



An Analysis on the Competitive Capabilities of Stereo Vision Cameras to Generate Pointclouds and Estimate 3D Object Locations: Stereo-Point Cloud 3D Object Detection

Masters Thesis in Mechatronics

Kristian Gonzalez, September 2019

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science (M.Sc.).

Supervisor: Prof. Dr. Wolfgang Ertel
Ravensburg-Weingarten University of Applied Sciences
Co-supervisor: Prof. Dr. Stefan Elser
Ravensburg-Weingarten University of Applied Sciences

Abstract

Artificial intelligence is a rapidly growing field that has come to be the core of the future industry of autonomous driving. In order to detect the world around the vehicle, lidar sensors have played a key role in giving distance information to compliment camera and inertial data. Lidar sensors, however, are still expensive, slow, and not widely available. This issue may be addressed by attempting to obtain the same information via other sensors. In this paper, the performance of a stereo-camera sensor for distance sensing is assessed. In order to obtain these results and comparison, the KITTI dataset was used as the source of the information as well as a rough benchmark against other methods.

This paper demonstrates an offline proof-of-concept stereo-based 3D object detection network, which will be referred to as Stereo Point Cloud net, or SPCLnet. It is offline because there are multiple stages that must be performed sequentially, rather than in a single all-in-one training step that may then run on a live datastream. The network is a composite of two well known networks: Pyramid Stereo Matching net (PSMnet) [2] and Frustum Point net (FPnet) [20]. A pair of images are fed into PSMnet, which then creates a disparity map. Next, the disparity map is reconstructed into a pointcloud using epipolar geometry. Finally, the pointcloud and the original left-side image is given to FPnet to create a 3D bounding box estimate for each detected class. In this paper, cars were the main focus, although pedestrians and cyclists could also be trained for and detected. This overall approach coincidentally imitates a paper that was published in the same timeframe that this thesis was being researched and investigated, calling this a “Pseudo-LiDAR” network [26]. The authors’ results are also reviewed and explained in-depth here.

The results from SPCLnet were encouraging, although it is clear that the performance offered is only competitive against lidar in near-ranged detection. This is also confirmed by other stereo-based networks. For the primary class of interest, cars, detection performance reached a 47% AP (average precision) at 30 meters and closer. As the number of detections are expanded to include farther objects, performance decreases [26]. These results and related conclusions are further explored in this paper. The author concludes that stereo vision has potential, but the quality of the generated stereo map is still the limiting factor. Real-time stereo-based object estimation is well within reach, and given the hardware and software available today.

Declaration of Originality

Hereby I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and all used resources are indicated in the list of references.

Contents

1	Introduction	1
1.1	Background	1
1.2	Motivation	1
1.3	Related Work	5
1.3.1	Stereo Vision	5
1.3.2	3D estimation with Stereo Disparity Maps	6
1.3.3	3D Estimation with Lidar	7
2	Pyramid Stereo Network and Disparity Map Reconstruction	9
2.1	Pyramid Stereo Matching Network Architecture	9
2.2	3D Reconstruction from Stereo Data	12
3	Frustum Pointnet and Point Clouds	16
3.1	Frustum Pointnet Architecture	16
3.2	Modification of Frustum Point Net	18
4	Experiments	20
4.1	Performance Metrics	20
4.1.1	Defining an Outcome	20
4.1.2	Intersection Over Union	21
4.1.3	Precision & Recall	21
4.1.4	Calculating Average Precision	22
4.2	Experiment 1: Lidar-Based Performance as a Baseline	23
4.3	Experiment 2: Stereo-Based Performance, using Stereo Dataset	24
4.4	Experiment 3: Stereo-Based Performance, using Object Detection Dataset	26
4.4.1	Similarity of Ground Truths Across Datasets	26
4.4.2	Results of Modification	27
4.5	Comparison with Other Networks	30
5	Conclusions and Future Work	32
A	Performance Metrics	34
A.1	Intersection Over Union	34
A.1.1	Example: IOU of a Ground Truth and Prediction Label	34
A.2	Precision & Recall	35
A.3	Calculating Average Precision	39
B	About the KITTI Dataset	40
B.1	Introduction	40
B.2	The Stereo Dataset	40
B.3	The Object Detection Dataset & Statistics	42
B.4	Evaluation Difficulty Levels	45

1 Introduction

1.1 Background

This thesis was performed alongside work and research done for the SMART3D project. The SMART3D project originally started in 2016 as an investigation into the feasibility of using a Photonic Mixing Device (PMD) sensor as an alternative to lidar in detecting 3D objects in short-range outdoor applications. Although PMD sensors initially showed promise, the lack of a publicly available dataset motivated the shift to stereo cameras, marking the start of this thesis. This work seeks to address and answer the how well a stereo-based 3D object detector performs, especially in comparison to lidar, the natural and primary choice for estimating an object's location in an outdoor environment (such as detecting other cars on the highway or navigating a crowded path). Stereo vision, specifically on vehicles, typically consists of a 2-camera setup using passive color light detection to generate an estimated stereo disparity map. This may then be transformed into a point cloud and subsequently analyzed with a Pointnet object detector. In this thesis, the performance of such a system, or neural network, is evaluated as well as compared with that of similar detection networks.

This thesis was born out of the SMART3D project, and thus the two have similar but slightly different goals. It must also be noted that an important and relevant paper was published by Wang, Chao, Garg, Hariharan, Campbell, and Weinberger [26] during the time the SMART3D project had already begun and before the completion of this thesis. This means that although the inspiration for stereo-based 3D object estimation came from two separate sources, this paper is technically following in similar footsteps, and indeed draws some ideas from the “Pseudo-LiDAR from Visual Depth Estimation” paper. The overlap of the two will be acknowledged when appropriate throughout this paper.

The SMART3D project itself is an investigation into short-range, outdoor sensor alternatives to lidar, originally based around PMD sensors. However, as mentioned above, the lack of PMD sensor information in a publicly available dataset led to the use of stereo sensor information instead. The motivation for stereo cameras as a lidar alternative is given below.

1.2 Motivation

The field of artificial intelligence has exploded in recent years, in part due to the usage of convolutional neural networks (CNN's) and the usage of graphics processing units, GPU's. This has enabled further research into systems that can quickly understand their surroundings, having applications in other fields, one of which is autonomous driving. The subfield of autonomous driving has seen great success in the application of camera data for 2D localization. However, driving is a process that requires some 3D knowledge, which means the usage of lidar has followed the growth of this sub-field. To its credit, lidar is a technology with multiple benefits: a lidar scan is precise, works outside, works in darkness, and provides immediate metric information about the world. Unfortunately, lidar sensors also have the disadvantage of being rather expensive when compared to passive sensors such as cameras. In this context, expensive means the relative amount of usable information that is output at a standardized rate per amount of money spent. There are a few factors to normalize, and

a more in-depth analysis of this is conducted below. In addition to the high cost, there may be other hidden disadvantages of using lidar, such as a weaker or missing return signal on reflective surfaces, potential interference of multiple lidar sensors if they're all in the same area (e.g. multiple autonomous vehicles at an intersection), and the relatively slow refresh rate lidar sensors can achieve when compared to passive sensing cameras. Therefore, despite lidar being a worthy technology of use, this paper seeks to explore an alternative technology, stereo vision, to estimate the 3D positions of objects.

There are some interesting benefits that may be gained with stereo vision, because stereo cameras are relatively cheap for the amount of information they provide, intuitively mimic the way that humans already perceive and navigate their world, are passive sensors that do not interfere with other sensors, and have a high refresh rate (dependent on the camera system, but 10 FPS, “frames per second”, is around the lowest value for any camera). Of course, using a stereo vision system is not without downsides, including: inability to function at night without adequate lighting, difficulty understanding large texture-less surfaces, and an error that increases with distance. This increasing error is described by Wang et al. [26] and visualized below in Figure 1.2.

Lidar dominates the environmental sensing space for some very good reasons. It is primarily used for its accuracy at intermediate distances, in the 5-100 meter range. It also is capable of sensing in a 360° field of view, and can continue detection during low-light conditions. However, most modern lidar sensors have a rolling shutter; when the lidar is moving and collecting data, each scan becomes deformed and must be corrected. They also typically cost much more than a stereo camera system, require moving parts (thus becoming sensitive to vibration), and the other reasons mentioned previously.

In order to make a more quantitative comparison between lidars and stereo cameras, the hardware setup used in the KITTI dataset will be analyzed. There are several object classes to choose from, but this paper will focus specifically on the “Car” class; cars are labeled far more often in this dataset compared to other classes, and have the inherent benefit of having a generally similar shape to each other. In collecting the necessary information to create the various datasets, Geiger, Lenz, and Urtasun [6] used a Velodyne HDL-64E lidar and two PointGrey Flea 2 color cameras. In 2012, when the paper was first published, an HDL-64E typically cost in the range of 75,000 USD (US dollars) and above, and the cameras had a cost in the range of 700 USD. Therefore, the two-camera system will be assumed to have cost 1,400 USD. For comparison, these prices will be normalized for point cloud density and quality. See Table 1.1 below for more information on both sensor setups.

A numeric comparison may be made between sensors, in an attempt to correctly account for several features that one may have while the other may not. Ideally, one would want to have both a high quantity of information as well as a high quality of information. To start, the “quantity” argument will be examined. Because lidar has 360° vision and cameras do not, the camera price and points per scan will be multiplied by four to simulate having cameras on each side of the car, thus simulating near-360° vision of the environment (although in the end this does not affect any ratios). Additionally, from scan to scan in the KITTI dataset, there is a varying number of points in the lidar point cloud. Thus, an average of the number

of points in each lidar scan was obtained.

A quick note about the camera FOV (field of view): although the stated raw FOV is 90° by 35°, there is some angular reduction due to both rectification and cropping necessary to standardize image sizes for disparity calculation. Thus, it will be assumed that although a single stereo camera system does not have its original FOV dimensions, it may have at least 90% of the original values.

Table 1.1: Economic comparison of KITTI dataset sensors, including a theoretical four-system stereo setup. hFOV refers to “horizontal field of view”, vFOV for vertical, cost is in estimated US dollar price in 2012, points/scan refers to how many points a sensor provides for each scan, and angular area is in “square radians” (elaborated below), describing the visible area based on a sensor’s vertical and horizontal FOV’s. The “Stereo (x4)” column simulates as-if a single stereo setup were to be used to obtain near-360° vision of the environment, imitating lidar.

Property	Lidar	Stereo (x1)	Stereo (x4)
hFOV (deg)	360	81	324
vFOV (deg)	20.0	31.5	31.5
Cost (USD)	75,000	1,400	5,600
Points/Scan	476,898	453,376	1,813,504
Angular Area (rad^2)	2.18	0.77	3.07
Points/Price (pts/USD)	6.36	323.84	323.84
Points/Area (pts/rad^2)	218,761	588,800	588,800

To properly clarify the table, “angular area” will be explained. Angular area, solid angle, square degrees, etc, is a value in “square radians” (rad^2) describing the visible region from a given viewpoint. Solid angle may be generally defined as given in Equation 1.1. In this equation, A is the spherical surface area, and r represents the sphere’s radius. Given that a sphere’s surface area is $4\pi rad^2$, the maximum theoretical solid angle any sensor may have is 4π , or about 12.57 square radians. However, since only vertical and horizontal field of view are given, the formula to calculate solid angle is then modified to Equation 1.2. Here, θ and ϕ represent the horizontal and vertical FOV, respectively. The spherical coordinate system used along with an example graphic is shown in Figure 1.1. As an example, the lidar system, with hFOV of [0, 360] degrees ([0, 2π] rad) and vFOV of [-10, 10] degrees ([−0.175, 0.175] rad), may thus be calculated to have an solid area of about 2.18 rad^2 .

$$\Omega = \frac{A}{r^2} \quad (1.1)$$

$$\Omega = \frac{1}{R^2} \int_{\theta} \int_{\phi} R^2 \cos(\phi) d\theta d\phi = (\theta_{b1} - \theta_{a1}) [\sin(\phi_{b2}) - \sin(\phi_{a2})] \quad (1.2)$$

Given the information above, stereo sensors are the clear winner in terms of points per dollar (points/price) as well as point density (points/area).

On the question of quality, lidar becomes the de facto standard of measure due to its high accuracy. This means that stereo vision quality is derived from how closely it matches a lidar scan. This also means that stereo vision quality is highly dependent on the system used, and can vary with both hardware and software implementation. A typical engineering threshold of 10% is used for comparison, meaning that a single stereo-generated point is considered “good”

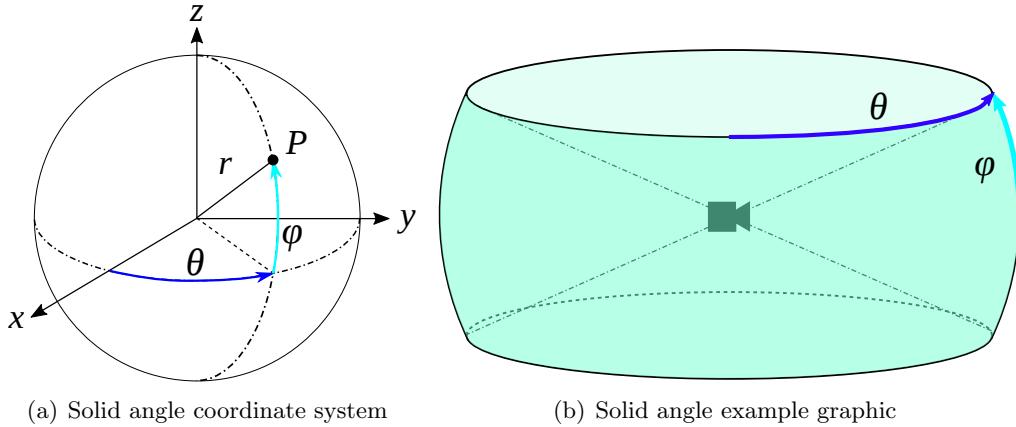


Figure 1.1: (a) Spherical coordinate system for Equation 1.2. (b) Example graphic demonstrating where angle values correspond to sensor. The transparent green region represents the sensor's visible field.

if its depth estimate is within $\pm 10\%$ of the absolute value of the corresponding lidar depth estimate. The horizontal and vertical distances, x and y , are ignored due to the assumed 1:1 correspondence when projecting a point cloud onto the image plane. The steps to determine this quality are:

1. Project lidar points to image space, the resulting matrix named L
2. Convert disparity values of stereo map to depth values, named S
3. Keep only points from stereo map that also exist in L
4. Aggregate depth pairs for as many images to compare as desired
5. For each ($i = \text{row}, j = \text{column}$) index, calculate relative error with reference to lidar as given in Equation 1.3.
6. Bin all results in 10 meter increments, relative to lidar values. In each subgroup, get the fraction of points within the bounds relative to the total number of points in the subgroup

$$e = \left| \frac{S_{i,j} - L_{i,j}}{L_{i,j}} \right| \quad (1.3)$$

After performing the following steps, Figure 1.2 was generated: A bar graph of binned ranges and the fraction of stereo points in each range that within error tolerance of the reference lidar value.

In summary, stereo vision is an attractive alternative to lidar vision because of its much higher point density, lower price, reasonable quality approximation of lidar, and the potential to improve through further research. Under the correct conditions, stereo vision sensors have the potential to compete with lidar sensors in terms of both quantity and quality.

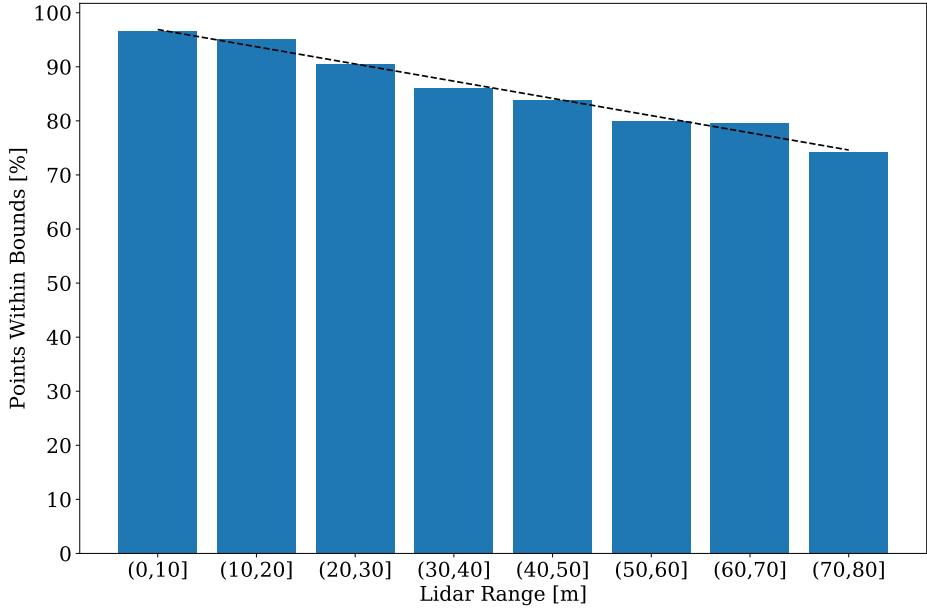


Figure 1.2: A bar chart describing what percentage of points lie within the given depth range. As an example, about 95% of stereo depth values greater than 10 meters but less than or equal to 20 meters are within $\pm 10\%$ of the corresponding lidar depth value. The first 1000 indices of the KITTI dataset were used, altogether comprising 17,134,011 point comparisons between stereo and lidar. The dashed line indicates a 1st order polynomial fit, with the slope equal to about -0.03 percentage points per range (i.e. every 10 meters, there's about -0.03 change in percentage of points within bounds). In the smallest subset of points, the (70,80] range, there are 75,224 points. The fully developed Pyramid Stereo Matching network for use in Experiment 3 was used to generate part of the data.

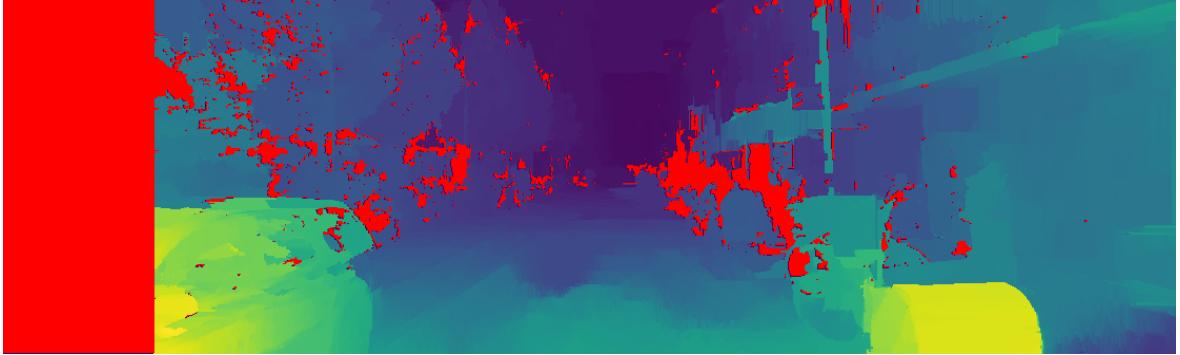
1.3 Related Work

1.3.1 Stereo Vision

Stereo disparity maps are built by comparing the pixel distance between two similar regions of two images. Generating a disparity map, however, was developed before the popularization of neural networks. The key step that most neural networks have been used to improve is matching cost computation [2], one of the four typical steps as described by Scharstein [23]. Out of this paper, the four main parts of conducting stereo vision have been classically defined as follows: matching cost computation, cost (support) aggregation, disparity computation and optimization, and disparity refinement.

Initially, built-in and readily available stereo disparity map computation techniques were used. In the Python OpenCV library, there is a “Semi-global Block Matching” (SGBM) algorithm, implemented based on the paper by Hirschmuller [8]. For an initial “rough” pass, this algorithm was cheap and fairly effective. Additionally, this algorithm is useful for fast, fairly-stable calculations and does not need a GPU to run. However, it suffers from a few key shortcomings. For one, there is a large rectangular portion of the disparity map that contains no values, as can be seen in the left side of Figure 1.3(a), where missing values are marked red. Next, when a set of pixels cannot be matched between images, the result is simply a value of zero, giving no information in that region; these locations are also marked in red. Finally, SGBM suffers from a significant lack of resolution, meaning that some objects of interest, such as the three pedestrians in the image (highlighted with red rectangles in the

color image), will not be seen well, or at all. For comparison, stereo images created with a neural network (NN) such as those in Figure 4.10 have a richer amount of information and have no missing pixels (i.e. a prediction is made at every pixel).



(a) SGBM disparity map



(b) LHS image

Figure 1.3: An example of a disparity map generated without a neural network approach. Though it has good merit, there are some noticeable shortcomings, with missing values colored red, uneven steps in value (car on left), and low resolution (three highlighted pedestrians not visible in map). This disparity map was generated in part by the left-side color image. Index 15 of the KITTI Object Detection dataset.

With this knowledge, the next step was to find an ideal NN-based stereo disparity estimation method. Notable stereo neural network approaches to disparity estimation include Pyramid Stereo Matching network (PSMnet) [2], iResNet [13], among others on the various stereo benchmarks. In order to select the best candidate, multiple benchmarks were considered. The first benchmark was naturally the KITTI Stereo challenge, which contains a list of the top performing networks. At the start of this thesis writing, PSMnet stood in the top 10 of stereo networks in this list [17].

1.3.2 3D estimation with Stereo Disparity Maps

There is currently a low amount of public research on stereo vision for use with 3D localization. Most approaches are typically derived from a lidar-based approach. Zeng, Hu, Liu, Ye, Han, Li, and Sun [30] produced a stereo alternative to their original “Real-Time 3D Vehicle Detection” network, yet had a difficult time adjusting the performance to account for the noise present in a stereo-generated point cloud, as their approach first transformed points

into a bird’s eye view (BEV), then performed detections. This approach essentially works on a 2D image of the desired area, using well-established 2D localization methods, but from a top-down view. Also working in 2D, but with a more traditional approach, Li, Chen, and Shen [12] takes the left and right images from each scene and obtains region proposals for both sides, eventually taking each proposal pair and estimating semantic information like “object class, stereo bounding boxes, dimension”, and so on. This approach does have its benefits, being deeply rooted in more traditional 2D localization techniques, such as those provided by Ren, He, Girshick and Sun [22], but still experiences shortcomings due to working with three-dimensional data in a two-dimensional space. Additionally, some prior information is required, named as being a “pre-set dimension prior” that is used for “dimension prediction”. In contrast, this thesis and the state of the art approaches avoid the need to have prior information about the world. Overall, the network seemingly has some difficulty learning to estimate information in at least one of the dimensions. In contrast to these previous approaches, the state of the art is currently achieved by Wang et al. [26] in an approach that reconstructs a stereo image into a point cloud, then manipulating the data in a three-dimensional space. Since the start of this paper, Wang et al.’s work became public knowledge, and was then published before this paper was completed. In the paper, “Pseudo-LiDAR from Visual Depth Estimation”, the authors took an extremely similar approach to this paper. Their network architecture takes a stereo image pair, generates a disparity map via Pyramid Stereo Matching network, converts it to a “pseudo-lidar” scan, and extracts 3D bounding box estimations via Frustum Pointnet. This development unfortunately has a strong overlap with the contents of this paper, yet did not have origins in short-range detection of objects. However, in their paper, the authors argue that within a short distance, up to 30 meters away, stereo vision performs competitively with lidar, which is encouraging for the work performed here.

1.3.3 3D Estimation with Lidar

Performing 3D estimation and localization with lidar has been a focus for a large portion of the field, and with good reason. Lidar data, while expensive to obtain, currently provides the best option for 3D information. Thus, there are many papers which focus on using lidar sensor capabilities to estimate 3D bounding boxes, to varying degrees of success. As mentioned previously, Zeng et al. [30] attempted to use lidar information with a novel 2D BEV approach, yet fell short of really taking advantage of the new dimension that lidar provided. The greatest breakthroughs have indeed been by networks that manipulate the information directly in the 3D space. Notably, Volumetric CNN’s such as by Wu, Song, Khosla, et al. [27] were one of the first leaps from 2D space to 3D space, using voxels created by the 3D information provided. However, this approach ultimately struggles with data sparsity and the massive computational cost of 3D convolution. Similar to the first two approaches named, multi-view CNN’s such as by Su, Maji, Kalogerakis, and Learned-Miller [24] attempt to use multiple 2D projections of the 3D space to estimate object locations. These also fail, namely when attempting to perform finer operations such as shape classification. One of the major successes in 3D object detection has been by Qi, Su, Mo, and Guibas [19] in the development

1 Introduction

of Pointnets, a network that could directly and scalably “summarize an input point cloud by a sparse set of key points”. This had a built-in level of robustness, because the network was designed to operate on an orderless set of points, with less-than-complete knowledge of the 3D shape. However, an improvement came by the same authors when “Frustum Pointnet” [20] was introduced. This network simplified the classification process by first using a 2D object detector as a filter to isolate a region of the point cloud. This cone, or “frustum”, then allowed the Pointnet network to operate on a substantially smaller set of points, reducing computation time. This paper takes advantage of this capability to great effect, as explained in Section 3.

2 Pyramid Stereo Network and Disparity Map Reconstruction

The network developed over the course of this thesis, called Stereo Point Cloud Network (SPCLnet) can be broken up into two main sub-networks, as shown below in Figure 2.1. The first sub-network is explained here, and the second sub-network is explained in the next section.

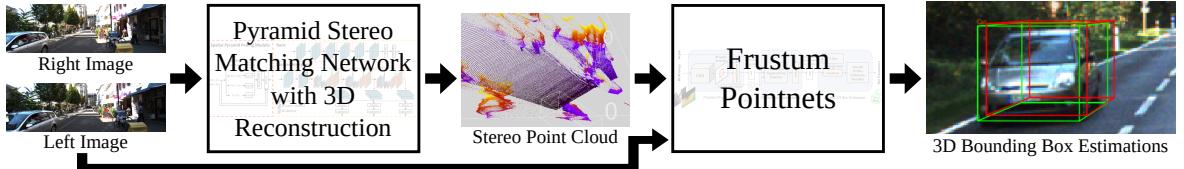


Figure 2.1: General architecture of Stereo-Pointcloud network. A stereo image pair is fed to a disparity estimation network, specifically PSMnet, which then also reconstructs the disparity map into a 3D point cloud. The point cloud is then fed into a pointnet (along with the original left image), specifically Frustum Pointnet, which finally returns a 3D bounding box estimation. Images reproduced from [6], [2], and [20].

The first sub-network is used to generate a stereo-based point cloud from a disparity map estimation network, created by Chang and Chen [2]. The majority of the following information is derived from their work. This sub-network consists primarily of a “Spatial Pyramid Pooling module” to look for matching information at various levels of detail, as well as a “stacked hourglass module for cost volume regularization”. After the sub-network is described, a brief explanation of reconstruction is also provided.

2.1 Pyramid Stereo Matching Network Architecture

The first task in creating a network that can take a stereo image pair and generate 3D bounding boxes, as described in Figure 2.1 above, is to have the capability of taking a stereo image pair and creating a disparity map. To that end, a best-in-class disparity generation algorithm was searched for and selected to be the Pyramid Stereo Matching Network (PSMnet). This network takes a deep learning approach to generating disparity maps from a pair of images. The network itself is near the top of the state of the art, and achieves this by the architecture of its network. The openly-available repository provided a foundation to start on towards making a compact, easy-to-use function.

First, the original evaluation code was tested out. Some modification and housekeeping had to be done to ensure that it was compatible with the local installation, as well as being adapted for use with Python 3. All deprecated functionality was manually removed. Finally, a self-made set of ground-truths were eventually generated in order to train the network on the KITTI Object Detection dataset instead of the Stereo dataset. This was done by projecting lidar points onto the image plane, removing points outside image bounds, and multiplying appropriate factors as outlined in the development kit. This is also clarified in Section 4.4.1.

Pyramid Stereo Matching Network, or PSMnet, is a network that takes advantage of an hourglass-shaped network. The result are very high quality, high resolution disparity

maps. The network takes advantage of GPU processing power, depending on NVIDIA’s GPU acceleration, similar to other networks. The network itself is written in PyTorch, a flexible and python-based network module which enabled simplified debugging. PSMnet’s architecture is shown below in Figure 2.2.

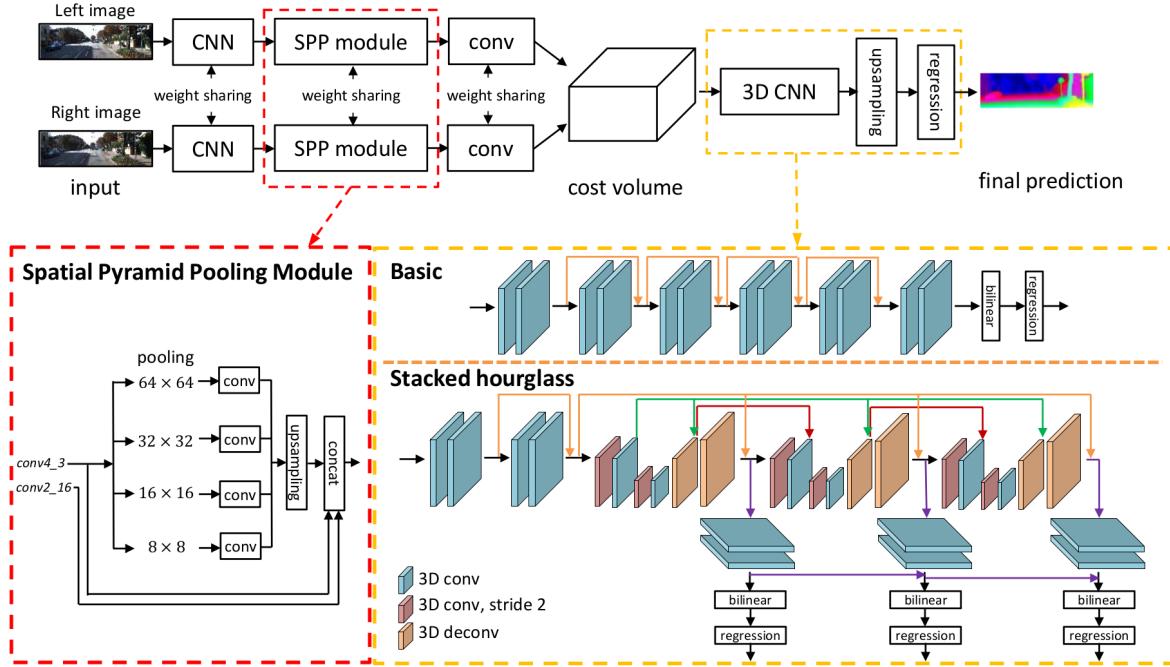


Figure 2.2: Generalized network architecture for Pyramid Stereo Matching network. The 3D CNN has two available configurations, Basic and Stacked hourglass. Stacked hourglass was used here. The Spatial Pyramid Pooling Module helps describe how features at varying magnification levels were concatenated together. Image reproduced from [2].

To properly understand PSMnet, the network will be explained in three general steps, as described by the original authors Jia-Ren Chang and Yong-Scheng Chen [2].

1. Initial 2D convolution into Spatial Pyramid Pooling
2. Aggregate all data into a cost volume
3. Perform 3D convolution and “deconvolution” on the cost volume until a final prediction is reached.

Step 1: Spatial Pyramid Pooling (SPP) Module

To begin finding a disparity map from an image pair, the images each fed into a 2D CNN with weight sharing (meaning that the two CNN’s have identical weights). The two images are thus changed from color images with dimensions (Height, Weight, Channel=3) to feature maps with dimensions (H/4, W/4, Features=32). These two maps which are then each fed into a SPP module, which uses average pooling to capture details of each map at various scales. In effect, the SPP module “zooms” at various magnifications in order to capture different levels of detail in the same image, all of which are then concatenated together, along with the original feature map, into an aggregated feature map. At this point, there feature dimension is rather large, expanded to 320 from 32. Additional convolution with a 1x1 filter size helps reduce the number of features back down to 32, and each aggregated feature map is sent into the cost volume to be further combined.

The primary reason for the SPP module is to take advantage of “image features rich with object context information” [2] while addressing the fixed-size constraint of CNN’s. By looking at the image at various magnifications, the network learns to “incorporate hierarchical context information”, such as an object like a car and its sub-objects like tires, windows, and so on. At this point, the network is not yet calculating anything like disparities, but rather learning the relationships between sub-regions that may later play a part in determining distance relationships between them.

Step 2: Cost Volume

Next, the two aggregate feature maps are combined into a single 4-dimensional matrix known as a cost volume. As described by Chang and Chen, the left and right feature maps are concatenated, resulting in a 4D “volume”, with dimensions (Height, Width, Disparity, Features). The disparity level is created from the various scales at which features are concatenated, or “levels” as the paper describes. Later on, these disparity values will be refined into a final estimate via regression. The usage of a cost volume will later necessitate the use of a 3D convolution, to deal with the extra dimension generated. At this point, the cost volume has a general shape of $(D/4, H/4, W/4, 64)$, as described by Table 1 in the original paper [2].

Step 3: 3D Convolutional Neural Network

The last major step of PSMnet is to take the cost volume and perform stereo matching. At this point, the task has been somewhat simplified by using an SPP module at the start in order to involve “different levels of features” [2]. There are two 3D CNN architectures that the authors developed, a “Basic” and “Stacked Hourglass” architecture. This thesis used the “Stacked Hourglass” network for its improved performance. In brief summary, the Basic 3D CNN is comprised solely of residual blocks that convolve the cost volume. This reduced cost volume is then upsampled up to a shape of (Height, Width, Disparity), and finally regression is used to “calculate the disparity map with size $H \times W$ ”.

The Stacked Hourglass architecture, in contrast to the Basic architecture, uses two kinds of convolutional layers along with complimentary transposed convolutional layers to repeatedly reduce and upsample the cost volume. Figure 2.3 below reproduces just the Stacked Hourglass network from Figure 2.2 above.

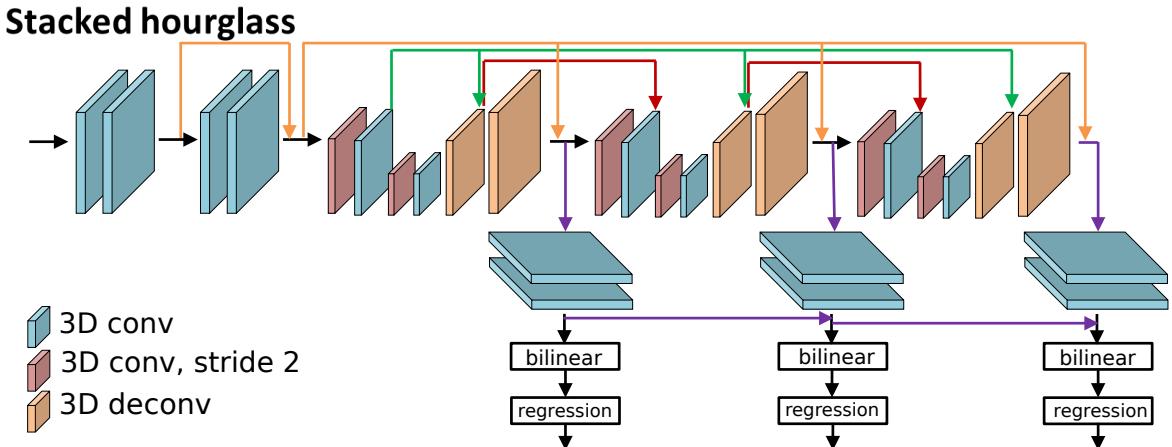


Figure 2.3: Detailed view of Stacked Hourglass architecture from PSMnet. Image reproduced from the PSMnet paper [2].

The first four layers (marked in blue) are 3D convolutional layers with a stride of 1. The disparity, height, and width dimensions don't change, but the number of features are reduced from 64 to 32. Next, the network actually begins its "hourglass" portion, with a 3D convolutional layer of stride 2 (marked red) halving the Disparity, Height, and Width dimensions (by half) while doubling the Features. This step is again repeated (without changing the Features size), bringing the dimensions to (H/16, W/16, D/16, Features=64).

This is the "smallest" that the cost volume will be, and it is next upsampled via 3D transposed convolution, or 3D "deconvolution" as it is also known. The simple reason that this network is upsampled using deconvolution is to reconstruction predictions about the disparity at a higher resolution. This is similar to how a low-resolution image may be upsampled into a higher resolution, requiring some kind of calculation to estimate the true value of unknown pixels. In the same way, deconvolution expands the cost volume again to correct the shrinking that happens with convolution. However, in order to recover some information about the values lost from the hi-res cost volume, a "skip layer" is also implemented. Skip layers, in a practical sense, are an operation to take the weights from a previous layer and add them element-wise to a future layer later on. This is best exemplified by the green arrows in Figure 2.3, although the orange and red arrows are also skip layers. Thus, features that may have been lost during convolution are partially restored during deconvolution.

Once the cost volume has been reduced and then upsampled in size, a single hourglass step has been taken. This hourglass shape is also known as an encoder-decoder architecture, popularized by SegNet, an architecture for image segmentation by Badrinarayanan, Kendall, and Cipolla [1]. In their paper, the authors argue that the key component is "the decoder network which consists of a hierarchy of decoders one corresponding to each encoder", enabling the ability to perform non-linear upsampling while still being memory efficient. The "stacked" nature of PSMnet comes from repeating this hourglass network multiple times, three in total.

Each time the network is upsampled, it outputs a new cost volume. Each of these cost volumes, eventually becoming disparity maps, are optimized during training. In testing, the final cost volume is used. This cost volume is then upsampled in a bilinear fashion, and finally the continuous disparity map is calculated via "disparity regression". The authors of PSMnet utilize this as described by Kendall et al. [11]: in contrast to estimating disparity with an argmin operation, Kendall defined a differentiable "soft argmin" function and capable of creating sub-pixel disparity estimates. Although the details of this operation will not be explained here, Chang implements this in PSMnet to collapse the four-dimensional cost volume into a two-dimensional disparity map.

2.2 3D Reconstruction from Stereo Data

Reconstruction is the process of taking a series of 2D images and producing a 3D representation of that information, as explained by Szeliski [25]. Here, the main goal is to end up with a point cloud, an unordered set of metric coordinates. Epipolar equations, equations derived from the geometrical relationship two cameras viewing a single scene, allow one to perform these conversions, which are explained below.

The overall steps of reconstructing the resulting stereo image consists of three general equations adapted for use with matrix operations[25, 26]. This also includes selecting the proper constants for each calculation. To achieve all this, a specific Python class was created that would initialize by loading the fully trained model and use it in a non-learning “testing” mode, whereby it would take in either an image path pair or a loaded pair of image arrays. The pair would then be evaluated in PSMnet, and the resulting disparity map would be converted to a point cloud using the epipolar equations in matrix form. Each record of the KITTI dataset has its own calibration file, but the baseline value remains the same.

To begin describing how the transformation is done, a simple problem is presented with the following assumptions. Let there be two cameras in a scene, both facing the same direction and able to see an object at point P , such as a ball. The following assumptions are made:

- The two cameras are identical in performance, producing equal quality images at the same point in time.
- The camera centers are aligned (for simplified rectification).
- Images produced are already rectified (to remove all distortion). Rectification involves using the camera’s intrinsic properties in order to correct lens distortion, specifically barrel distortion.
- The camera centers O_i are separated by a baseline distance b .
- The two cameras have identical horizontal focal lengths f_U .
- The ball appears at some horizontal distance u_i , whether the origin is measured from the left side of the image or the center.

When the two cameras take an image of the ball, the light rays they capture may be drawn as given in Figure 2.4. The figure gives a top-down view of what both cameras capture, and the horizontal pixel location at which the light ray comes into contact with each camera center.

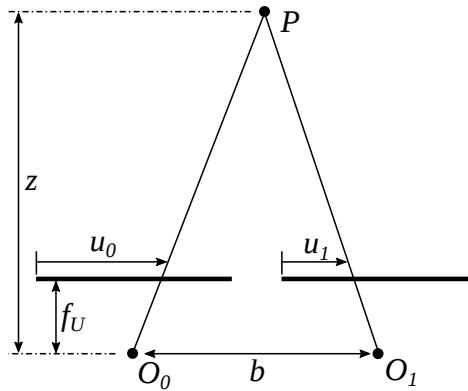


Figure 2.4: Simplified graphic demonstrating how disparity may be derived from two cameras. The images the cameras produce must first be rectified.

Given the above physical setup, the equation describing the relationship between disparity and depth z may be found. When a matrix of disparity points is given, denoted as D , the resulting equation is then:

$$z = \frac{f_U b}{D} \quad (2.1)$$

Thus, a depth map may be generated by using this equation on each disparity value. The process of determining horizontal distance x and vertical distance y requires further exploitation of the camera geometry, and may be calculated then as:

$$x = \frac{z(u - c_U)}{f_U} \quad (2.2)$$

$$y = \frac{z(v - c_V)}{f_V} \quad (2.3)$$

It is important to note that f_U and f_V are for horizontal and vertical focal lengths (respectively), (c_U, c_V) are the pixel center of the image, and (u, v) represent a (column, row) coordinate location in D with the origin at the image's top-left corner.

In practice, the calculation is performed in a series of matrix calculations rather than as an element-by-element operation. These steps are covered below with accompanying Python code. This code assumes the usage of the Numpy module and that the disparity map has already been generated and saved.

The first step in the reconstruction process is determining the baseline, focal lengths, and image center. The baseline is a given constant from the KITTI development kit, listed as a fixed value of 0.54 meters. The accuracy of this measurement is not known beyond that. For each scene, the focal lengths are determined directly from the left color camera calibration matrix P_2 , where $f_U = P_2[0, 0]$ (row 0, col 0) and $f_V = P_2[1, 1]$. The image centers are found by dividing the disparity dimensions by half of the image dimensions, which corrects for non-standardized image dimensions present throughout the dataset. The depth matrix Z is calculated via matrix operations in nearly the same way as given in Equation 2.1. All code used below and throughout this work can be found in the thesis Gitlab repository [7].

```
import numpy as np
# STEP 1: generate / load a disparity map
disp=np.load(dpath)

# STEP 2: obtain parameters
P2 = rec.P2 # pre-existing object that parses calibration file
fHoriz = P2[0,0]
fVert= P2[1,1]
Baseline = 0.54 # meters. Obtained from devkit
imH,imW = np.array(pil.open(lpath).convert('RGB')).shape
Cu=disp.shape[1]-imW/2 # Cu = imW/2 - (imW - dispW)
Cv=disp.shape[0]-imH/2 # Cv = imH/2 - (imH - dispH)

# STEP 3: Calculate depth from disparity values
zvals = fHoriz*baseline/disp
```

Next, the X and Y matrices are generated by creating a vectorized forms of their corresponding theoretical equations:

```
# STEP 4: Convert x_px & y_px to metric
xpos=np.arange(0,zvals.shape[1])-Cu
```

```

ypos=np.arange(0,zvals.shape[0])-Cv
ulocs=np.tile(xpos*-1,(zvals.shape[0],1))
vlocs=np.tile(ypos*-1,(zvals.shape[1],1)).T
xvals = ulocs*zvals/fHoriz
yvals = vlocs*zvals/fVert

```

Once each matrix has been created, they may be combined to form a set of three dimensional coordinates, as shown below in Figure 2.5(e).

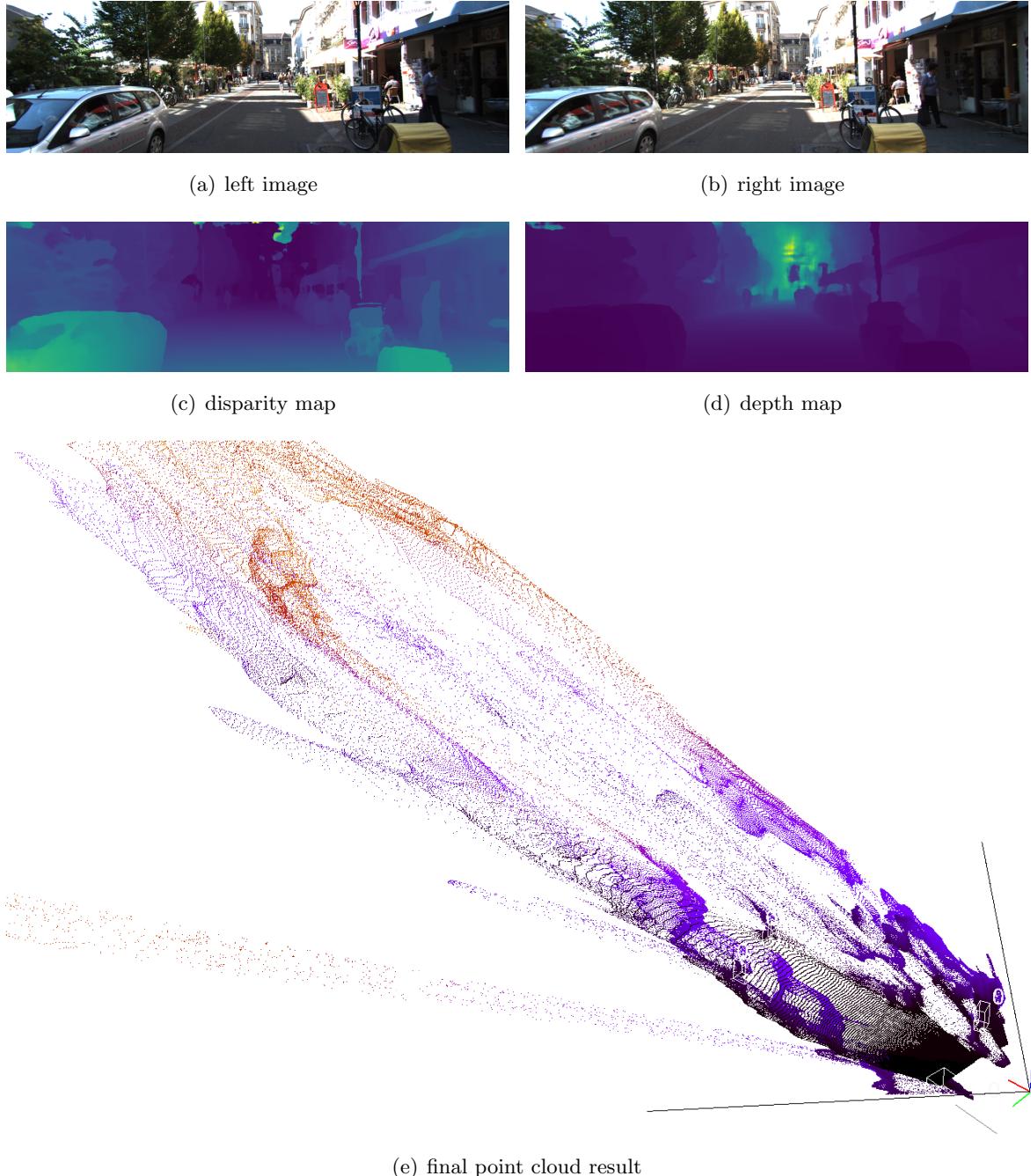


Figure 2.5: Various states of transformation from an image pair to a point cloud. Two images, (a) and (b) are given to PSMnet, which outputs a disparity map (c), which is changed to a depth map (d) via 2.1, and is finally transformed into a point cloud (e) with the remaining two epipolar equations. Subfigure (e) shows the camera horizontal field of view with two black lines.

3 Frustum Pointnet and Point Clouds

Frustum Pointnet (FPnet) is one of the top-performing 3D object estimation networks on the KITTI Object Detection leaderboard. Its power comes from the ability to work with an incomplete or even reduced set of point data while still taking advantage of mature 2D object detection techniques. The network will be described in detail here, with the general architecture of FPnet is shown below in Figure 3.1, reproduced from the original paper by Qi et al [20].

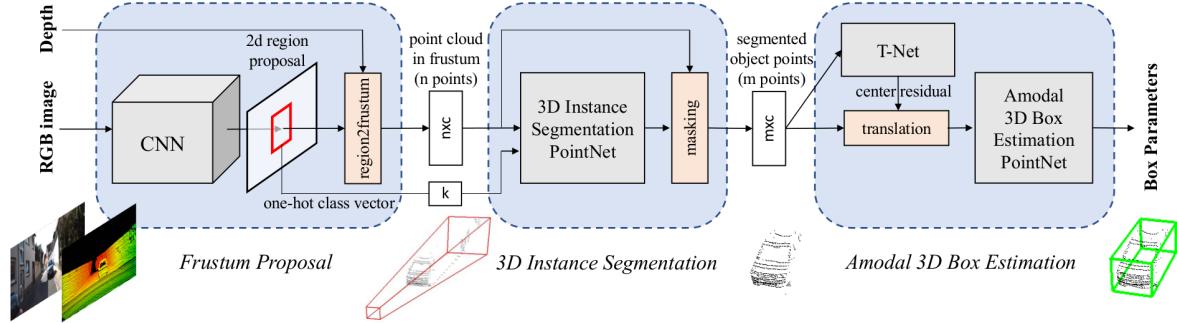


Figure 3.1: Frustum Pointnet architecture. Reproduced from original paper [20].

3.1 Frustum Pointnet Architecture

The network itself works in three separate but closely related steps:

1. Frustum Proposal: A 2D CNN takes an RGB image and localizes a set of detections as region proposals, also generating a “one-hot” class vector that determines which classes are searched for in the next step.
2. 3D Instance Segmentation: For each region proposal, a cone or “frustum” is cut out from the point cloud, and a 3D instance segmentation is performed, and a masking step removes both needless foreground and background points.
3. Amodal 3D Box Estimation: Each filtered subset of the point cloud is then subjected to amodal 3D box estimation, meaning that the incomplete set of points are then used to estimate a final set of 3D bounds defining the estimate for each detection.

Step 1: Frustum Proposal

The very first step of FPnet requires a subnetwork, a 2D object detector, in order to begin generating proposals. The exact nature of this detector can be swapped for any 2D detector, such as R-CNN, SSD, etc. but the core step is to obtain 2D pixel-wise bounding boxes of target classes. Per the authors, an Feature Pyramid Network model is used to determine these bounding boxes, developed by Lin, Dollár, Girshick, He, Hariharan, and Belongie [14]. As described by Qi et al., they “pre-train the model weights on ImageNet classification and COCO object detection datasets and further fine-tune it on a KITTI 2D object detection dataset to classify and predict amodal 2D boxes.” [20].

Once these bounding boxes have been created, called “2D region proposals”, they are reconstructed into cone-like volumes called frustums via a custom-made function. In combi-

nation with the raw depth map, sub-cloud is created for each region proposal along with a “one-hot class vector”, essentially passing on the 2D information about each frustum.

This step is straightforward enough, and the reasoning is obvious: 2D object detection has matured much faster than that of 3D, to the point that some 2D detectors can run in real-time with an average precision over 90%. This reduces the search space from an entire lidar scan down to a narrow cone of points. As stated by Qi et al, “the computational complexity of 3D search typically grows cubically with respect to the resolution and becomes too expensive for large scenes or real-time applications” [20].

Step 2: 3D Instance Segmentation

Once a set of sub-level point clouds and their respective class vectors have been generated, this sub-cloud must be further filtered by removing “clutter”. The core task here is to remove foreground and background points, which is implemented via a “PointNet-based” network and whose sub-architecture is shown in Figure 3.3(a). Pointnet, as developed by Qi, Su, Mo and Guibas [19], is capable of taking a point cloud and creating a “probability score for each point that indicates how likely the point belongs to the object of interest” [20]. Pointnet is itself a special type of spatial transformer network (STN), based on the work by Jaderberg, Simonyan, Zisserman, and Kavukcuoglu [10]. STN’s are designed to transform data (scale, rotate, flip, etc), and learn how to do so as part of a larger network architecture. This enables some unique and powerful classification capability, where the network learns to first reshape the data before attempting to classify it.

There is an added benefit to Pointnet as well: each sub-cloud has a prior category that is believed to be the primary subject of that sub-cloud, such as a car or pedestrian. The network can then learn to look for “car-like” points, based on the prior training. Pointnet itself is a robust network model that can handle point corruption, extra noise points, and even reduced points when segmenting objects. As shown below in Figure 3.2, the network learns to remove occlusion (foreground noise) and clutter (background noise). The result from this step is a smaller sub-cloud of segmented object points along with all previous class information.

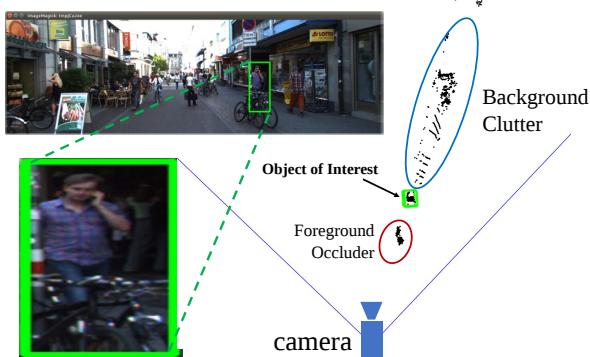


Figure 3.2: Frustum Pointnet 3D segmentation example. Reproduced from original paper [20].

This is an important step in the detection pipeline because the network must be able to remove noise since the next step, amodal estimation, depends on having as little extraneous information as possible. This step is a 3D analog of the 2D segmentation task, where each

pixel is either included or excluded as a part of the target class. Additionally, the origin of these points is transformed from the sensor coordinate to a local “3D mask” coordinate (centered at the sub-cloud centroid and oriented in the same direction as the frustum), while also learning to recognize the general geometry of an object of a certain category [20].

Step 3: Amodal 3D Box Estimation

The final step in FPnet is to perform amodal 3D box estimation, meaning the estimation is done with incomplete or missing data. This step is performed in two parts: a “light-weight” version of Pointnet named “T-Net” is used solely to estimate the actual center of the object from the given sub-cloud, and then this transformed set of points is run through another Pointnet to estimate the final 3D bounds. Both the center-estimation step and the bounding box step are supervised separately. The T-Net and the amodal estimation Pointnet are shown below in Figure 3.3(b) and (c), respectively.

T-Net is a lightweight version of Pointnets, and has the sole task of estimating the “true center” [20] of the object, which is then set as the origin. Object priors are not used here; instead, supervised learning teaches the network to estimate the general size of the object, given only the sub-cloud set of points. This better prepares the next Pointnet to then predict the all parameters for bounding box estimation: object center (c_x, c_y, c_z), size (h, w, l) and heading angle θ [20]. The bounding box center is later refined by combining the various residual centers from previous transforms.

The need to change the residual center stems from empirical proof that the network learns how to estimate the object orientation much better when this step is enabled. The original paper demonstrates (in Table 8 of the source) that there is an accuracy improvement from 12.5% to 74.3% by including the various sub-nets to rotate and recenter the origin, strongly implying that the network needs to work with a local origin rather than a global one to better learn its parameters.

3.2 Modification of Frustum Point Net

Frustum Pointnet, being originally written for Python 2, needed to undergo some minor adaptations for usage with Python 3. Additionally, several modules and functions were deprecated or needed modification, and so had to be properly adapted. There are also two actively maintained versions of the Frustum Pointnet code. as explained in the original paper [20]: the first which uses standard Tensorflow libraries and is based off Pointnets [19], and the second which uses custom Tensorflow operators that must be compiled, based off Pointnets++ [21]. For the purposes of this paper, version 1 was used.

In order to transition from using the provided pre-trained model and preprocessed data to self-made preprocessed and trained data, a set of steps and “paths” were created to guide what was happening in each step and set of “runs”. As shown below in Figure 3.4, there are various steps with a gradual transition to stereo-based point cloud data.

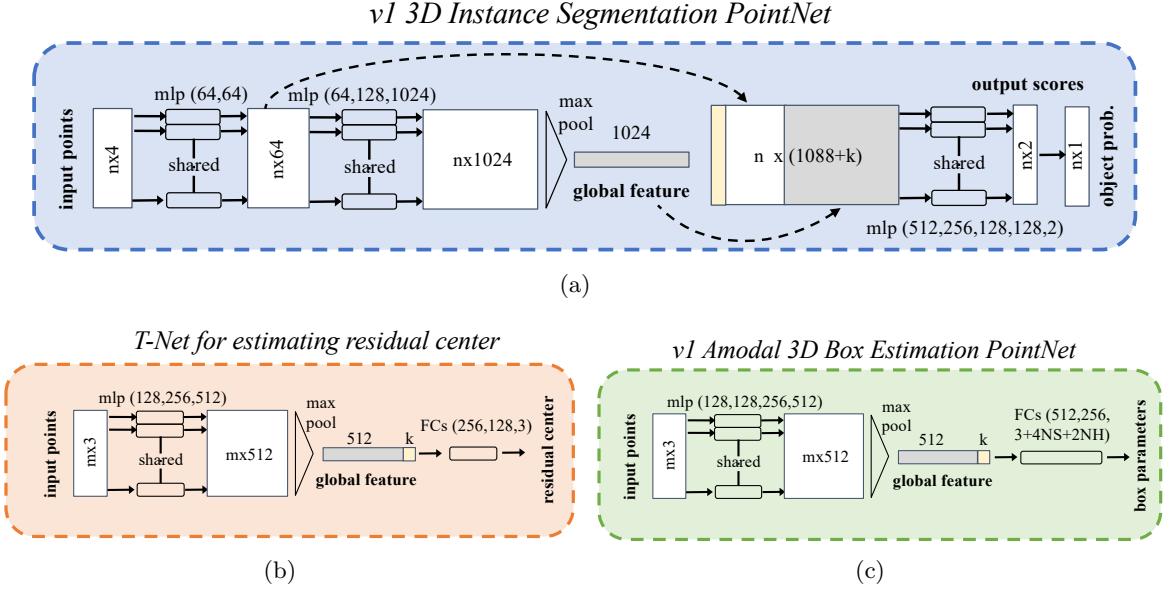


Figure 3.3: Frustum Pointnet sub-networks. Reproduced from original paper [20] Subfigure (a) is the architecture used during instance segmentation: removing foreground and background noise. Subfigure (b) shows T-Net, used for estimating the true object center. Subfigure (c) represents the final Pointnet, used to estimate the object’s local bounding box values.

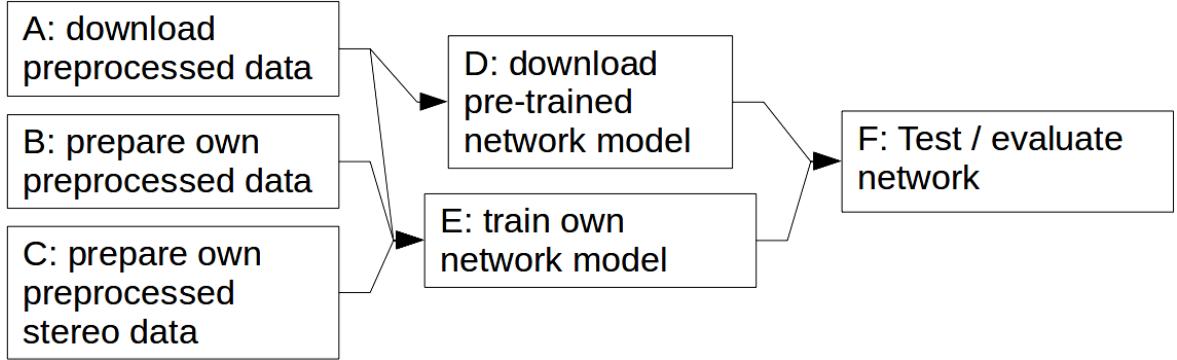


Figure 3.4: Various paths that were planned to simply transition from lidar-based 3D estimation to stereo-based 3D estimation. Generally, paths went chronologically as follow: ADF, AEF, BEF, and finally CEF.

The most basic path, ADF, is simply using the given model to evaluate and achieve similar results to the official paper. Next, AEF used lidar pointclouds, but the model was trained locally. BEF required locally running the “preprocessing” step, but was still functionally similar to the previous two paths. Finally, CEF truly deviates from the other paths by using a stereo-generated point cloud to preprocess data, then train on that, and finally evaluate the model’s detection capability.

4 Experiments

Once the network was built and implemented, several experiments were conducted and iterated upon in order to achieve the best possible performance with the given datasets and knowledge. The first subsection is dedicated to explaining the performance metrics of 3D object detection, then moves on to describing the results of each experiment and investigation performed with the network. As noted below and in the results of each experiment, the class “Car” is the sole class that is examined. Other classes, such as Cyclist and Pedestrian, have evaluation code and even good ground truth labeling for them, but not in the sheer volume that the Car class has. Additionally, cars have the inherent benefit of having a generally uniform shape (box-shape body, wheels, etc). See Appendix B for more information. All code used in both producing the results as well as the graphics of this paper are provided by [7].

4.1 Performance Metrics

In order to determine the quality of results obtained, a few different metrics are implemented as are standard in practice. Here, the various metrics are explained, and in Appendix A examples are provided. The concept of verifying accuracy in a statistical estimate is covered in [15], and some of this information is covered briefly below. In the field of image recognition, one particularly interesting metric is “Intersection over Union”, also referred to as IOU or Jaccard Index. From this metric, precision and recall may then be calculated. Precision and recall are fundamental in obtaining more abstracted metrics such as average precision (AP) and mean average precision (mAP), which are how different networks are compared. The following information is primarily taken from the well-known PASCAL VOC (visual objects classes) Challenge [5] and “An Introduction to Information Retrieval” by [15].

4.1.1 Defining an Outcome

The first step in measuring performance is categorizing what a result may be. With a visual task such as object classification, there are 4 possible outcomes (all references to true/false positive/negatives in this document will be referred to as “outcomes”), which will be as shown below. In general, there may be a True Positive, False Positive, True Negative, or False Negative outcome. In practice, True Negatives are not used, and the remaining three are used to varying degrees. Each is better clarified as such:

- True Positive: Correctly detecting a true object
- False Positive: Incorrectly detecting something that isn’t there
- False Negative: Incorrectly ignoring a true object
- True Negative: Correctly ignoring something that isn’t there

	True...	False...
Positive		
Negative	...	

Figure 4.1: A visual representation of various outcomes, ground truths as solid black boxes and detections as dotted red boxes.

To actually classify a result in one of these categories, computer vision depends on using the overlap of a given bounding box estimate with a bounding box ground truth. If the overlap is above some threshold, the estimate is said to be a true positive. In some metrics, two estimates overlapping the same ground truth will only count the larger overlap, or the more confident estimate. This overlap is formally known as an Intersection over Union (IOU) value, or a Jaccard index.

4.1.2 Intersection Over Union

In image recognition, as well as other spatially-based tasks, accuracy is needed in various forms to know “how well” a prediction overlaps, or matches, the ground truth. If, for example, an object-detection algorithm predicts the location of a car in a photo, one would like to know if such an estimate has any value, ideally with as few parameters as possible.

In light of this, IOU encompasses all relevant aspects of rating the overlap of two geometric shapes (e.g. rectangles) while enabling an intuitive, non-binary scoring of an estimate. IOU is calculated as two bounding regions’ intersection over their union, just as the name states. Visually, this looks something like the below in Figure 4.2. Uniquely, the calculation of area and intersection specifically for image boundaries is inclusive of the bounds, meaning that the length of a given difference must have “+1” added to it.

In order to formally calculate IOU, a generalized form may be generated to apply to n-dimensions. The generalized mathematical equation is simply as follows. Given a region A and a region B:

$$IOU = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad (4.1)$$

4.1.3 Precision & Recall

Once all detections have been tallied and placed into their correct categories, their precision and recall may be calculated. In information retrieval, precision is defined as “the fraction of retrieved documents that are relevant”, and recall is defined as “the fraction of relevant documents that are retrieved”. Reworded for object detection: precision is the number of correct predictions divided by the total number of predictions, and recall is the number of correct predictions divided by the total number of possible answers. These two are also given as equations in terms of true positives and so on.



Figure 4.2: Example of ground truth bounding box (solid green) and prediction bounding box (dashed red). In this image, the overlap between the green and red regions is the intersection, while the combined area is the union. The IOU of the two boxes is 0.64. Index 8 of the KITTI Object Detection dataset.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \quad (4.2)$$

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \quad (4.3)$$

Theoretically, precision and recall are generated at multiple points by varying a probability threshold and graphing each result. In practice, a precision-recall curve is generated by a more complex process that involves taking the cumulative sum of the detections in order of descending confidence level (1.0, 0.99, ...). Once a precision vector and a recall vector have been created, the plot may appear something like below, reproduced from Figure A.5 in Appendix A.

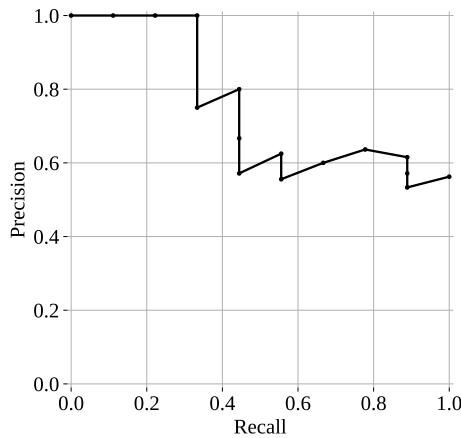


Figure 4.3: A typical precision-recall curve. Taken from the example in Appendix A.

4.1.4 Calculating Average Precision

Once a set of recall and precision data has been calculated, then AP is a fast and simple calculation. Formally, average precision is defined as the area beneath a precision-recall curve. However, this metric has been calculated in various forms and often modified by the scientific community, including the previously common 11-point average precision used

by the Visual Objects Challenge (VOC), used until 2009 and by several subsequent object recognition challenges. 11-point AP has been argued to be slightly more robust against very jittery curves, but also judges results rather “crudely” and less accurately [4]. For the purposes of computer vision, AP may also be considered a weighted sum of precision, considering that the differences in recall value may be used as the weights since their sum is 1. Thus, to match the function output by [18], one may simply take a rectangular numerical integration of the values in Table A.2. It must be kept in mind that the order of the values MUST be in order of ascending recall value. The mathematical formula for calculating AP is then given in 4.4, where 1 is the starting index and n is the length of the vector:

$$AP = \sum_{i=1}^{n-1} (re_{i+1} - re_i)(pr_{i+1}) \quad (4.4)$$

4.2 Experiment 1: Lidar-Based Performance as a Baseline

To first properly understand the performance capability of lidar-based 3D object detection, the typical pipeline was first verified locally. There are multiple setup steps as given by the authors, with a specific training-validation split. This was followed, and the performance was indeed on par with what exists officially on the KITTI 3D Object Detection benchmark. As described in Figure 3.4, there are three paths that emulate the methodology official results, all of which use lidar as the source of point cloud information:

- ADF (downloading the pretrained model and evaluating immediately)
- AEF (download preprocessed data, train network, and evaluate)
- BEF (prepare preprocessed data, train network, and evaluate)

Aside from path ADF, which of course features no training component, the training loss of each path is visualized below in Figure 4.4. path “AEF” was performed twice (revision 0 and revision 1), during different times in the project; The performance remained relatively consistent, however.

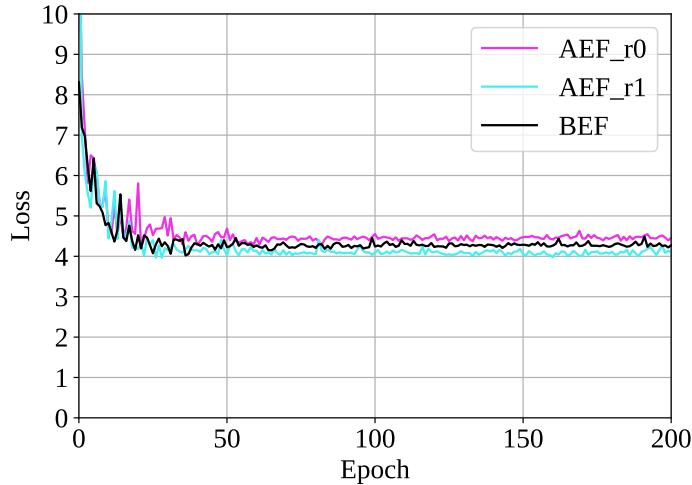


Figure 4.4: Loss over epochs graph for Experiment 1. Run AEF.r1 achieved the lowest training loss, settling to a value of around 4.1. The loss value represents an average score of how much a network predicts incorrectly during training.

Taking a slightly closer look at the training data, the lowest value loss that the network achieved while training was 4.1. The rate at which loss was changing also seemed to decrease to zero, indicating that the network was quite stable in its weights by the end of the training. For future reference, path BEF will be used to compare to other networks since it performed the best of these run paths.

Additionally, the precision-recall curve of each path for the “Car” class is also shown in Figure 4.5. The curves generally indicate that the lidar-based point clouds were quite robust, and the AP score for the three paths are as follow.

Table 4.1: FPNet lidar-based results, across varying difficulties. BEF performed marginally better than the other runs, and was thus used as a reference in future experiments. AP score is calculated by finding the area under the precision-recall curve.

Path	AP_Easy	AP_Med	AP_Hard
ADF	84.07	71.28	63.35
AEF0	82.59	68.72	61.25
AEF1	83.51	68.99	61.04
BEF	84.12	71.00	63.27

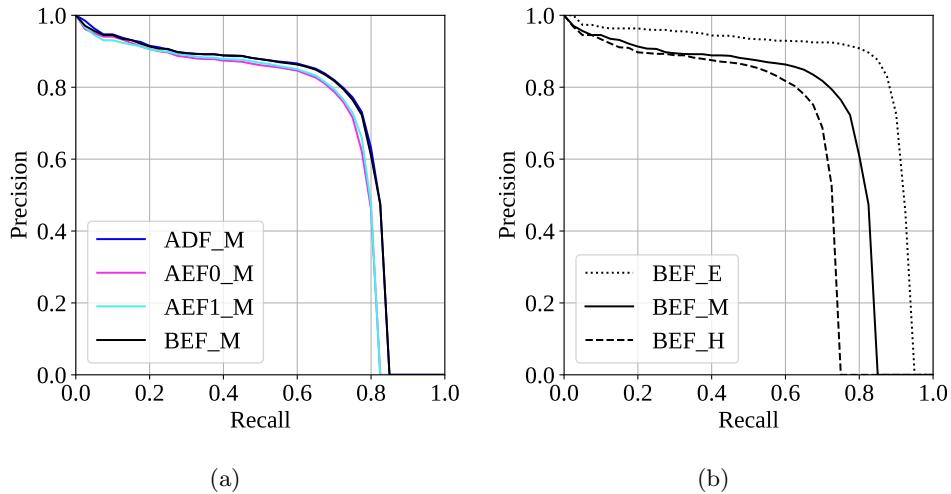


Figure 4.5: Precision recall curves for Experiment 1. Subfigure (a) shows various runs at compared medium difficulty, while Subfigure (b) shows the best-performing run BEF at Easy, Moderate, and Hard difficulties.

Given that these are results from a lidar-based network, they are therefore the benchmark against which performance was then measured.

4.3 Experiment 2: Stereo-Based Performance, using Stereo Dataset

The first experiment had the objective of using the stereo network, once trained, to provide a new source of point cloud data. This experiment corresponds with the path “CEF”: preparing preprocessed data with stereo vision instead of lidar, train the network model on this information, then evaluate the network performance. Because this experiment was later further modified, it is known as CEF_r0, while the second version is named CEF_r1.

The training loss is provided below in Figure 4.6, with a comparison to previous the previous, lidar-based path “BEF”.

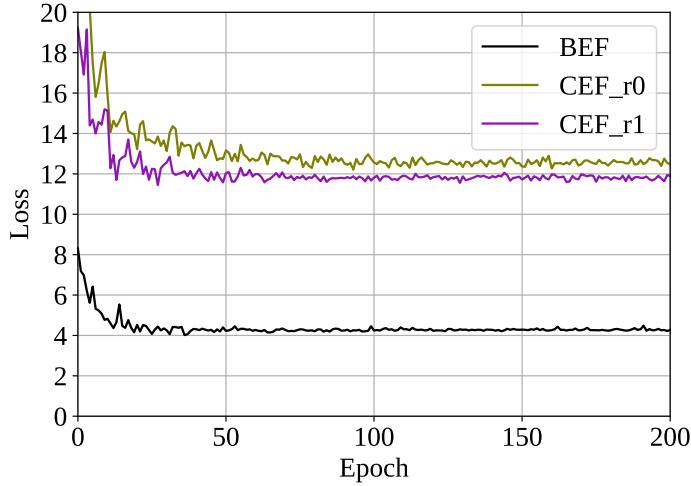


Figure 4.6: Loss over epochs graph for Experiment 2. Run CEF_r1 achieved the lowest final loss value, although reference curve BEF shows that lidar-based methods typically achieve lower loss during training.

As can be seen from the figure, the training loss also reached a point of equilibrium and ceased decreasing after some time, but at a much higher value than path BEF. CEF_r0’s loss value instead settled closer to a value of 12.4.

In the evaluation, CEF_r0 performed at a significantly lower level than BEF, indicating a need to further refine the model. However, there is some promise in this result, as it indicates that the system does perform well in the “Easy” mode, when cars are generally within 30 meters of the sensor. The results from this experiment, as well as further reading, motivated the next experiment conducted. This is shown below, in Figure 4.7.

After the initial results CEF_r0, the 3D reconstruction step was reexamined for any possible errors. It was then discovered that PSMnet internally crops stereo image pairs into a standardized size, a significant piece of previously unmentioned information. The primary cause of non-standardized image sizes in the object detection dataset is due to the rectification performed on each raw image. This means that the images, after being rectified, are then cropped in a non-standardized way to remove the resulting non-image regions. PSMnet itself only works on a standardized image size, so it therefore takes the smallest dimensions across the dataset and crops all image pairs to that size, [1232,368] pixels. The PSMnet crop is anchored at the bottom-right corner of each image, which provides some consistency but may still contribute to a source of error, since the exact method of cropping both rectified data and images for stereo processing are not totally aligned. A second and lesser source of possible error that was corrected in CEF_r1 was the method of reconstruction used. This updated method is reflected in 2.2. The constant being used for all disparity-to-depth transformations was consistent, rather than being the product of baseline and F_U , horizontal focus length. Thus, Figure 4.7 and Table 4.2 show the improvement from CEF_r0 to CEF_r1.

Table 4.2: Experiment 2 results, with Frustum Pointnets using stereo data. Average precision shown for the first two attempts, across varying difficulties, for the Car class.

Path	AP_Easy	AP_Med	AP_Hard
CEF0	35.61	23.02	20.43
CEF1	42.57	28.60	24.07

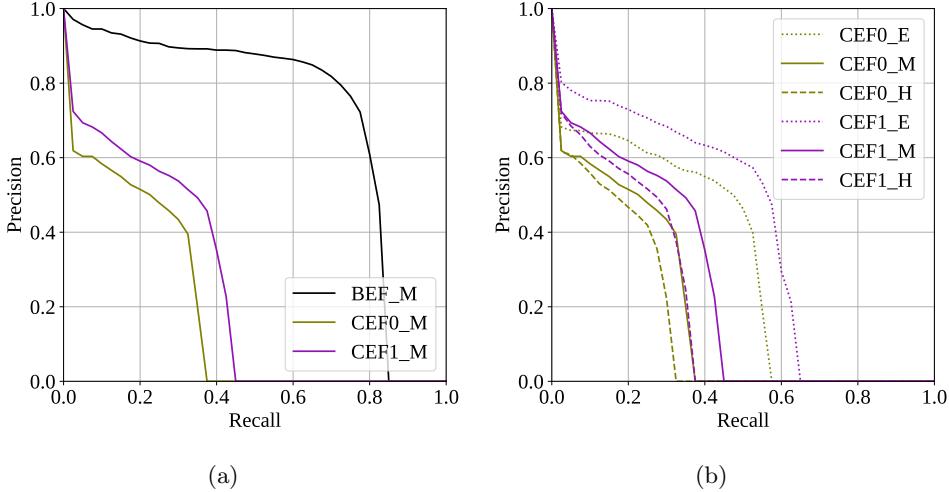


Figure 4.7: Precision-recall graphs for both CEF_r0 and CEF_r1, with lidar-based result BEF also shown for reference. In (a), the improvement between r0 and r1 manifests in an AP_M increase of 5 percentage points, and at “Easy” difficulty, the best curve in (b) has an AP of 42.57.

4.4 Experiment 3: Stereo-Based Performance, using Object Detection Dataset

Upon reviewing the most closely related paper, by [26], the suggestion of changing the training dataset for PSMnet was taken. A pretrained stereo model was once again taken, directly from the authors, and finetuned exclusively on the same training split used by FPnet. This is in contrast to the original KITTI stereo dataset, used by the authors. The reasons for this are given below as well as further clarified in Appendix B.

4.4.1 Similarity of Ground Truths Across Datasets

After the initial evaluation, the network was prepared and trained on KITTI data. The original authors of PSMnet provide a model pre-trained on the Freiburg SceneFlow dataset; this model was also used. After this step, however, it must be clarified that there are images in the KITTI stereo dataset that overlap greatly in similarity to those in the KITTI object detection dataset. The difference between the two datasets are essentially that one has been optimized for object detection, and the other for stereo disparity map estimation. This similarity was also noted in [26], and this paper took the same approach to make corrections, as described below. Figure 4.8 below also demonstrates the similarity between two sample images, which points to the risk of having a part of the network trained on what may be validation or even evaluation data. This would result in data leakage and thus put the validity of the results into question.



(a) 000132_10 from Stereo



(b) 000286 from ObjDet

Figure 4.8: Example of how two unique records from the separate datasets have overly-similar scenes. There are multiple examples of this, which creates the motivation to generate a secondary set of disparity ground truths for the object detection dataset

In order to deal with the dataset overlap, the KITTI object detection dataset was adapted to be used for training stereo data. Lidar data was projected onto the left image plane, cropped to the original image bounds, converted to integer values, then multiplied by a constant. These steps are in line with both the way that [26] were able to generate their self-made stereo ground truths as well as how the KITTI dataset authors generated their ground truth, as described in the stereo development kit. Locations in the image that have no lidar data are left alone at a value of 0.0, and finally the resulting array is saved to a PNG image file format containing integer values. Please refer to the generation script in the repository for the exact steps used to generate the pseudo ground truth stereo images [7].

4.4.2 Results of Modification

It was discovered at some point that the training / validation data used in the KITTI stereo image set was overlapping too much with images used in the KITTI object detection task. Thus, [26] was referred to and guided the change in retraining PSMnet. The difference in training is shown below.

Figure 4.9 below shows the loss throughout the training of PSMnet on the object detection dataset, also known as PSMstar (based on the similar name given by Wang et al [26], as well as the original stereo dataset model for reference).

In addition to the quantitative difference in training, a qualitative difference in the esti-

4 Experiments

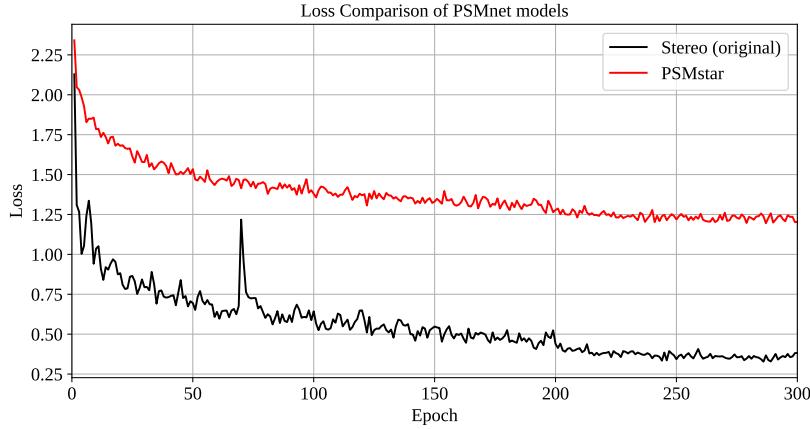


Figure 4.9: Loss graphs for both the original stereo-dataset PSMnet model and the retrained, object-detection-based “PSMstar” model. The average difference between the two curves is a loss of about 0.82. The higher loss for PSMstar may result from using self-made ground truth labels rather than using the more detailed stereo dataset versions.

mation pattern may be seen below, in Figure 4.10. Overall, the change in quality may be seen as a more detailed but less contrasting depth map.



(a) Stereo estimation with original PSMnet model



(b) Stereo estimation with new PSMnet model

Figure 4.10: Comparison of old and new model for PSMnet. Index 15 of the KITTI Object Detection dataset. There are typically less extreme values, yet also some higher contrast in the low (distant) values, as visible by a more clear outline of two pedestrians in the center of the image. See Figure 2.5(a) for photo reference.

The training loss of the modified FPnet is provided below in Figure 4.11, once again with run BEF plotted for reference. Runs CEF_r2 and CEF_r3 are nearly identical, although a

slight modification was tested out with CEF_r3: removing all points in the stereo-generated point cloud that were 1 meter above the sensor vertical height. However, the decrease of information, whatever its quality, seems to have had a very slight negative impact rather than positive one on the overall result.

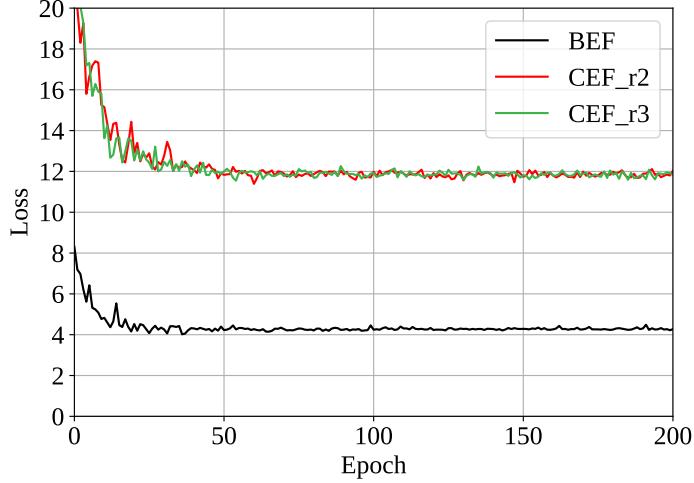


Figure 4.11: Loss over epochs graph for Experiment 3. Run CEF_r2 and r3 both ended training with a final loss of about 12.0. However, this value is still much higher than the reference BEF training loss.

The precision-recall curves for the “Car” class is given below in Figure 4.12, including one only for CEF_r3, as well as the AP scores for the various difficulties in Table 4.3.

Table 4.3: Experiment 3 results, with FPNet stereo-based results. Runs r2 and r3 shown across varying difficulties.

Path	AP_Easy	AP_Med	AP_Hard
CEF2	47.41	30.69	25.15
CEF3	45.43	30.45	24.29

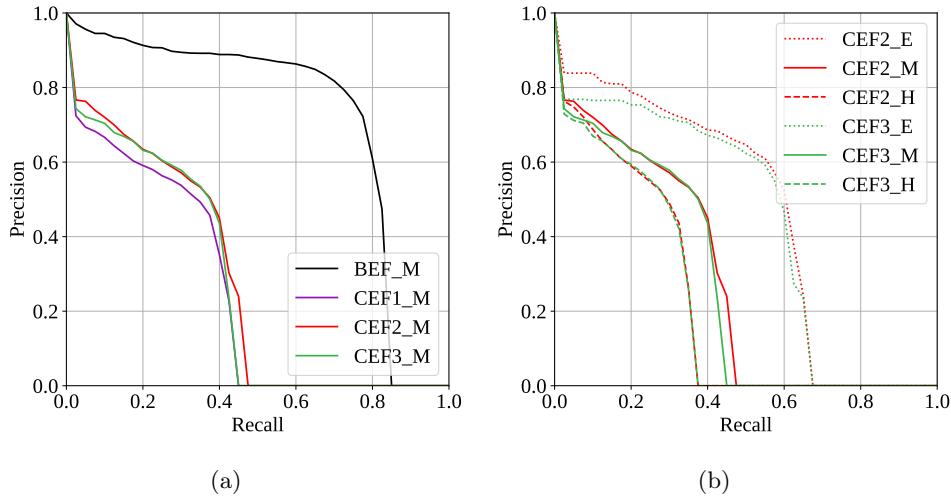


Figure 4.12: Precision-recall graphs for both CEF_r2 and CEF_r3 in (a), with lidar-based result BEF also shown for reference. Subfigure (b) further shows both across the Easy, Moderate, and Hard difficulties.

Although the results still lag behind lidar-based performance, Table 4.3 shows yet another improvement in AP from CEF_r1 to CEF_r2, improving by about 5 percentage points on “Easy”, 2 points on “Medium”, and 1 point on “Hard”.

4.5 Comparison with Other Networks

In addition to both developing and testing “SPCLnet”, the official performance data of other networks was downloaded, graphed, and compared. In addition to the self-made network, the following networks were compared:

- Pseudo-Point Cloud network, by [26]
- STD: Sparse-to-Dense 3D Object Detector for Point Cloud, [29]
- RCNN Stereo Network, by [12]
- RT3D: Real-time 3D vehicle detection, by [30]

Table 4.4: Results achieved by other notable networks on the KITTI dataset.

Path	AP_Easy	AP_Med	AP_Hard
STD	86.61	77.63	76.06
WANG	55.40	37.17	31.37
RCNN	49.23	34.05	28.39
RT3D	28.50	24.10	20.32

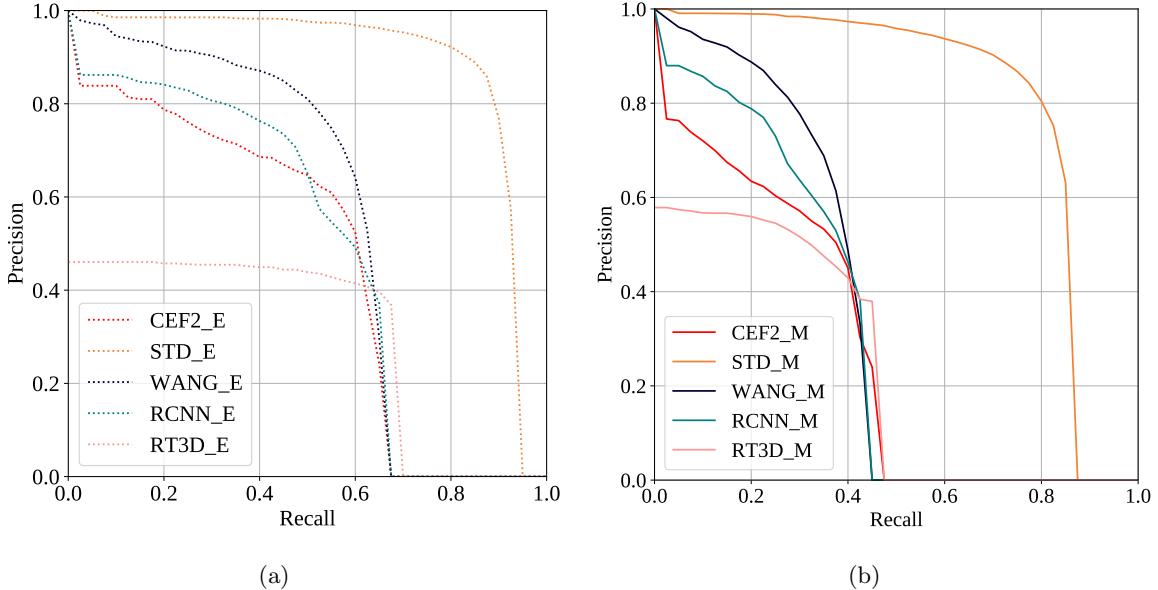


Figure 4.13: Precision-recall graphs. Subfigure (a) compares run CEF_r2 to other notable networks at Easy difficulty. Subfigure (b) compares the same set at Moderate difficulty.

The best performing iteration of SPCLnet, CEF_r2, while not performing the outright best, shares some interesting characteristics with the other stereo networks. For one, there is a sharp decline at the top-left of the curve, a trait that the RCNN curve also shares. This

is typically caused when the threshold to determine outcomes is higher, occurring at higher precision and lower recall values. In addition to this, all stereo algorithms reach 0.0 precision at approximately the same recall value, at each given difficulty (0.5 recall at Moderate, 0.7 at Easy). This would indicate that even when the threshold is nearly nonexistent, there simply aren't enough valid detections to outweigh the number of False Negatives.

5 Conclusions and Future Work

The main effort of this thesis has gone towards asking one general question: can stereo vision reasonably provide an alternative to lidar sensors in the realm of 3D detection? There are two specific conclusions that have been drawn from the literature as well as this work, specifically: Stereo vision can potentially compete with lidar as a viable distance sensor at relatively short distances, and the quality of the disparity data has a strong impact on the final results (and is a source of possible future work). Additional future work lies in real-time applications, which are within reach.

Stereo vision cameras may always trail a lidar sensors in terms of quality, but there is a cost-quality tradeoff that becomes appealing under certain circumstances. As can be seen by the results in Table 4.3 and Table 4.4, stereo-based networks suffer a larger decrease in performance when compared between the “Easy” and “Moderate” difficulties (corresponding to objects located within 30 meters and objects located beyond 30 meters, respectively). Figure 1.2 also demonstrates that as lidar reference distance increases, the fraction of stereo points correctly within error bounds decreases. There are large strides being made in the field, from pure stereo approaches [30, 12] to macro-networks that draw inspiration from lidar based approaches [26]. Throughout the duration of this paper’s writing, there have already been a few new submissions to the KITTI dataset, indicating a growing interest in this subfield.

There are several sources and explanations for the disparity in accuracy, the most prominent being the quality of the disparity map estimator itself. This paper and Wang et al. [26] both utilized PSMnet, but the likelihood of small differences in configuration between the two may have resulted in significant performance differences. The second major source of stereo camera error may lie in the KITTI dataset source images. Minor inaccuracies at even a few pixels can degrade system performance, and thus more investigation is needed into generating the most accurate disparity map and subsequent point cloud possible. Clearly, the latter half of SPCLnet and the pseudo-point cloud network by [26], Frustum Pointnets by [20], can achieve near state-of-the-art results when given the right information, i.e. accurate point cloud. Looking at Figure 4.13, all stereo networks seem to have an upper limit on performance: none perform beyond a certain recall level, such as 0.5 at Moderate difficulty. This presents an opportunity to explore new datasets that include both lidar and stereo image data. As of now, the KITTI object detection dataset, initially published in 2012 [6], is roughly 7 years old, and most of the images on this dataset are about 0.5 megapixels (MP) in resolution. This is simply not enough to provide adequate information for generating a disparity map, aside from apparent need to generate one’s own set of ground truth data from the object detection dataset in order to even train a stereo network on it. New datasets are showing some level of promise, such as the Apolloscape dataset [9] or even moderately aged datasets such as [3], and future research would benefit greatly from the inclusion of disparity map ground truth data.

Real-time stereo-based 3D estimation is within reach for a few simple reasons. For one, there are already real-time stereo disparity map estimators and commercial products, although their output is generally only a point cloud at best, if not a disparity map. Next,

optimizations may be made in the network architecture presented in this paper. The data are transformed several times when moving between the 2.5D disparity map format to the 3D point cloud format, and this may be simplified in a unified pipeline, such as moving the reconstruction step to a later point with a subset of points.

A Performance Metrics

Examples are given below for explicitly calculating the performance metrics used in determining the quality of 3D bounding box (“bbox”) predictions generated by the Stereo Point Cloud network (SPCLnet)

A.1 Intersection Over Union

Recall that IOU is critical in vision networks for determining whether a result is a true positive (TP) or a false positive (FP). To assist in understanding IOU, a code snippet as well as an example are presented. All aspects of calculating the IOU (including area and intersection) are broken up into multiple pieces, but presented together below. For n-dimensions, the code (presented here in python) is as follows:

```
import numpy as np
def extent(box,inclusive=False):
    o = 1 if(inclusive) else 0 # add '1' if inclusive is true
    b = np.array(box).reshape((2,-1)).T # now in internal convention
    return np.product([i[1]-i[0]+o for i in b])
def intersection(box1,box2,inclusive=False):
    o = 1 if(inclusive) else 0 # add '1' if inclusive is true
    b1=np.array(box1).reshape((2,-1)).T
    b2=np.array(box2).reshape((2,-1)).T # internal convention
    c=np.stack((b1,b2),2)
    # for each dimension, get (min(upperbound)-max(lowerbound)) and get product
    val=1
    for i in range(len(b1)):
        ans=np.min(c[i,1,:])-np.max(c[i,0,:])
        val*=max(ans,0) # if have negative dimension, have no intersection
    return val

def IOU(b1,b2,inclusive=False,criterion=-1):
    inter = intersection(b1,b2,inclusive)
    union = extent(b1,inclusive)+extent(b2,inclusive)-inter
    if(criterion== -1):
        return inter / union
    elif(criterion== 0):
        return inter / extent(b1,inclusive)
    else:
        return inter / extent(b2,inclusive)
```

Figure A.1: Python implementation of generalized IOU calculation.

A.1.1 Example: IOU of a Ground Truth and Prediction Label

Suppose there is an image, as given below, where there is a ground truth label ‘gt’ and a prediction label ‘pr’ with 2D bounding boxes formatted as $[x_1, y_1, x_2, y_2]$, all units in pixels. To determine the IOU of the image, the calculations are listed below. Because boxes represent pixel values, remember to add “1” to each dimension.

1. Find area of each bounding box:

$$A_{gt} = (x_2 - x_1 + 1) * (y_2 - y_1 + 1) = 16,335 \text{ [px}^2\text{]}, A_{pr} = 12905 \text{ [px}^2\text{]}$$

2. Find overlapping area, e.g. intersection (see A.1 for more info): $I = 11455 \text{ [px}^2\text{]}$

3. Calculate IOU: $\frac{I}{A_{gt}+A_{pr}-I} = 0.644$

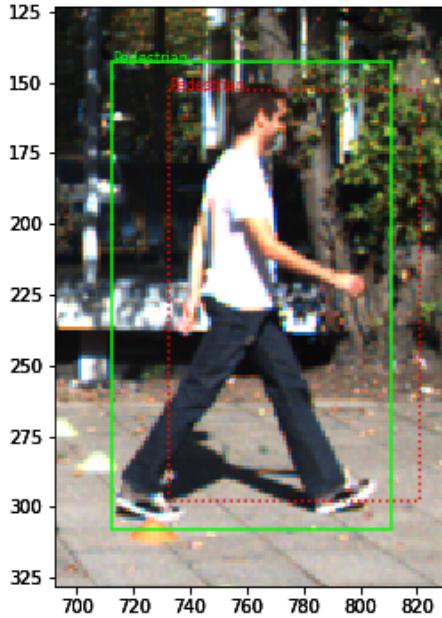


Figure A.2: IOU calculation example. Ground truth (green solid) has BB: [712,143,810,307]. Prediction (red dotted) has BB: [732,153,820,297]. IOU is 0.644.

A.2 Precision & Recall

Gathering all predictions together, one may then take the confidence score and outcome (TP, FP) of each prediction and create a precision-recall curve. To avoid extra steps regarding calculating a set of bounding boxes and determining IOU score, the following example uses a set of randomly generated outcomes values and confidence scores, listed below in Table A.1.

Table A.1: Precision-recall dummy data. Outcomes are either TP or FP, given as a ‘1’ or ‘0’, respectively. Confidence values are how confident the “detection” was. In the true evaluation code, TP/FP is determined internally, and confidence values are the “scores” in each detection label. Rows are conveniently given in order of descending confidence score.

Outcomes	Confidence
1	1.000
1	0.994
1	0.981
0	0.932
1	0.919
0	0.879
0	0.854
1	0.846
0	0.768
1	0.763
1	0.752
0	0.728
1	0.728
0	0.706
0	0.653
1	0.653
0	0.502
0	0.309
0	0.298
0	0.223

The steps are as follow:

1. Begin with a confidence score vector and a outcomes vector
2. Sort scores and corresponding outcomes by descending score value
3. OPTIONAL: Identify unique index locations for approximate threshold indices (done in official code, not necessary)
4. Create “cumulative sum” TP-vector and FP vectors
5. Calculate precision and recall in vector form (unrefined)
6. Refine precision vector pr and recall vector re (OPTIONAL: remove indices after TP vector stops increasing)
7. OPTIONAL: Sort by descending recall value (done in official code, not necessary)

Step 1: Obtain data

To begin, a randomly generated set of results (outcome and confidence scores) are given in Table A.1. One may imagine that this table was generated after comparing car 3D bounding boxes against a set of ground truths and ensuring the IOU value was above a minimum threshold.

Step 2: Sort data by descending confidence score

In the case of this example, the values already sorted in this manner. Care must be taken to ensure that both a given result’s confidence AND outcome (True or False) are kept in the same index location.

```
1 indices=np.argsort(ypred)[::-1]
2 ypred=ypred[indices]
3 ytrue=ytrue[indices]
```

Step 3 (OPTIONAL): Obtain indices of unique vectors

The purpose of this is to simply avoid needless calculation later, although empirically it has been found that the final graph and corresponding AP score are either identical or nearly so. This step WILL be carried out in this example, but may be omitted. Looking at the table, and assuming zero-based indexing as is typical in Python and other programming languages, the resulting vector is: [0,1,2,3,4,5,6,7,8,9,10,12,13,14,15,16,17,18]. Notice that index 11 and 19 are missing. ADDITIONALLY, the final index must be added to ensure that the graph terminates properly. If this step is not carried out, the correct value of the vector `threshold_idxs` is generated instead via Line 5 below. To demonstrate the difference, please refer to Figure A.3 below, and note that the AP value is still identical. In the vast majority of usage, the graph does indeed look identical.

```
1 if(get_unique_indices):
2     distinct_value_indices = np.where(np.diff(conf))[0]
3     threshold_idxs = np.append(distinct_value_indices,outcomes.size-1)
4 else:
5     threshold_idxs = np.arange(conf.size) # sequence from 0 to N-1
```

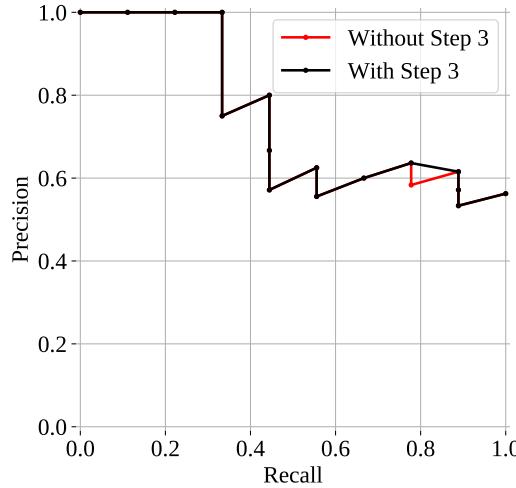


Figure A.3: Comparison of possible graph differences if Step 3 is skipped.

Step 4: Get Cumulative Sum for TP & FP

The generation of an aggregate TP and FP vector enables precision and recall to be calculated as a vector as well, not only as scalar values. This is achieved by calculating the cumulative sum of the TP and FP arrays.

```

1 tps = np.cumsum(outcomes)[threshold_idxs]
2 fps = 1+threshold_idxs - tps

```

Step 5: Calculate Precision & Recall

Precision and recall are calculated, but are still in a “raw” form after this step. This means that there is some extra, unnecessary data as well as a theoretical point to be appended. As given by Equations 4.2, precision calculated, with some adaptation for code. Recall is often calculated differently than the theoretical form, here given as a division of the cumulative sum of true positive values by the final value.

```

1 precision = tps/(tps+fps)
2 precision[np.isnan(precision)]=0
3 recall = tps/tps[-1]

```

Step 6: Refine the data

Here, indices are removed after the cumulative sum of TP ceases to increase; additionally, artificial point at recall=0, precision=1 is added to the vectors (inserted at index 0). The first modification is optional, merely an aesthetic / time-saving choice, shown below in Figure A.4, but the artificial point is necessary for an accurate calculation of AP.

```

1 if(remove_extra_indices):
2     lastind = tps.searchsorted(tps[-1])+1
3 else:
4     lastind = -1
5 recall = np.append(0,recall[:lastind])
6 precision = np.append(1,precision[:lastind])

```

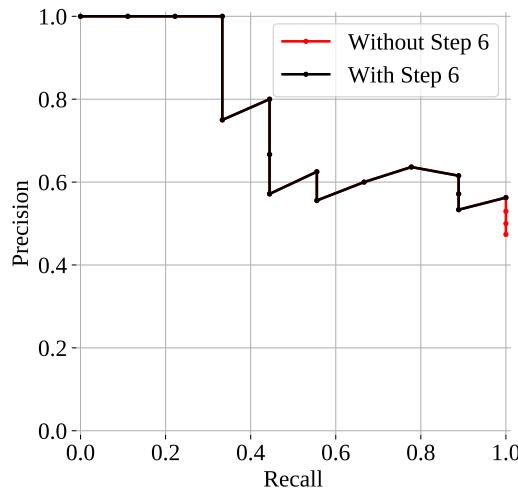


Figure A.4: Comparison of possible graph differences if Step 6 is skipped.

Step 7 (OPTIONAL): Sort by Descending Recall Value

This is an arbitrary step completed by the official matplotlib code, but may be carried out nonetheless if a certain convention is to be followed. It must be noted, however, that the proper calculation of AP requires the precision and recall vectors to be ordered by *ascending* recall value.

```

1 recall=recall[::-1]
2 precision=precision[::-1]
```

Thus, the final values of precision and recall are calculated as given below in Table A.2 and graphed in Figure A.5

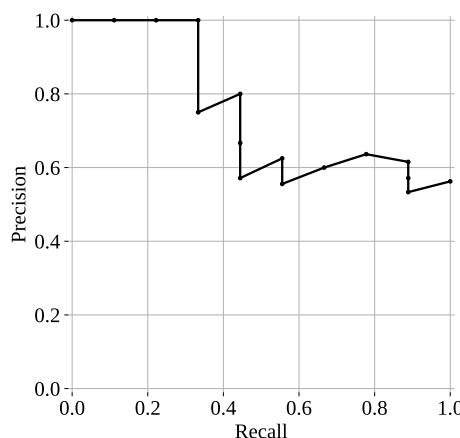


Figure A.5: Precision-recall curve if all steps are followed. Mathematically, AP value is same as without optional steps.

Table A.2: Precision-recall results, following all steps.

Recall	Precision
1.00	0.56
0.89	0.53
0.89	0.57
0.89	0.62
0.78	0.64
0.67	0.60
0.56	0.56
0.56	0.62
0.44	0.57
0.44	0.67
0.44	0.80
0.33	0.75
0.33	1.00
0.22	1.00
0.11	1.00
0.00	1.00

A.3 Calculating Average Precision

To obtain the Average Precision from the previous example, simply follow Equation 4.4, or as implemented below (in a non-vectorized form). The AP of the values in Table A.2 is 0.75992.

```

1 apsum=0
2 for i in range(0,len(precision)-1):
3     print(i,end=' ')
4     apsum+=(recall[i+1]-recall[i])*precision[i+1]

```

B About the KITTI Dataset

B.1 Introduction

To better understand the actual data that is being worked on, the KITTI dataset will be explained in-depth here. The acronym KITTI is a combination of “KIT” (Karlsruhe Institute of Technology) and “TTI” (Toyota Technical Institute, Chicago), the two cooperating institutions on the project. The very first thing to know about the KITTI dataset is that it is actually a set of datasets, each dataset specialized for some specific AI task. This means that there are some number of evaluation tasks / benchmarks, and a not-necessarily-unique dataset is used to enable completion of that task. The individual tasks are listed below with a brief description:

- Stereo, Optical Flow, Sceneflow (2012 & 2015): An older (2012) and updated (2015) task comprised of 200 training and 200 test scenes. Each scene consists of a left and right image as well as a “multi-view extension”.
- Depth Completion / Estimation: A task comprised of about 93 thousand scenes containing RGB images and lidar scans.
- Visual Odometry / SLAM: 11 training and 11 testing stereo sequences with relevant grayscale, color, lidar, and ground truth poses.
- 2D / 3D / Bird’s Eye View (BEV) Detection: 7481 training and 7518 test scenes with relevant color images, temporally preceding images, and lidar
- Single-/Multi- Object Tracking: An older (single) and in-development (multi) task consisting of multiple training and test sequences with relevant image, lidar, and GPS/IMU data.
- Road / Lane Detection: Around 289 training and 290 test images used for detecting various road scenes, containing image and lidar information.
- Semantic / Instance Segmentation: A task comprised of a set of stereo images like the Stereo task, but containing pixel-level localization of each class.

Each evaluation task also brings with it a “dev kit” (a set of files that explain some of the technical information as well as helpful code), relevant camera / sensor calibration data of some kind, as well as some kind of ground truth label / file for each scene to assist with training. It should be noted that raw data of each dataset is available as well, but these are often unused and only kept as original source files for transparency.

Each task is associated with one or more papers published by KIT and TTI, but these will not all be listed here. Instead, we shall now explain two datasets, especially the most relevant one: the KITTI object detection dataset, referred to as the KITTI dataset.

B.2 The Stereo Dataset

The stereo dataset, specifically 2015, will be briefly covered to describe its merits as well as why it presented overlap issues with the object detection dataset. What makes the stereo dataset important is that it is the dataset that was initially used to finetune the Pyramid Stereo Matching network and also the dataset used to help it achieve such a high position in the stereo benchmark.

The dataset itself is made up primarily of 200 training scenes and 200 testing scenes. The training scenes contain ground truth disparity maps, while the testing scenes do not and are used for final evaluation and submission to the benchmark website. For the project, only the training scenes were used. Each training scene is comprised of a left-hand side (LHS) color image, a RHS color image, a calibration file (containing transformation and other matrices), and a ground truth disparity file. There is also an available “multi-view extension”, containing 20 extra frames per scene, but this was not used. Figure B.1 below shows all relevant data in a single sample scene: the left and right images, as well as a disparity map ground truth.

The ground truth labels were generated by a “semi-automatic” process, as described by Menze and Geiger: the process was to “extract disparity maps directly from the 3D information in the laser scans and fit geometrically accurate CAD models to moving 3D point clouds” [16]. This explains several things about the ground truth disparity map in Figure B.1(c): the image is generally sparse, has no sample values above a certain horizontal line, and has a high density of pixels in the region where a car is outlined.



Figure B.1: Sample scene. Index 000000_10 of the KITTI stereo dataset.

Ultimately, however, this dataset was not used for the primary reason described by [26]: there was an unacceptable amount of overlap between the stereo dataset training data and the similar but unrelated object detection dataset. This could lead to validation images being used in training, an issue known as data leakage. Figure 4.8, shown in Section 3, gives an example of a scene in the stereo dataset nearly matches a scene in the object detection dataset.

B.3 The Object Detection Dataset & Statistics

The Object Detection dataset is the primary source of sensor data and media for this project, including PSMnet once it was determined that the stereo dataset had too much similarity with this dataset. This similarity could lead to data leakage, contaminating the integrity of the results. The Object Detection dataset is comprised of 7481 training scenes and 7518 testing scenes. The testing scenes were ignored for the purposes of this project, since they are unlabeled for KITTI benchmarking. Each scene in the training dataset contains a LHS (left-hand side) image, RHS image, lidar data, calibration data, and ground truth labels. An example of the images and lidar are given below in Figure B.2.



Figure B.2: Sample scene from object detection dataset. Index 15 of the KITTI Object Detection dataset.

In addition to the above lidar projection, the point cloud may be better visualized with an isometric view, rather than one from the camera’s perspective. This is provided below. As is immediately obvious in Figure B.3 and from knowledge of the hardware itself, a lidar scan is capable of capturing points surrounding the sensor in a 360° horizontal view. However, because of the nature of a typical color camera, points that are outside the camera’s field of view are filtered out.

Another key piece of the object detection dataset is the label data. Ground truth labels are simply lines of text that contain an array of information about each instance of an object in the scene. An example label with value names is given below in Table B.1. There are 15 values in each label, and an extra 16th “Score” value for prediction labels, a decimal value in range [0,1]. The raw text for this label simply looks like so:

```
Car 0.89 0 2.29 0.00 194.70 414.71 373.00 1.57 1.67 4.14 -2.75 1.70 4.10 1.72
```

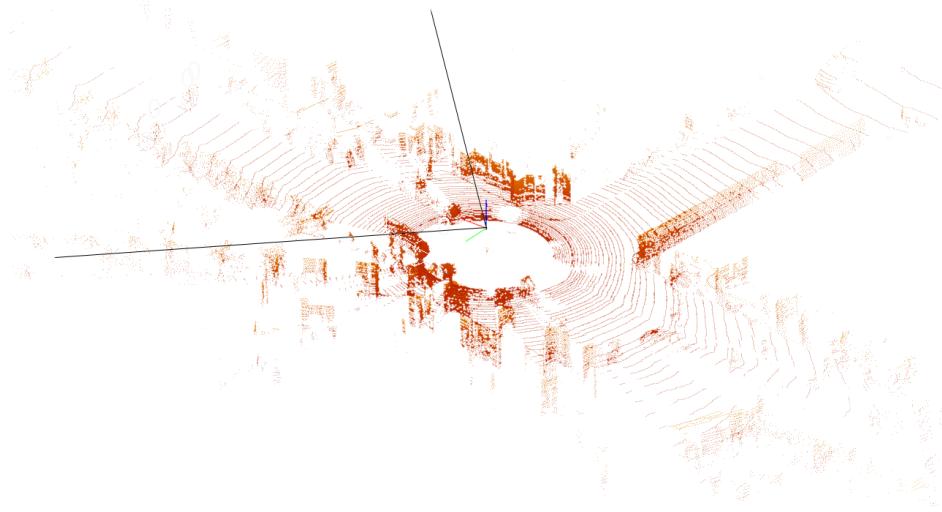


Figure B.3: Lidar pointcloud, alternate view. 3D bounding boxes are included as well as the horizontal field of view (angled lines leading away from coordinate system origin). Index 15 of the KITTI Object Detection dataset

Table B.1: KITTI object detection sample label from a given scene. Index 15 of the KITTI Object Detection dataset

Class	Trun	Occl	Obs	BBx1	BBy1	BBx2	
Car	0.89	0	2.29	0.00	194.70	414.71	
BBy2	HT	WD	DP	Xc	Yc	Zc	Roty
373.00	1.57	1.67	4.14	-2.75	1.70	4.10	1.72

In order to better understand the meaning of each value, each dataset provides a “development kit” that contains a “readme.txt” file. In the object detection devkit, the following text reproduced in Figure B.4. Some of the names are different than what is used, but the order and purpose are identical.

Ct.	Name	Description
1	type	Describes the type of object: ‘Car’, ‘Van’, ‘Truck’, ‘Pedestrian’, ‘Person_sitting’, ‘Cyclist’, ‘Tram’, ‘Misc’ or ‘DontCare’
1	truncated	Float from 0 (non-truncated) to 1 (truncated), where truncated refers to the object leaving image boundaries
1	occluded	Integer (0,1,2,3) indicating occlusion state: 0 = fully visible, 1 = partly occluded 2 = largely occluded, 3 = unknown
1	alpha	Observation angle of object, ranging [-pi..pi]
4	bbox	2D bounding box of object in the image (0-based index): contains left, top, right, bottom pixel coordinates
3	dimensions	3D object dimensions: height, width, length (in meters)
3	location	3D object location x,y,z in camera coordinates (in meters)
1	rotation_y	Rotation ry around Y-axis in camera coordinates [-pi..pi]
1	score	Only for results: Float, indicating confidence in detection, needed for p/r curves, higher is better.

Figure B.4: Original text from readme.txt file of devkit_object resources. The columns are: Ct. (“count”), Name (of value), and Description.

Beyond a few samples of the dataset, some statistics about the dataset can be found in both the accompanying paper as well as by looking through the dataset itself. From the

B About the KITTI Dataset

original 2012 paper, [6] provided a subfigure of the class count across both training and validation data, reproduced below in Figure B.5. When observing only the training data, the count of each class can be described. A quick summary of each value across all labels in the training dataset is given below in Table B.2. Note that the min/max range ignores the DontCare class and its values, such as -1000 for z_c .

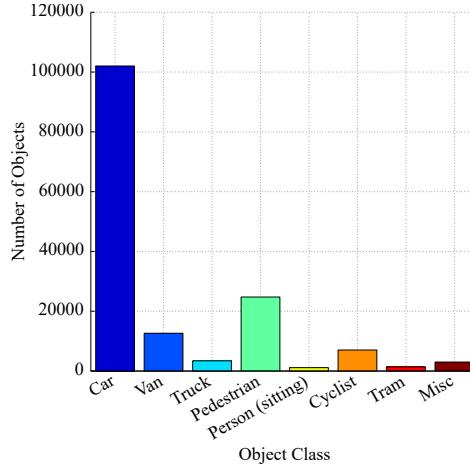


Figure B.5: Object class count in KITTI object detection dataset, reproduced from [6].

Table B.2: KITTI dataset label statistics. Each numbered value's minimum and max values are given. Variables xc , yc , and zc are distance to the bounding box center of mass, and pixel values are given to sub-pixel accuracy.

Name	DataType	Unit	min	max
Class	string	...	n/a	n/a
Truncation	float	...	0.00	1.00
Occlusion	integer	...	0	3
ObsAngle	float	radians	-3.14	3.14
bbx1	float	pixels	0.00	1240.54
bby1	float	pixels	0.00	323.98
bbx2	float	pixels	14.71	1241.00
bby2	float	pixels	140.79	375.00
ht	float	meters	0.76	4.20
wd	float	meters	0.30	3.01
dp	float	meters	0.20	35.24
xc	float	meters	-44.03	40.06
yc	float	meters	-2.14	5.93
zc	float	meters	-3.55	146.85
rotY	float	radians	-3.14	3.14

The final bit of interesting statistical information may be found by counting the number of instances of each class. This is already provided by [6], but in verifying the values provided only in the training data the exact number of instances of each class are listed below in Table B.3.

Table B.3: KITTI object detection class count. Semi-automatically counted through only labeled training data.

Class	Car	Pedestrian	Van	Cyclist	Truck	Misc	Tram	Person_sitting
Count	28742	4487	2914	1627	1094	973	511	222

B.4 Evaluation Difficulty Levels

One of the last but certainly not least important aspects of the KITTI dataset is the difficulty rating system. The KITTI dataset certainly has a variety of scenes and training data, but it also allows scoring to be performed on three difficulty levels, defined as follow:

1. Easy: 40 pixel min bbox height, max level: Fully visible (0), max truncation: 15%
2. Moderate: 25 px minimum bbox height, max occlusion level: 1, max truncation: 30%
3. Hard: 25 px min bbox height, max occlusion level: 2, max truncation: 50%

Naturally, the “Easy” difficulty typically earns a higher AP score than “Moderate” (or “Medium”), and “Moderate” in turn typically yields a higher AP score than “Hard”. The official final score that a network receives when submitted to the KITTI Leaderboard is the “Moderate” mean AP score. This paper specifically deals with the “Car” class, so we may be able to better understand how these difficulties affect other parameters. Specifically related to the “Car” class, 3D bounding box is considered a detection at 70% overlap with a ground truth. The other two classes, “Pedestrian” and “Cyclist”, only require an overlap of 50%. The remaining other classes are not typically included in evaluation and scoring.

In the paper [26], the “Easy” evaluation for the “Car” class is said to test on car labels located mostly within 30m of the ego vehicle, which was also claimed by another group, [28]. This was verified, and it is shown below in Figure B.6.

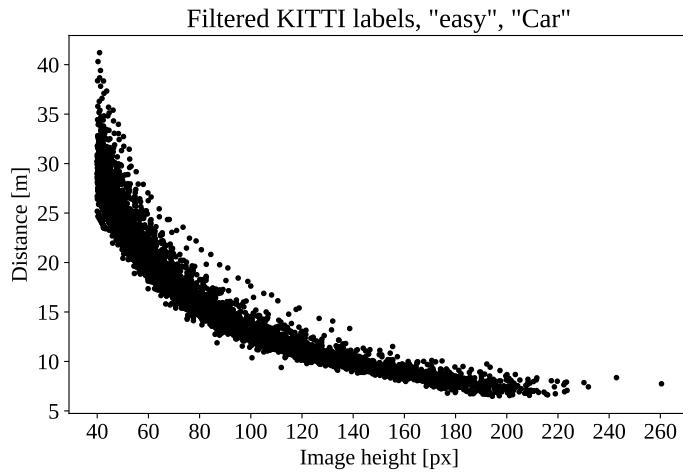


Figure B.6: Filtered set of labeled training data containing only labels with the “Car” class and those which are defined as “Easy” difficulty. Just over 3% of these filtered labels are beyond 30m. An interesting observation that may be made from the pixel height / distance relationship is that they are inversely correlated.

References

- [1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.
- [2] Jia-Ren Chang and Yong-Sheng Chen. Pyramid Stereo Matching Network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5410–5418, 2018.
- [3] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. *CoRR*, abs/1604.01685, 2016.
- [4] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015.
- [5] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.
- [6] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
- [7] Kristian Gonzalez, Markus Schneider, and Christoph Wiedemann. Smart3d project code repository, 2019.
- [8] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on pattern analysis and machine intelligence*, 30(2):328–341, 2007.
- [9] Xinyu Huang, Xinjing Cheng, Qichuan Geng, Binbin Cao, Dingfu Zhou, Peng Wang, Yuanqing Lin, and Ruigang Yang. The ApolloScape Dataset for Autonomous Driving. *arXiv: 1803.06184*, 2018.
- [10] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and others. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015.
- [11] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. End-to-end learning of geometry and context for deep stereo regression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 66–75, 2017.
- [12] Peiliang Li, Xiaozhi Chen, and Shaojie Shen. Stereo R-CNN based 3d Object Detection for Autonomous Driving. *arXiv preprint arXiv:1902.09738*, 2019.

- [13] Zhengfa Liang, Yiliu Feng, Yulan Guo, Hengzhu Liu, Wei Chen, Linbo Qiao, Li Zhou, and Jianfeng Zhang. Learning for Disparity Estimation through Feature Constancy. *Computer Vision and Pattern Recognition*, 2018.
- [14] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.
- [15] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.
- [16] Moritz Menze and Andreas Geiger. Object Scene Flow for Autonomous Vehicles. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.
- [17] Moritz Menze, Christian Heipke, and Andreas Geiger. The KITTI Vision Benchmark Suite - Stereo Evaluation 2015. http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d, March 2019. [Online; accessed 25-March-2019].
- [18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [19] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017.
- [20] Charles Ruizhongtai Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum PointNets for 3d Object Detection from RGB-D Data. *CoRR*, abs/1711.08488, 2017.
- [21] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.
- [22] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*, abs/1506.01497, 2015.
- [23] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3):7–42, 2002.
- [24] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015.
- [25] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.

- [26] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Weinberger. Pseudo-LiDAR from Visual Depth Estimation: Bridging the Gap in 3d Object Detection for Autonomous Driving. In *CVPR*, 2019.
- [27] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.
- [28] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018.
- [29] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Std: Sparse-to-dense 3d object detector for point cloud. *arXiv preprint arXiv:1907.10471*, 2019.
- [30] Yiming Zeng, Yu Hu, Shice Liu, Jing Ye, Yinhe Han, Xiaowei Li, and Ninghui Sun. Rt3d: Real-time 3-d vehicle detection in lidar point cloud for autonomous driving. *IEEE Robotics and Automation Letters*, 3(4):3434–3440, 2018.