다 [1]. 그러나 몇 가지 어려움이 존재하여 여전히 풀기 어려운 문제로 거론되고 있다. 무한하게 많은 3차원 이미지가 2차원으로 투영될 수 있기 때문에 깊이를 정확히 하나로 추정하기에 까다롭다. 사람의 경우, 다양한 조명 환경에서의 질감(texture)이나 시점(perspective), 알고 있는 객체에 대한 상대적 크기 등의 local cue와 장면 내 전체적 모양과 레이아웃 등의 global context를 적절히 활용하여 자연스럽게 깊이를 추정할 수 있다 [1]. 그러나 이 작업을 수행하는 컴퓨터 비전 기능을 만들기에는 단일 이미지 내 사용할 수 있는 단서(cue)가 부족하고, scale ambiguity 문제가 존재하며, 재질이 반투명하거나 반사되는 성질이 있다면 깊이 추정의 정확도가 크게 하락할 수 있다 [2].

카메라는 매우 조밀한(dense)하고 색상을 포함한 센싱 데이터를 얻을 수 있는 센서로, semantic 정보를 얻을 수 있다는 특징이 있다. 그러나 카메라는 2차원 평면으로 정보를 표현하므로 객체의 깊이 정보를 알 수는 없다는 한계가 있다. 반대로 LiDAR는 센싱 데이터의 밀도가 희박하고(sparse) semantic 정보를 얻을 수는 없으나, 레이저를 물체에 조사하여 반사광을 측정함으로써 객체까지의 깊이를 비교적 정확히 측정할 수 있다는 장점이 있다. 본 프로젝트는 위 센서들의 장단점을 융합하여 사용하기 위하여 설계되었다.

이 보고서의 2장에서는 Monocular depth estimation의 선행 연구들을 간단히 리뷰한다. 이어지는 3장에서는 주어진 dataset 내 이미지로부터 깊이를 추정하고, 추정치를 LiDAR 데이터로 평가하는 프로세스 및 상세 구현 내용을 소개한다. 4장에서는 설계한 프로세스를 ACE Lab에서 제공한 dataset을 이용하여 수행함으로써 깊이 추정 및 평가

1. <https://github.com/EunGiHan/depth-estimation-with-lidar>

를 수행하였으며, 마지막 5장에서는 프로젝트와 실험 결과 전반에 대한 결론을 제시한다.

2. Related Works

이미지에서 깊이를 추정하는 것은 곧 3차원 공간 정보를 복원하는 기술로 볼 수 있다. 전통적으로는 인간의 두 눈 (Stereo vision)이 사물을 보고 판단하는 방식에 영감을 받은 다중 시점 매칭 알고리즘을 만들고 이를 기반으로 깊이 정보를 예측하였다 [3]. Saxena 등 [4]이 최초로 DNN 방식을 적용한 것을 시작으로, 최근에는 심층학습 기반(learning based)의 깊이 추정 방식들이 주를 이뤄 연구되고 있다. Learning based depth estimation 기법들은 크게 supervised, semi-supervised, unsupervised(self-supervised) learning 방식으로 다시 나눌 수 있다 [1].

Supervised learning 방식에서는 이미지의 각 픽셀 (pixel-wise)에 대응되는 깊이 정보가 ground truth로서 주어져야 한다. 깊이 정보는 RGB-D 카메라나 다채널 레이저 스캐너의 point cloud data로부터 얻을 수 있다 [5]. Eigen 등 [6]이 합성곱(convolution) 아키텍처를 도입한 이후로 대부분의 후속 연구들이 Deep CNN 방식을 채택한 supervised learning 방식을 사용하고 있다.

최근에는 encoder-decoder 방식을 추가하여 큰 성능 향상을 이루고 있기도 하다. Encoder는 주로 이미지 분류 네트워크인 VGG, ResNet, DenseNet 등의 심층신경망을 사용하여 이미지에서 특징(features)를 추출한다. Backbone이라고도 불리며, 합성곱(convolution) 연산과 폴링(pooling) 연산을 수행한다 [7].

Decoder는 encoder에서 출력한 features를 정합(aggregate)하고, encoder에서 줄여들었던 해상도(resolution)를 복원하며, 최종적으로 픽셀별로의 깊이를 추정한다. Decoder는 주로 합성곱 연산과 up-sampling 연산이 연속적으로 이루어져 있는 구조이다 [2].

최근에는 자연어 처리(Natural Language Processing, NLP)에 성공적으로 적용되었던 Attention Mechanism을 도입하여 Transformer를 단일 이미지의 깊이 추정 작업에 적용하는 사례가 많아지고 있다. 아래에서 간단히 소개할 BTS, DPT, AdaBins, DepthFormer 등이 supervised learning 방식의 모델에 해당한다.

Semi-supervised learning 방식은 weakly supervised learning 방식으로도 불린다. 상대적 깊이를 사용하거나 [8], LiDAR 데이터를 추가적으로 이용하는 등 [9]의 연구가 진행된 바 있다.

Unsupervised learning, 혹은 self-supervised learning 방식은 ground truth의 깊이 정보를 사용하지 않고, 보정(rectified)된 stereo image 쌍만을 이용하여 깊이 추정 모델을 훈련(train)시킨다. [5]의 저자들은 MonoDepth 모델을 제안하여, 깊이 추정 문제를 이미지 복원(reconstruct) 문제로 전환하여 깊이를 추정한다.

본격적으로 해당 프로젝트를 진행하기에 앞서, 사용할 모델을 선정하기 위하여 몇 개의 깊이 추정 모델을 조사하였다. 아래는 간략한 모델별 특징을 나타냈다. Supervised learning 기반 모델과 unsupervised learning 기반 모델을 모두 포함하도록 조사 대상을 선정하였으며, 타 모델과의 두드러지는 차별점을 가졌거나 후속 연구들에서 자주 언급되는 연구, 또는 좋은 성능을 보였던 연구를 파악하고자 했다.

2.1. BTS

Lee 등이 [1]에서 제시한 모델 BTS는 Encoder-Decoder 구조를 갖는 supervised learning 모델이다. Encoder backbone으로 Dense Feature Extractor를 사용하여 feature map을 생성하며, 이 결과를 ASPP(Atrous Spatial Pyramid Pooling) 레이어에 전달해 contextual information을 추출한다. Decoding 단계에서 각 multiple stages에 LPG(Local Planar Guidance) 레이어를 사용해 large scale variation을 파악하고, 해상도를 입력 이미지 크기로 다시 복원한다. Decoder의 각 레이어들은 4차원 평면 coefficients를 각각 학습하고 그 출력을 비선형적으로 결합하여 최종 깊이 추정치를 반환한다.

2.2. DPT

Ranftl 등이 [7]에서 제안한 DPT 모델은 Encoder-Decoder 구조를 갖는 supervised learning 모델이다. Dense Prediction Transformer (DPT)를 제안하여 Encoder backbone으로 Vision Transformer(ViT)를 사용한다. 모델의 아키텍처는 크게 4 단계로 이루어져 있다. (1) 입력 이미지를 토큰(token)으로 바꾸는 embed 단계와, (2) 여러 개의 stages에서 특징을 추출하는 Transformers, (3) 여러 해상도에서 서로 다른 stages의 토큰을 합쳐 이미지 형상의 feature maps를 만드는 Rsassemble, 그리고 (4) residual convolution units를 활용해 feature maps를 합치고 up-sampling하여 최종 깊이 추정치를 반환하는 Fusion 단계이다. 모든 stages에서 global receptive field를 가지고 있어 globally coherent

한 깊이를 예측할 수 있으며, 해당 모델로 monocular depth estimation 뿐만 아니라 semantic segmentation도 수행할 수 있다.

2.3. AdaBins

Bhat 등이 [10]에서 제안하였다. BTS나 DPT 모델처럼 Encoder-Decoder 구조를 갖는 supervised learning 모델에 해당하나, 기존 연구들이 monocular depth estimation을 회귀(regression) 문제로 해결한 것과는 달리, [10]의 저자들은 이를 분류(classification) 문제로 접근하였다. 표준적인 Encoder-Decoder의 결과를 입력으로 받는 AdaBins(Adaptive Bin-width Estimator) 모듈은 Mini-ViT를 사용해 bin width b 와 Range-Attention Maps \mathcal{R} 을 계산하고, 각 scene의 features에 따라 bins를 적응적으로 바꾼다. 또한 classification을 수행함으로써 깊이 값들을 이산화(discretization)한다. Bin centers의 선형 결합으로 최종적인 깊이의 예측값을 생성한다.

2.4. DepthFormer

DepthFormer 역시 Encoder-Decoder 구조를 갖는 supervised learning 모델이며, Li 등이 [2]에서 제안하였다. 해당 모델의 큰 특징 두 가지는 (1) Transformer branch와 Convolution branch로 구성된 encoder 구조와 (2) 두 branch를 정합(aggregate)하기 위한 HAHI(Hierarchical Aggregation and Heterogeneous Interaction) 모듈이다.

합성곱(convolution) 연산만을 사용하는 기존 모델들은 CNN의 특성인 spatial inductive bias를 사용하기 때문에 global한 객체 간 관계를 놓치거나 원거리 객체의 깊이 정보를 정확히 추정하지 못하는 등의 문제가 발생했다. 이를 보완하기 위하여 Transformer와 Convolution branch로 각각 나누어 모델의 학습을 진행하여, local 정보를 탐색할 뿐만 아니라 long-range correlation도 파악할 수 있게 된다. 또한 Encoder의 ViT로 Swin Transformer [11]를 사용함으로써 계층적 특징(hierarchical features)를 추출하고 연산의 복잡도를 개선했다.

DepthFormer는 이러한 두 branch를 단순한 late fusion이 아닌 HAHI 모듈을 사용해 정합함으로써 feature을 향상시킬 뿐만 아니라 model affinity도 향상시킨다. Figure 1은 DepthFormer의 아키텍처를 간단히 나타낸 그림이다.

2.5. MonoDepth

앞서 살펴본 네 모델과는 다르게 unsupervised learning 방식을 가진다. Godard 등이 [5]에서 처음으로 제안한 모델이다. [5]의 저자들은 supervised learning 방식이 각 이미지 픽셀에 일일이 대응하는 ground truth 정보가 필요하고 데이터셋 구축에 따른 비용이 매우 크다고 지적하였다. MonoDepth의 큰 특징은 binocular stereo camera로 얻은 stereo 이미지 쌍이 주어졌을 때, 왼쪽 이미지에서 추정한 깊이 정보를 바탕으로 오른쪽 이미지를 복원하고, 이때 양 이미지 간의 시차(disparity)를 모델의 학습에 이용한다는 것이다. 또한 [5]에서는 새로운 학습 loss를 제안하여, 추정한 depth maps 간 지속성(consistency)을 강화했다.

단일 이미지에서의 깊이를 추정하는 해당 프로젝트를 진행하기 위하여 쓰리스랩 팀이 사용한 모델은 DepthFormer이다. Stereo image의 사용 가능 여부를 사전에 알 수 없다는 가정 하에, 단안카메라로 수집된 데이터셋에서도 모델을 학습하거나 구동할 수 있도록 unsupervised learning 방식보다는 supervised 방식이 적합할 것이라 판단하였다. 또한 DepthFormer는 프로젝트 수행일 기준(2022.06월 07 월) 오픈 데이터셋인 KITTI [12] Eigen Split dataset에서 AbsRel(Absolute Relative Error) 기준 3위(0.052)의 매우 성능을 보이고 있는 모델이었다. Benchmark로 KITTI Eigen Split을 사용한 이유는 본 프로젝트가 자율주행 교육 과정의 일부이며, 실내 상황보다는 다양한 실외 상황에서 강건하게 작동할 수 있어야 하기 때문이다. 모델 선택 과정 중 DepthFormer 모델을 사용하여 복수 개의 서로 다른 dataset에서 depth map을 얻는 데 성공하였으며, 데이터셋에 따른 모델의 구조 변경이 가능할 것이라 판단하였으므로 해당 모델을 선택하였다.

3. Methods

쓰리스랩 팀에서 ‘Monocular Depth Estimation and Evaluation with LiDAR’ 프로젝트 수행을 위하여 설계한 프로세스는 아래와 같다.

- (1) **사전 준비:** Calibration 정보를 포함한 주어진 dataset의 구성과 형식을 확인하고, 추가적으로 획득해야 할 정보를 정의한다. 모델을 이용하거나 추정치를 평가하기 위하여 데이터를 가공할 방법 및 형태 등을 결정한다.

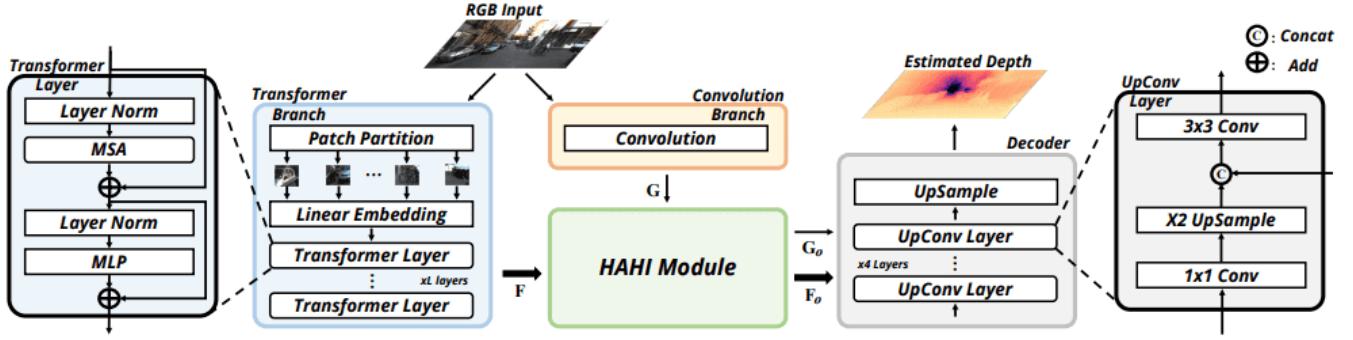


Figure 1. An overview of DepthFormer

- (2) LiDAR 데이터 변환: LiDAR의 point cloud data를 획득하고 2차원 이미지 평면으로 투영한다. 해당 값은 깊이 추정의 ground truth로 기능한다.
- (3) 단일 이미지 깊이 추정: 깊이 추정 대상이 될 이미지를 획득하고 모델에 입력하여 깊이 추정치(depth map)을 얻는다.
- (4) 추정 결과 평가: 프로세스 (2)와 (3)의 결과를 비교하여 예측의 정확도나 오차를 평가한다.

Figure 2는 위 프로세스를 도식화한 내용이며, 이어지는 내용은 각 프로세스의 세부적 내용이다.

3.1. 사전 준비

주어진 dataset을 분석하는 단계이다. 추정 대상이 되는 이미지를 비롯하여 ground truth를 만들기 위하여 이용할 수 있는 정보 등을 파악하고, 데이터셋에 따른 세부 수행 전략 등을 구상한다.

우선, dataset의 기본적 정보로서 데이터를 수집한 지역과 시점을 파악한다. 실내 환경은 실외 환경에 비해 깊이 값의 절대적 크기가 작거나 분포가 제한적이다. 반면 실외 환경은 수집된 시점과 지역에 따라서도 분포가 매우 다양하다. 깊이 추정 모델에 따라 각 환경에서의 상이한 성능을 보이는 경우가 있다. 따라서 해당 정보를 파악하는 것이 중요하다.

dataset 구축 시 사용된 센서와 그 특징 역시 파악한다. LiDAR의 종류(2D 또는 3D)와 채널의 개수는 레이저 스캐닝 결과의 density를 결정하며, 채널이 많을수록 더 조밀한 ground truth 정보를 얻을 수 있다. Camera의 종류(stereo 또는 monocular camera)는 사용할 수 있는 모델의 종류(e.g. Unsupervised learning method)나 알고리즘을 결정한다.

데이터의 형식과 크기 역시 확인한다. 본 프로젝트 수행에 필수적으로 필요한 정보인 image와 point cloud data

가 각각 어떤 포맷으로 제공되는지 확인하여, 모델에 입력하거나 서로 그 값을 비교할 때 적절한 변환과정의 필요성을 파악한다. Dataset의 크기가 매우 크면 컴퓨팅 파워의 한계로 인하여 연산 속도나 성능에 영향을 미칠 수 있기 때문에, dataset 크기 역시 사전에 파악해둔다.

또 하나 중요한 것은 캘리브레이션(calibration) 정보로, dataset을 수집한 센서들의 좌표계(coordinates)와 각 계수(coefficients)가 의미하는 바, 각종 표기법(notation)을 파악해야 한다. 캘리브레이션 정보는 카메라의 왜곡 보정, LiDAR의 3D 좌표공간에서 카메라 이미지 평면으로의 투영 등의 과정에 사용되므로 해당 값을 적절히 사용해야만 올바른 추정과 평가가 가능하다.

3.2. LiDAR 데이터 변환

두 번째 프로세스는 LiDAR의 sensing data인 point cloud data를 깊이 추정의 ground truth로 만들기 위한 전처리 단계이다. LiDAR의 raw data를 입력으로 하여, 모델에서 사용할 이미지와 동일한 크기와 해상도의 평면으로 point cloud data를 투영한 결과를 반환한다.

깊이 추정 모델의 출력은 픽셀 단위로 구한 깊이값이다. 깊이 추정 과정의 ground truth로서 point cloud data를 사용하기 위해서는 이미지 픽셀의 각 위치와 정확히 대응되는 point를 찾을 수 있어야 한다. 따라서 point cloud data를 깊이 추정 모델의 입력인 이미지와 같은 2차원 평면으로 투영한다. 모델의 출력을 3차원 좌표로 변환하지 않고 point cloud data를 2차원 평면으로 투영하는 이유는, 차원을 축소하는 연산이 차원을 확장하는 연산보다 상대적으로 단순하기 때문이다. 캘리브레이션 정보에 LiDAR-Camera 혹은 World-Camera 등 추가적 연산 없이 충분한 정보가 마련되어 있기 때문이다.

구체적 변환 과정은 이렇다. 우선, point cloud data가 담긴 raw data 파일(e.g. BIN, PCD, ROS bag 등)을 NPY파

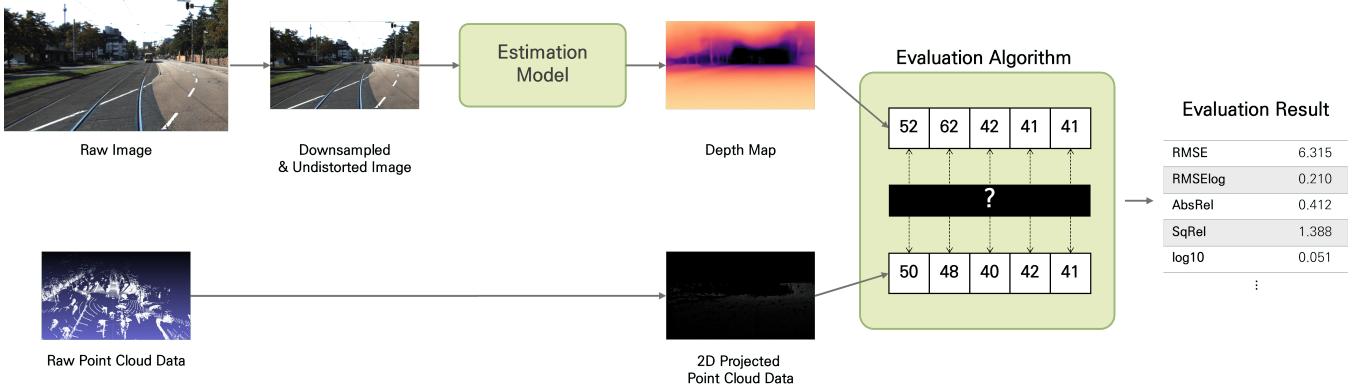


Figure 2. An overview of project process

일로 변환한다. NPY 파일은 Python에서 지원하는 Binary 파일의 일종으로, NumPy 라이브러리의 배열을 담을 수 있다. 프로젝트에 사용한 프로그래밍 언어는 Python으로, NPY 파일을 이용함으로써 Python에서 제공하는 라이브러리 NumPy를 통해 또 다른 변환 과정 없이도 손쉽게 파일로부터 데이터를 얻고 곧바로 사용할 수 있다.

주어진 캘리브레이션 정보에 따라 상이하나, 주로 LiDAR 좌표계에서 Camera 좌표계로의 변환은 rotation matrix \mathcal{R} , translation vector \mathbf{t} 과 projection matrix \mathbf{P} 등을 3 차원 공간 점의 위치에 곱하여 이루어진다.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{P} [\mathbf{R} | \mathbf{t}] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

이때 주의할 것은, LiDAR는 여러 채널을 가지고 있고 360° 회전하며 데이터를 수집하므로, 주로 정면을 향하는 카메라보다 훨씬 많은 range를 탐지하고 있다. 따라서 투영 결과에서 얻는 각 point의 좌표가 이미지 평면 바깥에 존재하거나, depth 값이 음수(-)로 후면부 깊이 데이터를 가리키고 있다면 이들은 제거해야 한다. 또한 탐지 범위 초과 혹은 미만 등의 이유로 나타나는 Nan 값 역시 제외한다.

이렇게 2차원 이미지 평면으로 투영한 결과는 또 다시 NPY 파일로 저장하고, 이미지 평면에 gray-scale로 depth 정보를 시각화하여 PNG 파일로도 저장한다. 이로써 한 프레임에 대한 추론 및 평가가 끝나거나 프로세스가 종료된 후에도 중간 산출물에 접근할 수 있으며, KITTI Eigen Split에서 제공하는 형태와 동일하게 ground truth 정보를 이미지로부터 얻고 싶을 때에도 사용할 수 있다.

3.3. 단일 이미지 깊이 추정

프로젝트 수행의 핵심 중 하나로, 주어진 하나의 이미지로부터 픽셀 별 깊이(depth)를 추정하는 단계이다. 해당 프로세스의 입력은 이미지 파일로, 주로 PNG 형식을 이용한다. 입력된 이미지는 우선 카메라 Intrinsic Matrix \mathbf{P} 를 사용하여 렌즈의 왜곡을 제거한 뒤 모델에 전달하여 추론을 진행한다. 모델의 출력이자 해당 프로세스의 결과로는 픽셀 별 깊이값, 즉 depth map을 반환한다. 2장에서 밝힌 바와 같이 쓰리스랩 팀은 단일 이미지의 깊이 추정에 DepthFormer 모델을 사용한다.

모델의 학습은 open dataset인 KITTI Eigen Split dataset으로 진행하였다. KITTI dataset은 stereo camera를 이용하여 이미지를, 3차원 다채널 LiDAR로 point cloud data를 수집하여 구축된 dataset이다. 특히 깊이 추정을 위해 가공되어 제공되는 KITTI Eigen Split dataset은 stereo pair 이미지 중 왼쪽 이미지와 이에 대응하는 위치에 깊이 값을 나타낸 ground truth 데이터를 각각 png 파일로 제공하기 때문에 학습 및 검증을 용이하게 진행할 수 있다. 본 팀은 모델의 학습에는 훈련용 데이터로 2만 6천 개의 이미지를, 검증용으로 697개의 이미지를 사용하였다. 학습은 Nvidia GTX 1080ti 2way 혹은 RTX 3080 12GB에서 진행했으며, 각각 38,000 iter에 약 12시간과 120,000 iter에 15시간이 소요되었다.

추가적으로 본 팀은 기존의 DepthFormer 모델이 사용하던 Swin Transformer (V1)를 Swin Transformer V2 [13]로 변경해 보았다. V2에서는 residual-post-norm과 scaled cosine attention을 사용함으로써 모델 용량이 깊어질수록 활성함수의 값이 급격하게 증가하는 현상을 완화시킬 수 있다. 또한 로그 공간의 continuous position bias를 사용함으로써 relative position bias가 window 간에 부드럽게 전이될 수 있도록 한다. 따라서 Swin Transformer V2로

DepthFormer의 구조를 변경한다면 성능이 개선된 V2의 사전학습 모델(pretrained model)을 해당 프로젝트에 사용할 수 있다.

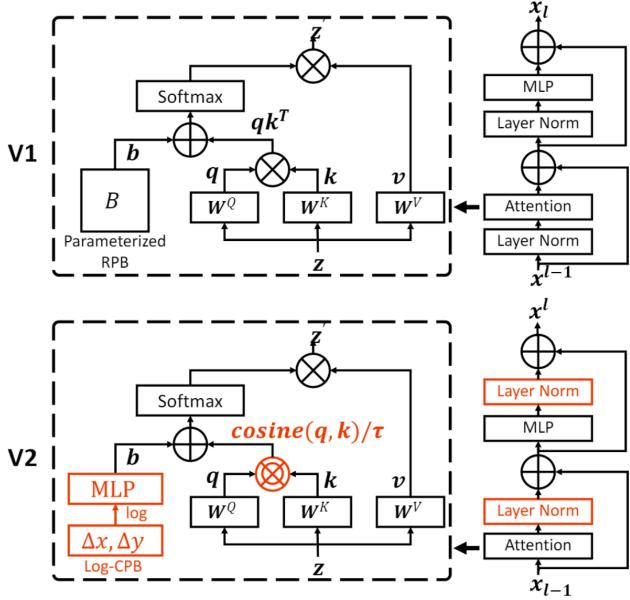


Figure 3. Swin transformer V1 and V2

그러나 SwinV2-L 사전학습 모델을 사용할 경우, VRAM 용량의 부족으로 모델의 학습을 진행할 수 없었다. 이에 대응하여 Swin Transformer V2의 구현이 잘 되었는지 확인하기 위하여 SwinV2-B를 사용하고, 이에 따라 DepthFormer 모델의 네트워크 사이즈를 축소하여 학습을 진행하였다. TABLE 1에서 Swin-L(*)은 기존 DepthFormer에서 사용한 모델에 해당한다.

Figure 4는 본 팀이 변경한 구조의 모델을 학습 및 검증했을 때의 각종 지표의 개선 추이를 나타낸 것이다.

3.4. 추정 결과 평가

LiDAR 데이터 변환 과정과 단일 이미지 깊이 추정 과정의 결과로 각각 깊이 추정치(depth map)과 ground truth 값을 얻었다. 이들은 픽셀별(pixel-wise)로 비교가 가능한 형태이며, 각종 metrics를 활용해 추정의 정확도와 오차를 계산해 평가를 진행할 수 있다.

한 이미지 내 i 번째 픽셀에서의 깊이 추정치를 y_i , 실제 깊이값(i.e. point cloud의 투영된 형태)을 y_i^* 라고 할 때, 아래의 7개의 평가 지표를 사용해 추정 결과를 평가한다. 해당 지표들은 여러 monocular Depth Estimation 모델들의 성능 지표로서 주로 사용되는 지표로 선정하였으며, 이들은 KITTI Eigen Split의 평가 지표이기도 하다.

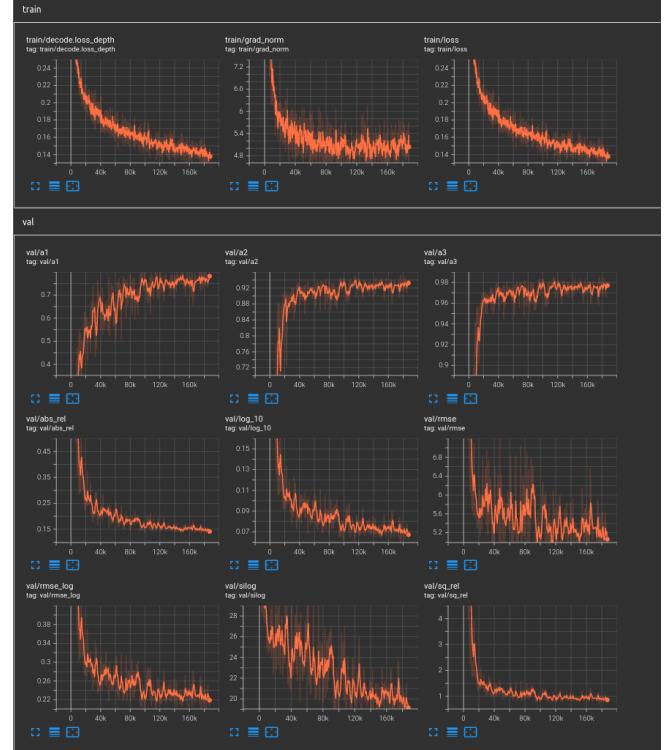


Figure 4. 변경한 구조의 DepthFormer 성능 지표

Threshold는 정확도를 측정하는 지표이며, 나머지 6개인 RMSE, RMSElog, SILog, AbsRel, SqRel, log10은 오류를 측정하는 지표에 해당한다. Threshold는 높을수록, 나머지 지표는 낮을수록 좋은 성능이라 할 수 있다.

- **Root Mean Squared Error(RMSE)**는 추정값과 실제값의 차이를 다루는 대표적 척도로, 정밀도(precision) 파악에 적합하고 이상치(outlier)에 비교적 강건(robust)하다.

$$\sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - y_i^*)^2}$$

- **Root Mean Squared Log Error(RMSElog)**는 RMSE와 유사한 방식이나 이를 log scale에서 비교를 한다. RMSE보다 이상치에 강건하여 지표의 변동폭이 크지 않다. Log의 빨惛은 나눗惛으로 전환될 수 있다는 특성 때문에 RMSElog는 상대적 오차를 측정한다. 추정값이 실제값보다 작을 때, 즉 under estimation 상황에서 over estimation 상황보다 더 큰 페널티를 부과한다. 이 역시 log 함수가 양의 실수 공간에서 우상향하고 빨惛이 나눗惛으로

name	pretrain	resolution	acc@1	acc@5	#params
Swin-L(*)	ImageNet-22K	224x224	86.3	97.9	197M
Swin-L	ImageNet-22K	384x384	87.3	98.2	197M
SwinV2-B	ImageNet-22K	384x384	87.1	98.2	88M
SwinV2-L	ImageNet-22K	256x256	86.9	98.0	197M
SwinV2-L	ImageNet-22K	384x384	87.6	98.3	197M

TABLE 1. PERFORMANCE OF SWIN TRANSFORMER PRETRAINED MODEL

로 전환될 수 있다는 특성 때문이다.

$$\sqrt{\frac{1}{N} \sum_{i=1}^N (\log y_i - \log y_i^*)^2}$$

- **Scale-invariant Log-arithmetic Error(SILog)**는 [14]에서 제안된 방식으로, 값의 scale과 무관하게 두 점 간의 오차를 계산하기 위하여 고안되었다. $\alpha(y, y^*) = \frac{1}{n} \sum_{i=1}^N (\log y_i^* - \log y_i)$, $d_i = \log y_i - \log y_i^*$ 일 때,

$$\begin{aligned} & \frac{1}{2N} \sum_{i=1}^N (\log y_i - \log y_i^* + \alpha(y, y^*))^2 \\ & \frac{1}{n} \sum_{i=1}^N d_i^2 - \frac{1}{n^2} \left(\sum_{i=1}^N d_i \right)^2 \end{aligned}$$

- **Threshold**는 오차가 $thr = \alpha^t$ ($t = 1, 2, 3$, $\alpha = 1.25$) 이내인 픽셀의 비율 $\delta_1, \delta_2, \delta_3$ 을 말한다.

$$\% \text{ of } y_p \text{ s.t. } \max\left(\frac{y_p}{y_p^*}, \frac{y_p^*}{y_p}\right) = \delta < thr$$

- **Absolute Relative Error(AbsRel)**은 추정값과 실제값의 상대적 차이를 평균한 것으로, 실제값 대비 상대오차를 파악하는 데 쓰인다.

$$\frac{1}{N} \sum_{i=1}^N \frac{|y_i - y_i^*|}{y_i^*}$$

- **Squared Relative Error(SqRel)** 역시 AbsRel와 유사한 방식을 사용하며, 추정값과 실제값의 절댓값 차이가 아닌, 차이의 제곱을 평균한다는 데서 차이를 지닌다. 비교적 이상치에 민감한 성격을 지닌다.

$$\frac{1}{N} \sum_{i=1}^N \frac{|y_i - y_i^*|^2}{y_i^*}$$

- **Mean Log10 Error(log10)**은 추정값과 실제값에 각각 log을 취해 그 오차의 합을 평균한다.

$$\log_{10} = \frac{1}{N} \sum_{i=1}^N |\log_{10} y_i^* - \log_{10} y_i|$$

한 이미지에 대하여 위 7개의 지표를 각각 구할 수 있으며, 추가적으로 같은 dataset 내 모든 프레임(이미지)에 대해서 평균을 내어 경향성을 평가한다. 해당 평가 결과 역시 텍스트 파일(txt)로 저장하여 추후 분석에 바로 사용할 수 있도록 구현하였다.

no.	thr1	thr2	thr3	RMSE	RMSELog	SILog	AbsRel	SqRel	log_10
0900	0.979	0.992	0.997	7.660	0.373	37.332	0.280	2.074	0.125
0901	0.975	0.992	0.996	7.608	0.383	38.179	0.302	2.143	0.132
0902	0.974	0.991	0.996	7.408	0.390	38.694	0.316	2.267	0.134
0903	0.972	0.991	0.996	7.673	0.406	40.068	0.343	2.520	0.143
0904	0.972	0.990	0.996	8.012	0.417	41.176	0.351	2.639	0.147
0905	0.973	0.991	0.996	8.361	0.423	42.209	0.339	2.736	0.147
0906	0.974	0.989	0.995	8.736	0.440	43.880	0.333	2.895	0.150
0907	0.972	0.988	0.994	8.874	0.463	45.835	0.351	3.140	0.160

Final Report									
Bag	thr1	thr2	thr3	RMSE	RMSELog	SILog	AbsRel	SqRel	log_10
01	0.976	0.988	0.993	12.063	0.541	52.444	0.353	4.332	0.171

Figure 5. 개별 프레임 및 전체 데이터셋에 대한 평가 결과 예시

4. Experiments

3장에서 설명한 프로젝트의 프로세스를 ACE Lab에서 제공한 dataset을 이용해 수행하였다.

4.1. 사전 준비

서울 강남 일대에서 80 채널 3D LiDAR와 HD Camera로 수집된 데이터로 구축된 데이터셋을 이용한다. 이미지와 LiDAR point cloud data를 포함한 dataset으로, 캘리브레이션 계수들과 notation, 좌표축의 관계, 캘리브레이션에 사용한 체커보드 이미지 등 필요한 정보가 추가적으로 제공되었다. Dataset은 총 7개의 ROS bag 파일로 이루어져 있으며, 각 파일의 녹화 시간과 크기는 TABLE 2와 같다.

file	duration (s)	size (GB)
gangnam_1.bag	90	9.5
gangnam_2.bag	37.1	3.9
gangnam_3.bag	49	5.2
gangnam_4.bag	44.6	4.7
gangnam_5.bag	44.2	4.6
gangnam_6.bag	53.7	5.6
gangnam_7.bag	79	8.4

TABLE 2. ACE LAB DATASET ROSBAG FILES

이미지 데이터는 sensor_msgs/Image ROS msg을 가지며 약 30 Hz로 publish 되고, point cloud data는 sensor_msgs/PointCloud2 msg로 약 10 Hz로 publish 된다. 이미지 데이터가 약 3배 정도 더 많다는 것을 알 수 있다.

본 팀은 깊이 추정 및 평가를 실시간으로 구현하지 않는다. 따라서 프로세스 수행 전에 rosbag 파일에서 필요한 이미지 및 point cloud data를 미리 추출하여 저장하였다. LiDAR의 publish 속도(약 10 Hz)가 카메라(약 30 Hz)보다 낮기 때문에 synchronize를 위해 LiDAR의 timestamp에 대응하는 Camera topic을 추출하였다. Point cloud data는 NumPy 배열로 변환하여 NPY 파일로, 이미지는 PNG 파일로 저장한다.

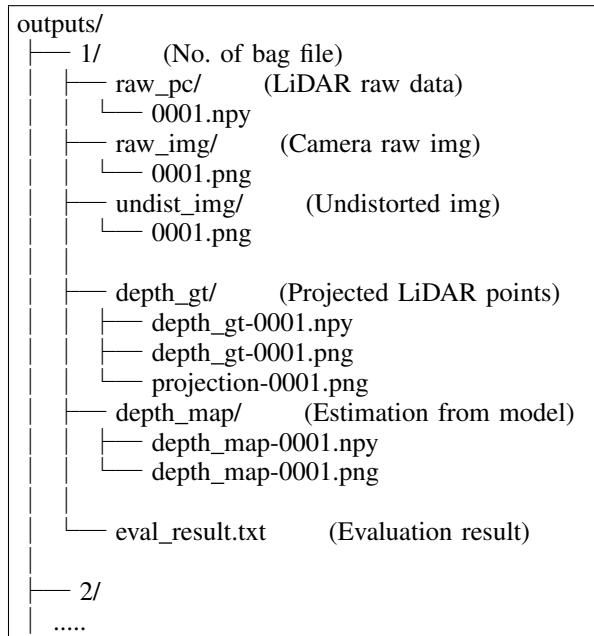


Figure 6. 쓰리스랩 팀이 ACE Lab dataset에 사용한 directory 구조

4.2. LiDAR 데이터 변환

Ground truth 데이터를 만들기 위하여 NPY 파일을 읽어 포인트 클라우드 데이터를 획득하고, dataset에서 제공하는 캘리브레이션 정보로 point cloud data를 2차원 이미지 평면으로 투영한다. 해당 dataset에는 차량(ego vehicle)을 비롯한 카메라와 LiDAR의 좌표계와, Camera-Ego 및 LiDAR-Ego의 변환을 위한 계수가 제공되었다.

Coefficient	Description
D	Distortion Coefficients
K	Intrinsic Matrix
P	Projection Matrix
R	Rotation matrix (ego-camera, lidar-ego)
t	translation vector (ego-camera, lidar-ego)
size	size of image

TABLE 3. CALIBRATION COEFFICIENTS FOR ACE LAB DATASET

Raw point cloud data(NPY 파일)는 각 점 당 LiDAR 좌표계 기준의 3차원 좌표(X, Y, Z)와 반사강도(intensity) 값을 가지고 있다. NPY 파일을 불러와 각 위치 좌표가 Nan 이 아닌 점들만 추린다. 그리고 LiDAR 좌표계에서 Ego Vehicle 좌표계로 각 점의 좌표계를 전환한다.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{ego} = [\mathbf{R}|\mathbf{t}] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_{lidar}$$

이렇게 얻어진 3차원 Ego Vehicle 좌표계의 점을 2차원 카메라의 이미지 좌표계로 투영한다.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_{im} = \mathbf{P} [\mathbf{R}|\mathbf{t}] \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{ego}$$

위 식을 통해 최종적으로 이미지 평면에서의 각 점의 좌표 (u, v)와 그곳에서의 깊이값 s를 얻을 수 있다. 이때, 이미지 평면의 좌표는 정수이므로 나눗셈의 소수점 이하 단위는 버림한다.

ACE Lab dataset의 이미지의 경우, 그 크기가 Depth-Former 모델에 그대로 사용하기에 매우 크다. 따라서 원본 이미지의 가로와 세로를 각각 절반으로 하는 down-sampling을 진행하는데, 줄어드는 이미지의 크기에 맞추어 ground truth 정보도 정합되어야 하므로 각 좌표 (u, v)에 2를 나눈 값을 저장한다.

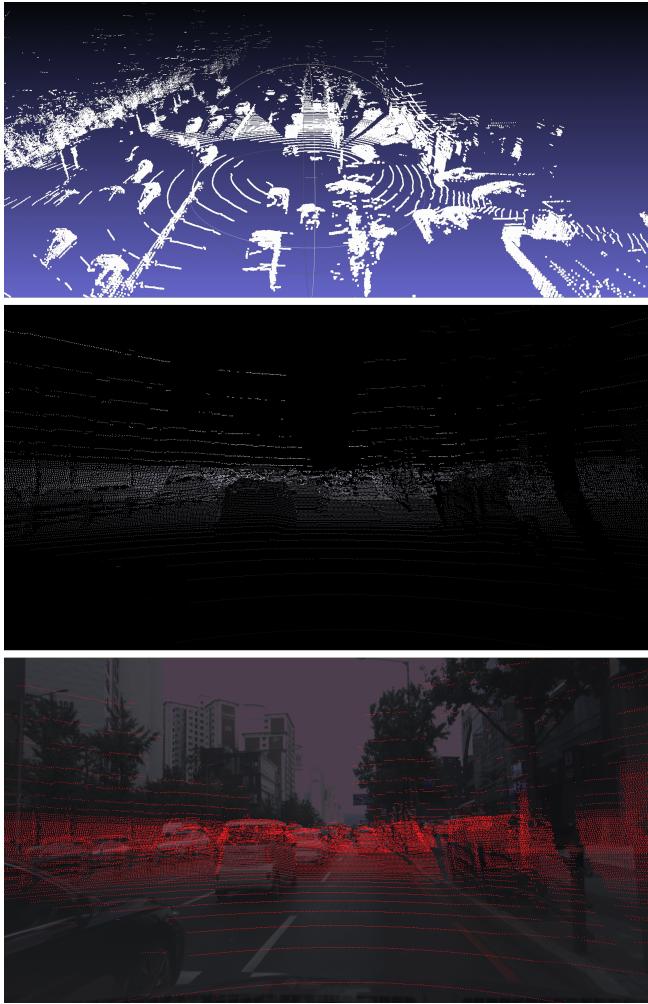


Figure 7. (위) LiDAR Point Cloud Data (중간) Point Cloud Data의 이미지 평면으로의 투영 결과 (아래) 이미지와 투영 결과

4.3. 단일 이미지 깊이 추정

ACE Lab의 rosbag 파일로부터 얻은 raw image data를 왜곡을 보정한 뒤 모델에 입력해 깊이 추정치인 depth map을 얻는 단계이다. 그러나 앞서 밝힌 바와 같이 이미지 원본 데이터는 DepthFormer의 입력으로 사용하기에 메모리 부족 이슈를 일으켰다. 따라서 모델 입력 직전에 이미지의 down-sampling을 먼저 진행하여 가로와 세로 크기를 절반씩 줄인다. Down-sampling은 OpenCV의 cv2.resize() 함수를 사용하였으며, 이미지의 사이즈를 축소할 때 주로 사용하는 보간 방식인 INTER_AREA를 인자로 전달하였다.

앞서 3장에서 밝힌 바와 같이, 본 팀은 기존 DepthFormer의 Swin Transformer를 V1에서 V2(SwinV2-B)로 변경하였다. 그러나 ACE Lab dataset에 대하여 깊이 추정 정확도가 V1보다 현저히 낮았다. 때문에 ACE Lab data에 대한 추정은 V1을 사용한 기존 모델을 학습시켜 진행하였다.

이어지는 추가적 학습 및 연구에서 정확한 이유를 파악하고 정량적 성능 비교를 진행하고자 한다.

모델의 학습은 3장에 서술한 바와 같이 오픈 dataset인 KITTI Eigen Split를 사용했다. 비록 훈련 dataset의 수집 장소가 검증 dataset의 수집 장소인 강남 일대와 큰 차이가 있으나, 공통적으로 야외 도로 상황에 대한 데이터셋을 포함한다는 점에서 큰 무리가 없을 것이라 판단하였다. Figure 7와 8은 입력 이미지와 그 이미지로부터 얻은 depth map 출력 결과 예시를 나타내었다.

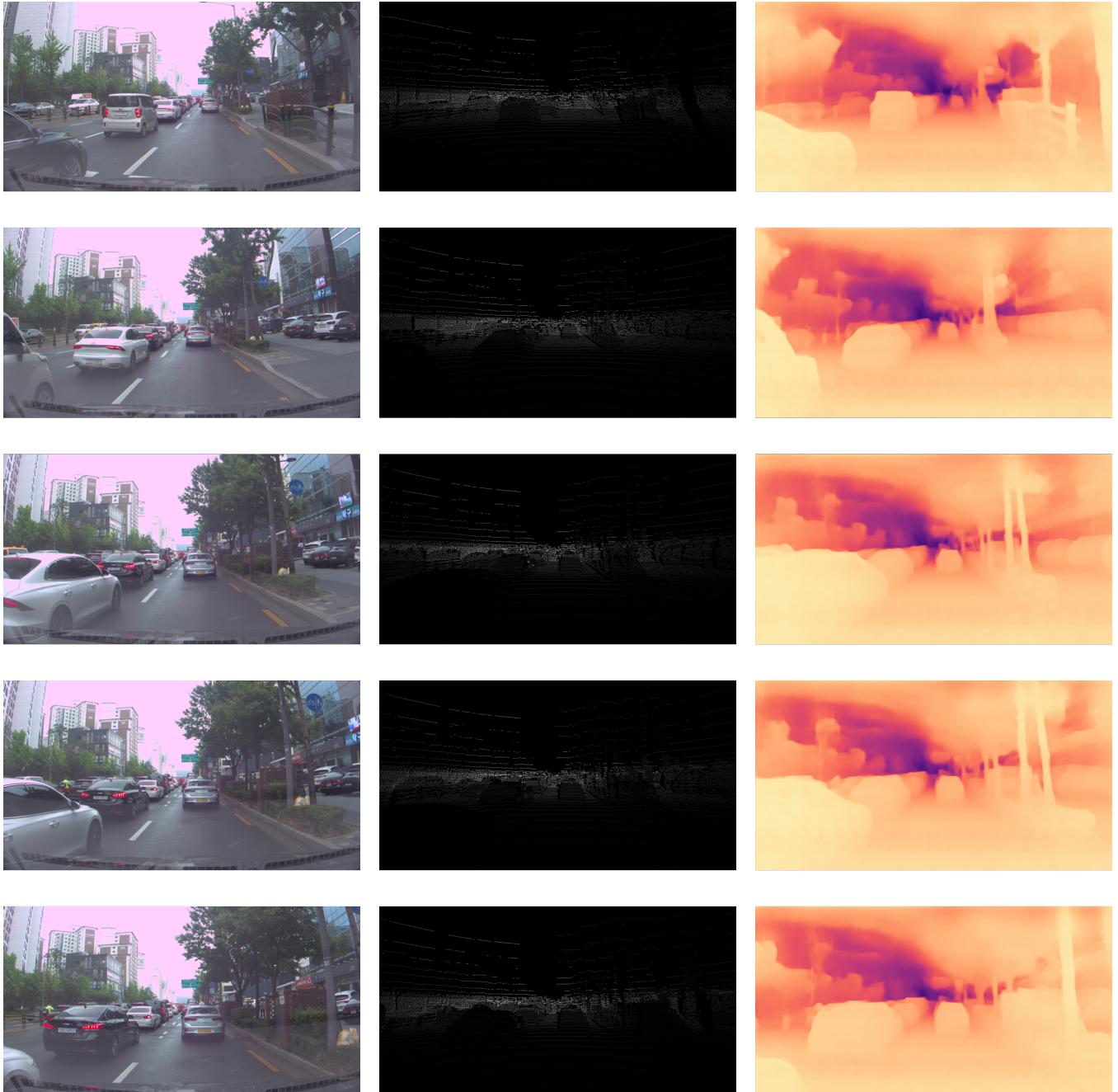


Figure 8. (위) Down-sampled and undistored image (아래) Depth map 결과

4.4. 추정 결과 평가

주어진 7개의 rosbag 파일 별로의 정확도 및 오차 평가 결과는 TABLE 4와 같다. 앞서 밝힌 바와 같이 RMSE, REMElog, SILog, AbsRel, SqRel, log10, Threshold로 총 7개의 지표를 사용하였다. 각 rosbag 파일 내 프레임별로 구한 수치를 프레임 전체 개수로 나누어 bag 파일 별 정확도 및 오차를 계산하였다.

이때, DepthFormer 모델에서 설정한 최대 추정 깊이 (max depth)가 80m이고, 깊이 값이 음수라면 후방 객체를 탐지한 결과이므로, $0 \leq \text{depth} \leq 80$ 범위를 벗어나는 데이터는 제외하고 평가를 진행한다. 이를 Radius Outlier Remover 방식이라고 한다.



Raw Image

Point Cloud (GT)

Depth Map

Figure 9. Raw Image data와 Point cloud data(projected, ground truth), Depth Map 예시

bag file	$\delta_1 \uparrow$	$\delta_2 \uparrow$	$\delta_3 \uparrow$	RMSE \downarrow	RMSElog \downarrow	SILog \downarrow	AbsRel \downarrow	SqRel \downarrow	log ₁₀ \downarrow
gangnam_1.bag	0.583	0.787	0.872	12.080	0.542	52.714	0.357	4.406	0.172
gangnam_2.bag	0.558	0.843	0.938	9.953	0.408	39.977	0.320	3.249	0.140
gangnam_3.bag	0.663	0.881	0.940	9.381	0.446	43.600	0.290	2.668	0.142
gangnam_4.bag	0.572	0.831	0.930	11.396	0.463	45.319	0.333	3.891	0.154
gangnam_5.bag	0.556	0.728	0.831	13.487	0.582	54.104	0.355	4.999	0.187
gangnam_6.bag	0.546	0.842	0.934	11.132	0.398	38.742	0.301	3.437	0.135
gangnam_7.bag	0.665	0.805	0.874	16.061	0.497	47.352	0.263	4.907	0.139

TABLE 4. EVALUATION RESULT FOR ACE LAB DEPTH ESTIMATION TEST SET

5. Conclusion

5.1. ACE Lab Dataset 깊이 추정 및 평가

본 프로젝트는 하나의 이미지에서 깊이 정보를 추정하고 이 결과를 LiDAR 포인트 클라우드 데이터를 이용해 정확도를 판단한다. 쓰리스랩 팀은 dataset 내 3차원 포인트 클라우드 데이터를 2차원 이미지 평면으로 투영하여 ground truth로 삼고, 이미지에서 얻은 픽셀 별 깊이 추정치를 비교함으로써 정확도 및 추정 오차를 판단했다.

7개의 dataset은 대체적으로 비슷한 결과를 보였다. 같은 dataset을 이용한 benchmark가 없어 직접적 비교는 불가능하나, KITTI Eigen Split을 대상으로 시험한 타 모델의 결과[2]와 비교하면, 본 팀의 모델이 약 0.3배(e.g. δ_1)에서 5배(e.g. RMSElog) 정도의 차이를 보였다. Ground truth의 품질과 dataset의 크기, 모델 학습의 정도 등 기존의 연구들이 더 좋은 환경 및 조건을 가지고 있었다는 점을 감안할 때, 모델의 학습 시간이 짧고 학습에 사용한 데이터의 크기가 작은 등 부족한 조건임에도 비교적 준수한 평가 결과를 보여주고 있음을 알 수 있었다.

기존 DepthFormer 모델에 비해 본 팀의 결과가 비교적 큰 오차를 보이는 이유는 여러 가지로 추측해볼 수 있다. 가장 먼저, ground truth로 사용하는 LiDAR 포인트 클라우드 데이터가 카메라 이미지에 비해 매우 sparse하고 이를 투영하는 과정에서 노이즈가 포함되거나 투영 오차가 발생했을 수 있다. 이 문제는 깊이 추정 결과를 평가할 때 오차의 원인이 모델의 추정 오류에 있는지 애초에 정답(ground truth)이 잘못되었는지 알 수 없다는 데 문제가 있다.

모델의 입력인 이미지를 down-sampling한 여파도 고려해야 한다. 각 픽셀마다 저장된 RGB 정보로부터 깊이 정보를 추정하는데, down-sampling하며 정보가 소실되거나 추정 과정에 유의미한 영향을 줄 수 있다. 포인트 클라우드 데이터 역시 단순한 1/2 연산으로 down-sampling된 이미지와 크기를 맞추는데, 이 역시 해당 픽셀의 정확한 실제 위치로 투영되는지 보장할 수는 없을 것이다.

5.2. 추가적 학습 및 연구

[2]에서는 깊이의 range를 0-20m, 60-80m, 그 이상으로 나누어 모델의 성능을 추가적으로 검증하였다. Convolution의 연산 특성 상 spatial inductive bias를 사용하므로 원거리의 객체에 대한 정보를 파악하기 힘들거나 깊이 추정의 정확도가 떨어지는 현상이 발생하기도 한다. 따라서 Transformer를 변형한 본 팀의 모델이 각 범위에 따른 정확도에 편차가 있는지를 점검할 수 있을 것이다.

학습 데이터인 KITTI Eigen Split dataset과 검증 dataset인 ACE Lab Dataset은 수집 환경이 매우 다르다. 전자의 경우 일반 도로 뿐만 아니라 주택가, 자동차전용도로 등 교외지역의 다양한 상황을 담고 있으며 독일에서 수집되었다. 반면 후자의 경우 대부분 복잡한 강남 일대의 도로 상황을 담고 있으며, 교외와는 다른 도시에서 주행하고 있다. 해외 자료를 국내 환경에 적용하는 데 문제점은, 교차로나 이면도로, 곡선로가 많은 국내 도로 환경에 적용하기 어렵다는 점이다. 따라서 더 높은 정확도를 위하여 도시 도로 상황을 대상으로 한 dataset(e.g. Waymo Open Dataset)이나 한국 도로 상황을 담은 dataset으로 추가적 모델 학습을 진행할 수 있을 것이다.

본 프로젝트에서는 깊이 추정 모델로 지도학습을 이용하는 DepthFormer를 사용했으나, 지도학습의 단점은 거대한 ground truth 정보가 필요하다는 점이며, 특히 깊이 정보는 ground truth 데이터를 얻기 매우 어렵다. 깊이 정보를 포함하는 자율주행 dataset도 희귀하기 때문에, 깊이 ground truth 정보 없이 stereo camera만을 이용하는 dataset을 이용하여(e.g. 42dot Open Dataset) 학습을 시킬 수 있는 비지도 학습 모델을 사용해 추가적 학습을 진행하고 성능 향상을 노려볼 수도 있다.

Appendix

쓰리스랩 팀은 원활한 협업과 코드 품질 향상을 위하여 GitHub의 Actions 기능을 이용하여 CI(Continuous Integration) 기능을 구현하였다. Python의 code formatter인 Black과 isort를 적용하여 코드의 포맷을 검사하고 자동으로 교정하여 저장한다. Figure 10는 문자열의 최대 길이 및 따옴표의 사용 규칙에 따라 자동으로 코드를 교정하는 예시를 보여주고 있다.

```

111 00 -0.29 441.11 00
112 111 dataset = "ace"
113 111     if dataset == "ace":
114 111         lidar_left = np.loadtxt('Lidar-visual-Gullioneder/`camer-a`/front'
115 111             # cam.calib["w"] = calib.info["w"] # a distortion coefficients (0, 1)
116 111             # cam.calib["w"] = calib.info["w"] # (3, 3)
117 111             # cam.calib["w"] = calib.info["w"] # camera calibration matrix (3, 3)
118 111             # cam.calib["w"] = calib.info["w"] # projection matrix (3, 4)
119 111             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
120 111             # cam.calib["w"] = calib.info["w"] # translation (3, 1)
121 111             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
122 111             # cam.calib["w"] = calib.info["w"] # projection matrix (3, 4)
123 111             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
124 111             # cam.calib["w"] = calib.info["w"] # translation (3, 1)
125 111             # cam.calib["w"] = calib.info["w"] # image size (height: 1080, width: 2040)
126 111             # cam.calib["w"] = calib.info["w"]
127 111             # cam.calib["w"] = calib.info["w"]
128 111             # cam.calib["w"] = calib.info["w"]
129 111             # cam.calib["w"] = calib.info["w"]
130 111             # cam.calib["w"] = calib.info["w"]
131 111             # cam.calib["w"] = calib.info["w"]
132 111             # cam.calib["w"] = calib.info["w"]
133 111             # cam.calib["w"] = calib.info["w"]
134 111             # cam.calib["w"] = calib.info["w"]
135 111             # cam.calib["w"] = calib.info["w"]
136 111             # cam.calib["w"] = calib.info["w"]
137 111             # cam.calib["w"] = calib.info["w"]
138 111             # cam.calib["w"] = calib.info["w"]
139 111             # cam.calib["w"] = calib.info["w"]
140 111             # cam.calib["w"] = calib.info["w"]
141 111             # cam.calib["w"] = calib.info["w"]
142 111             # cam.calib["w"] = calib.info["w"]
143 111             # cam.calib["w"] = calib.info["w"]
144 111             # cam.calib["w"] = calib.info["w"]
145 111             # cam.calib["w"] = calib.info["w"]

111     else:
111         lidar_left = np.loadtxt('Lidar-visual-Gullioneder/`camer-a`/front'
112             # cam.calib["w"] = calib.info["w"] # a distortion coefficients (0, 1)
113             # cam.calib["w"] = calib.info["w"] # (3, 3)
114             # cam.calib["w"] = calib.info["w"] # camera calibration matrix (3, 3)
115             # cam.calib["w"] = calib.info["w"] # projection matrix (3, 4)
116             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
117             # cam.calib["w"] = calib.info["w"] # translation (3, 1)
118             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
119             # cam.calib["w"] = calib.info["w"] # projection matrix (3, 4)
120             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
121             # cam.calib["w"] = calib.info["w"] # translation (3, 1)
122             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
123             # cam.calib["w"] = calib.info["w"] # projection matrix (3, 4)
124             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
125             # cam.calib["w"] = calib.info["w"] # translation (3, 1)
126             # cam.calib["w"] = calib.info["w"] # image size (height: 1080, width: 2040)
127             # cam.calib["w"] = calib.info["w"]
128             # cam.calib["w"] = calib.info["w"]
129             # cam.calib["w"] = calib.info["w"]
130             # cam.calib["w"] = calib.info["w"]
131             # cam.calib["w"] = calib.info["w"]
132             # cam.calib["w"] = calib.info["w"]
133             # cam.calib["w"] = calib.info["w"]
134             # cam.calib["w"] = calib.info["w"]
135             # cam.calib["w"] = calib.info["w"]
136             # cam.calib["w"] = calib.info["w"]
137             # cam.calib["w"] = calib.info["w"]
138             # cam.calib["w"] = calib.info["w"]
139             # cam.calib["w"] = calib.info["w"]
140             # cam.calib["w"] = calib.info["w"]
141             # cam.calib["w"] = calib.info["w"]
142             # cam.calib["w"] = calib.info["w"]
143             # cam.calib["w"] = calib.info["w"]
144             # cam.calib["w"] = calib.info["w"]
145             # cam.calib["w"] = calib.info["w"]

111     return cam_calib
112

```

Figure 10. Black Formatting Result

또한 각 문서 및 함수에 documentation을 삽입하여 코드 파악 및 협업을 용이하게 하였다.

```

116 def project_point(dataset: str, lidar_point: np.ndarray, cam_calib: dict, lidar_calib: dict):
117     """get all lidar [X, Y, Z] data and project them to image plane
118
119     Args:
120         dataset (str): dataset(ace / kitti)
121         lidar_point (numpy.ndarray): a lidar point with [x, y, z] format
122         cam_calib (dict): camera calibration
123         lidar_calib (dict): lidar calibration
124
125     Returns:
126         numpy.ndarray: coordinate & depth in [row(x), col(y), depth] format
127
128     Notes:
129         * ACE Calib
130             * lidar [R|t]: lidar coord -> ego coord (3D)
131                 * 센서는 더 높이 더 앞쪽으로 밀려 있으므로, 보정 값은 깊이&높이는 커지고 좌우는 큰 변화 없음
132                 * camera P X |R|t: ego coord -> cam coord
133                 * 원점 절차: lidar -> ego -> cam
134         * KITTI Calib
135             * cam to cam: 4개의 카메라가 있어 서로 비워는 calib 정보가 있음
136             * velo to cam: 라이다를 카메라(0번)으로 바로 투영
137
138     lidar = np.append(lidar_point, [1], axis=0) # [x(forward) y(left) z(up) 1]
139     lidar = np.transpose(lidar) # (4, 3)
140
141     if dataset == "ace":
142         # lidar coordinate -> ego coordinate
143         ego = np.concatenate([lidar_calib["R"], lidar_calib["t"]], axis=1) # [R|t] matrix
144         ego = np.concatenate([ego, [0, 0, 0, 1]], axis=0) # (3, 4) -> (4, 4)
145         ego = np.matmul(ego, lidar) # [R|t] X [x y z 1]^T -> [x(forward) y(left) z(up)] (3, 1)

116     else:
117         lidar_left = np.loadtxt('Lidar-visual-Gullioneder/`camer-a`/front'
118             # cam.calib["w"] = calib.info["w"] # a distortion coefficients (0, 1)
119             # cam.calib["w"] = calib.info["w"] # (3, 3)
120             # cam.calib["w"] = calib.info["w"] # camera calibration matrix (3, 3)
121             # cam.calib["w"] = calib.info["w"] # projection matrix (3, 4)
122             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
123             # cam.calib["w"] = calib.info["w"] # translation (3, 1)
124             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
125             # cam.calib["w"] = calib.info["w"] # projection matrix (3, 4)
126             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
127             # cam.calib["w"] = calib.info["w"] # translation (3, 1)
128             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
129             # cam.calib["w"] = calib.info["w"] # projection matrix (3, 4)
130             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
131             # cam.calib["w"] = calib.info["w"] # translation (3, 1)
132             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
133             # cam.calib["w"] = calib.info["w"] # projection matrix (3, 4)
134             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
135             # cam.calib["w"] = calib.info["w"] # translation (3, 1)
136             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
137             # cam.calib["w"] = calib.info["w"] # projection matrix (3, 4)
138             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
139             # cam.calib["w"] = calib.info["w"] # translation (3, 1)
140             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
141             # cam.calib["w"] = calib.info["w"] # projection matrix (3, 4)
142             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
143             # cam.calib["w"] = calib.info["w"] # translation (3, 1)
144             # cam.calib["w"] = calib.info["w"] # rotation matrix (3, 3)
145             # cam.calib["w"] = calib.info["w"]

116     return ego
117

```

Figure 11. Black Formatting Result

Python 전용 Unit Test인 PyTest 라이브러리를 이용하여 Unit Test도 진행하였다. 이로써 함수의 구현 과정에서 예외사항이 없는지 확인하고 안정적 작동 여부를 확인할 수 있었다. Figure 12은 point cloud data 하나에 대하여 2 차원 이미지로의 투영 함수를 테스트한 결과로, 이미지 크기를 벗어난 점에 대해서는 FAIL 결과를 옳게 출력하는 것을 볼 수 있다.

References

- [1] Jin Han Lee, Myung-Kyu Han, Dong Wook Ko, and Il Hong Suh. From big to small: Multi-scale local planar guidance for monocular depth estimation. *arXiv preprint arXiv:1907.10326*, 2019.
- [2] Zhenyu Li, Zehui Chen, Xianming Liu, and Junjun Jiang. Depthformer: Exploiting long-range correlation and local information for accurate monocular depth estimation. *arXiv preprint arXiv:2203.14211*, 2022.
- [3] Seong-Hun Im. 인공지능 기반 3 차원 공간 복원 최신 기술 동향. *Broadcasting and Media Magazine*, 25(2):17–26, 2020.
- [4] Ashutosh Saxena, Sung Chung, and Andrew Ng. Learning depth from single monocular images. *Advances in neural information processing systems*, 18, 2005.
- [5] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 270–279, 2017.
- [6] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE international conference on computer vision*, pages 2650–2658, 2015.
- [7] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12179–12188, 2021.
- [8] Weifeng Chen, Zhao Fu, Dawei Yang, and Jia Deng. Single-image depth perception in the wild. *Advances in neural information processing systems*, 29, 2016.
- [9] Yevhen Kuznetsov, Jorg Stuckler, and Bastian Leibe. Semi-supervised deep learning for monocular depth map prediction. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6647–6655, 2017.
- [10] Shariq Farooq Bhat, Ibraheem Alhashim, and Peter Wonka. Adabins: Depth estimation using adaptive bins. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4009–4018, 2021.
- [11] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.
- [12] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.
- [13] Ze Liu, Han Hu, Yutong Lin, Zhiliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, et al. Swin transformer v2: Scaling up capacity and resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12009–12019, 2022.

```

platform win32 -- Python 3.9.7, pytest-6.2.4, py-1.10.0, pluggy-0.13.1 -- C:\Users\lumos\anaconda\python.exe
cachedir: pytest_cache
rootdir: C:\Users\lumos\Desktop\depth-estimation-with-lidar, configfile: pytest.ini
plugins: asyncio-2.2.0
collected 4 items

test/test_code.py::TestCalib::test_convert_pcd_to_xy: ..... test session starts .....
INFO    2022-07-08 13:41:36 logger::test_code.py::test_convert_pcd_to_xy:93: test_convert_pcd_to_xy
INFO    2022-07-08 13:41:36 logger::test_code.py::test_convert_pcd_to_xy:93: live log call
INFO    2022-07-08 13:41:36 logger::test_code.py::test_convert_pcd_to_xy:93: test_convert_pcd_to_xy
INFO    2022-07-08 13:41:36 logger::test_code.py::test_convert_pcd_to_xy:93: PASSED
test/test_code.py::TestCalib::test_in_image: ..... live log call .....
INFO    2022-07-08 13:41:36 logger::test_code.py::test_in_image:98: test_in_image
INFO    2022-07-08 13:41:36 logger::test_code.py::test_in_image:98: live log call
INFO    2022-07-08 13:41:36 logger::test_code.py::test_in_image:98: test_in_image
INFO    2022-07-08 13:41:36 logger::test_code.py::test_in_image:98: PASSED
test/test_code.py::TestCalib::test_projection: ..... live log call .....
INFO    2022-07-08 13:41:36 logger::test_code.py::test_projection:103: test_projection
INFO    2022-07-08 13:41:36 logger::test_code.py::test_projection:108: input point(xyz) : [-10.85400009 -14.15200043 -1.55200005]
INFO    2022-07-08 13:41:36 logger::test_code.py::test_projection:109: output point(xy) : [-170. 455.]
INFO    2022-07-08 13:41:36 logger::test_code.py::test_projection:110: depth: -12.34593545512293
INFO    2022-07-08 13:41:36 logger::test_code.py::test_projection:110: live log call
INFO    2022-07-08 13:41:36 logger::test_code.py::test_project_all_points:116: test_project_all_points
INFO    2022-07-08 13:41:36 logger::test_code.py::test_project_all_points:116: live log call
INFO    2022-07-08 13:41:36 logger::test_code.py::test_project_all_points:116: logger.info("output point(xy): " + str(projected_pos[-1]))
INFO    2022-07-08 13:41:36 logger::test_code.py::test_project_all_points:116: logger.info("depth: " + str(projected_pos[-1]))
E     assert_in_image(projected_pos, cam_calib["size"]) # 범위 벗어나면 Fail
E     AssertionError: assert False
E     + where False = in_image(Array([-170. , 455. , -12.34593545512293]), {'height': 1086, 'width': 2040})
test/test_code.py:112: AssertionErron
INFO    logger::test_code.py:103 test_projection
INFO    logger::test_code.py:108 input point(xyz) : [-10.85400009 -14.15200043 -1.55200005]
INFO    logger::test_code.py:109 output point(xy) : [-170. 455.]
INFO    logger::test_code.py:110 depth: -12.34593545512293
===== short test summary info =====
FAILED test/test_code.py::TestCalib::test_projection - AssertionError: assert False
1 failed, 3 passed in 9.22s =

```

Figure 12. An example of Unit Test

- [14] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. *Advances in neural information processing systems*, 27, 2014.