# Learning to Optimally Segment Point Clouds

Peiyun Hu, David Held, and Deva Ramanan

***Abstract*—We focus on the problem of class-agnostic instance segmentation of LiDAR point clouds. We propose an approach that combines graph-theoretic search with data-driven learning: it searches over a set of candidate segmentations and returns one where individual segments score well according to a data-driven point-based model of "objectness". We prove that if we score a segmentation by the worst objectness among its individual segments, there is an efficient algorithm that finds the optimal worst-case segmentation among an exponentially large number of candidate segmentations. We also present an efficient algorithm for the average-case. For evaluation, we repurpose KITTI 3D detection as a segmentation benchmark and empirically demonstrate that our algorithms significantly outperform past bottom-up segmentation approaches and top-down object-based algorithms on segmenting point clouds.**

***Index Terms*—Computer vision for automation, deep learning in robotics and automation, object detection, segmentation and categorization, semantic scene understanding.**

## I. Introduction

PERCEPTION for autonomous robots presents a collection of compelling challenges for computer vision. We focus on the application of autonomous vehicles. This domain has three notable properties that tend not to surface in traditional vision applications: (1) 3D sensing in the form of LiDAR technology, which exhibits different properties than traditional 3D vision captured through stereo or structured light. Despite significant work in this area, the right representation for such sparse 3D signals still remains an open question. (2) Contemporary approaches to object detection and scene understanding tend to be closed-world, where the task is predicting 1-of-N possible labels. But autonomous systems require the ability to recognize all possible obstacles and movers - e.g., a piece of road debris must be avoided regardless of what *name* it has. Such understanding is crucial from a safety perspective. Historically, this has been formulated as a perceptual grouping or bottom-up segmentation task, which is typically addressed with different approaches. (3) Finally, practical autonomous robotics makes heavy use of perceptual priors in the forms of geometric maps and assumptions on LiDAR geometry. Indeed, prior map was a crucial component among finishing entries in the DARPA Urban Grand Challenge [1], [2].

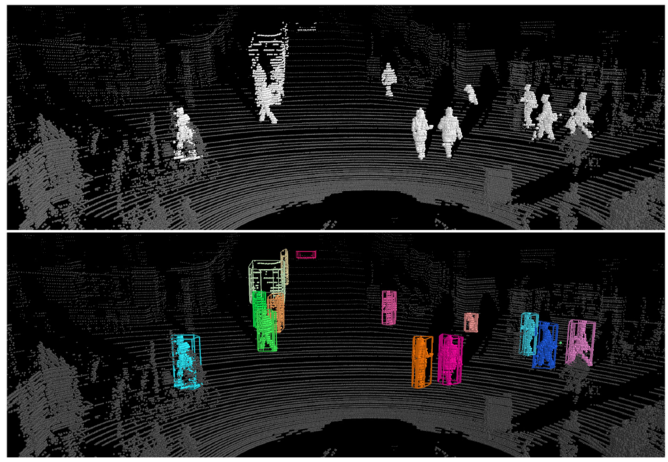


Fig. 1. Our proposed algorithm takes a pre-processed LiDAR point cloud with background removed (top) and produces a class-agnostic instance-level segmentation over all foreground points (bottom). For visualization, we use a different color for each segment and plot an extruded polygon to show the spatial extent.

*Motivation:* In this work, we focus on the problem of class-agnostic instance segmentation of LiDAR point clouds (Figure 1) in an open-world setting. We carefully mix graph-theoretic algorithms with data-driven learning. Data-driven learning has made an undeniable impact on computer vision, but it is difficult to make guarantees about performance when processing out-of-sample data from an open world. Geometric graph-based approaches for segmentation tend not to require training and so are less-like to overfit, but also tend to be brittle.

*Approach:* Our approach searches over an exponentially-large space of candidate segmentations and returns one where individual segments score well according to a data-driven point-based model of "objectness" [3]. We demonstrate that one can repurpose existing closed-world point networks [4] for bottom-up perceptual grouping tasks that generalize to objects rarely seen during training.

*Optimality:* We prove that our approach produces *optimal* segmentations according to a specific definition. First, we restrict the search into a subset of segmentations that are consistent with a hierarchical grouping of a point cloud sweep. Such hierarchical groups can be readily produced with agglomerative clustering [5], HDBSCAN [6], or hierarchical graph-based algorithms [7].

Manuscript received September 10, 2019; accepted January 2, 2020. Date of publication January 9, 2020; date of current version January 27, 2020. This letter was recommended for publication by Associate Editor F. Tombari and Editor E. Marchand upon evaluation of the reviewers' comments. This work was supported by the CMU Argo AI Center for Autonomous Vehicle Research. *(David Held and Deva Ramanan contributed equally to this work.) (Corresponding author: Peiyun Hu.)*

P. Hu and D. Held are with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: peiyunh@cs.cmu.edu; dheld@andrew.cmu.edu).

D. Ramanan is with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA, and also with the Argo AI, Pittsburgh, PA 15213 USA (e-mail: deva@cs.cmu.edu).

This letter has supplementary downloadable material available at https://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/LRA.2020.2965389

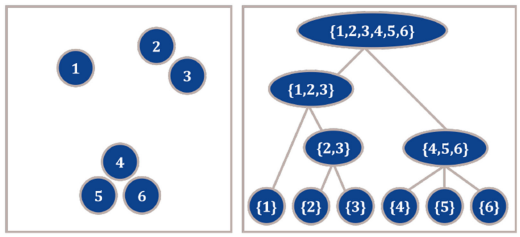



Fig. 2. On the left, we visualize a set with 6 points. According to Bell number, one will find 203 unique segmentations (partitions). Most of these are arbitrary and do not respect local geometry, e.g. $\{\{1,2,5\},\{3,4,6\}\}$. On the right, we implement geometric constraints with a tree formed by hierarchical grouping. Every vertex cut of this tree is automatically a segmentation that respects local geometry encoded by the tree, e.g. $\{\{1\},\{2,3\},\{4,5,6\}\}$.

Naive methods for producing a segmentation might apply a global threshold over the whole hierarchy. It turns out that one can produce an exponentially-large set of segmentations by applying different thresholds at different branches. We introduce efficient algorithms that search over this space of tree-consistent segmentations (Figure 2) and return the one that maximizes a global segmentation score that is computed by aggregating local objectness scores of individual segments.

*Evaluation:* We demonstrate empirical results on KITTI, a benchmark originally designed for closed-world object detection. Following past work, we repurpose it for open-world 3D segmentation [8]. We compare to existing bottom-up approaches [9] and state-of-the-art LiDAR-based object detectors after converting their output 3D bounding boxes to a point cloud segmentation. We demonstrate that our approaches outperform both baselines on less common classes.

## II. Related Work

Robust 3D object detection is crucial for downstream applications such as semantic understanding [10] and tracking [11]. Comparing to monocular 3D detection [12], we focus on LiDAR-based solutions in this letter.

*LiDAR segmentation:* Classic LiDAR segmentation algorithms use bottom-up grouping such as flood-filling [13], connected components [14], or density-based clustering [6]. Bottom-up strategies can also be applied on LiDAR sequences, allowing for motion as an additional cue [15]–[17]. Oftentimes such approaches are tuned for particular object categories such as cars. Our work differs in its use of static, single-frame cues that are not object-specific.

*LiDAR object detection:* There is an ever-increasing literature on data-driven object detection with LiDAR point clouds. Early approaches include fusion-based models that combine LiDAR and imagery [18], tracking-based detectors [19] and voxel-based classifiers [20]–[22]. We have seen approaches built upon raw point clouds such as PointRCNN [23]. Our approach is most related to Frustum PointNet [24] in the way we use pooled point cloud representation [4]. Our work differs in that we do not make use of camera input, and most notably, focus on *all* possible objects in an open world. Specifically, we compare to [18], [21], [22], [25] as a representative sample of the literature.

*Perceptual grouping:* Our graph-based approach is inspired by a long line of classic work on graph-theoretic perceptual grouping, dating back to normalized cuts [23], graph cuts [26], and spanning-tree approaches [27]. Such methods are typically used with hand-designed features, while we make use of data-driven techniques for learning a shape-based segment classifier.

*Image segmentation:* The idea of searching for an optimal image segmentation given a hierarchical image segmentation tree has been explored. [28] formulates neuron segmentation on electron microscopy images as a *maximum a posteriori* (MAP) labeling task on a tree-structured graph. It can be made equivalent to our search under certain conditions. [29] tackles the problem of class-agnostic instance segmentation in image space by exploiting visual appearance and motion. We discuss more in Sections III and IV-B.

## III. Approach

For 3D object point segmentation, the input is a 3D point cloud, which contains an *unknown* number of objects. The goal is to produce a point segmentation, in which every segment contains points from one and only one object.

*Segmentation:* A global *segmentation* $P_X$ is a partition of a set of points $X = \{x_i\}_{i=1}^N$ into subsets of points, i.e. $P_X = \{C_i\}_{i=1}^M$, where $M$ denotes the number of segments and $C_i \subset X$. We refer to each $C_i$ as a local *segment*. Importantly, every point exists in one and only one segment, meaning $\cup_{i=1}^M C_i = X$ and $\forall i \neq j, C_i \cap C_j = \emptyset$.

*Tree-consistent segmentations:* Let us use $S_X$ to denote the set of all possible global segmentations on $X$, i.e. all possible $P_X$. Without constraints, the size of $S_X$ is exponential in $N$ (i.e. the Bell number). In practice, we can reduce the number of candidates by enforcing geometric constraints. In this work, we implement the constraints by grouping all points hierarchically into a tree structure $T_X$. We will discuss how to build such a tree structure based on local geometric cues in Section III-D. For now let us assume the tree is given.

Once we specify the tree, we can focus on a strictly smaller set of segmentations that respect local geometry. We denote such set as $S_{X,T}$ and call them *tree-consistent* segmentations. As a reference, the size of $S_{X,T}$ is still exponential in $N$, when $T_X$ is a balanced binary tree.[1] We further illustrate the relationship between $S_X$ and $S_{X,T}$ with an example in Figure 2. Any tree-consistent segmentation from $S_{X,T}$ corresponds to a vertex cut set of the tree $T$, i.e. a set of tree nodes, which satisfy the following constraints: (1) for each node in the vertex cut, its ancestor and itself cannot both be in the cut and (2) each leaf node must have itself or its ancestor in the cut. Such relationship allows us to design efficient tree searching algorithms, as we will see later.

*Segment score:* Before we discuss how to score a *global* segmentation, we first introduce how to score a *local* segment. Given a local segment $C \subset X$, we define a function $f(C;\theta) : C \mapsto [0,1]$ that predicts a given segment's "objectness," where

[1]One can derive recurrence on the number of segmentations between depth d+1 and d as $K_{d+1} = K_d^2 + 1$ with $K_1 = 2$. Since $K_d > 2^{2(d-1)}$, $K_d/N_d > 2^{d-2}$, where $N_d = 2^d$ represents the number of leaves, it suggests the number of segmentations *at least* outgrow the number of leaves exponentially.

$\theta$ represents the parameters. One can implement such a function with a PointNet++, where $\theta$ would represent weights of the PointNet++. We will discuss how to learn this function in Section III-C. For now let us assume it is given.

*Segmentation score:* We now introduce how to score a *global* segmentation. Given a global segmentation $P_X = \{C_i\}_{i=1}^M$, we define its score $F(P_X;\theta) : P_X \mapsto [0,1]$ by aggregating over local objectness of its individual segments. Specifically, we introduce *worst-case* segmentation and *average-case* segmentation. Note that our objective can be made equivalent to [28] if we score a segmentation as the *sum* of its local segment scores. As we see in Section IV-B, this objective produces much larger oversegmentation error.

### A. Worst-Case Segmentation

*Worst-case* segmentation scores a global segmentation as the *worst* objectness among its local segments:

$$F_{\min}(P_X;\theta) = \min_i f(C_i;\theta), i \in 1 \dots M \tag{1}$$

where $P_X \in S_{X,T}$, $P_X = \{C_i\}_{i=1}^M$, and $C_i \subset X$. We define $P^*_{X,\min}$ as the *optimal worst-case segmentation* if

$$P^*_{X,\min} = \underset{P_X \in S_{X,T}}{\operatorname{argmax}} F_{\min}(P_X;\theta) \tag{2}$$

It turns out the problem of finding optimal worst-case segmentation has optimal substructure (Theorem 1), allowing us to find the global optimum efficiently with dynamic programming (Algorithm 1).

We briefly describe how the algorithm works. Given a set of points $X$ and a tree $T_X$, OPTMINSEG($X, T_X$) (Algorithm 1) produces an optimal worst-case segmentation $P^*_{X,\min}$ with score $F^*_{\min}(P^*_{X,\min};\theta)$. For simplicity, we refer to a node in the tree by the set of points it is associated with. The algorithm starts from the root node $X$ and chooses between a coarse segmentation ($\{X\}$) and a fine one. The fine segmentation will be the union of all $X$'s children's optimal worst-case segmentation, which can be computed recursively. The algorithm would first traverse down to the leaf nodes, representing the finest segmentation. Then it will make its way up, during which it finalizes optimal segmentations for each intermediate node by making local coarse vs. fine decisions. Eventually, it returns to the root node and produces an optimal worst-case global segmentation.

*Lemma 1:* Given pairs of non-empty sets that contain real numbers $(X_1, Y_1), \dots, (X_n, Y_n)$,

$$\forall i, \min_{x \in X_i} x \le \min_{y \in Y_i} y \Rightarrow \min_{x \in \cup_i X_i} x \le \min_{y \in \cup_i Y_i} y \tag{3}$$

*Theorem 1:* Given $C$ and $T_C$, Algorithm 1 finds the optimal segmentation $P^*_{C,min} = \operatorname{argmax}_{P_C \in S_{C,T}} F_{min}(P_C;\theta)$.

*Proof:* Proof by structural induction.

*Base:* When $N_C = \emptyset$, meaning $C$ corresponds to a leaf node in $T_C$, the algorithm returns $\{C\}$, which is the only segmentation in $S_{C,T}$ and obviously is optimal.

*Induction:* When $N_C \neq \emptyset$, we need to show that the algorithm will produce the optimal segmentation, i.e. $P^*_C$ and $F^*_C$, if it has access to the optimal segmentation for each of $C$'s child $C_i$, i.e. $P^*_{C_i}$ and $F^*_{C_i}$ (optimal substructure).

**Algorithm 1:** Optimal Worst-Case Segmentation.

```
function OPTMINSEG(C, T_C)
  return a segmentation P_C with a score of F_C
  P_C ← {C}
  F_C ← f(C; θ)
  N_C ← set of C's children nodes in T_C
  if N_C ≠ ∅ then
    for C_i in N_C do
      T_{C_i} ← subtree of T_C rooted at C_i
      P_{C_i}, F_{C_i} = OPTMINSEG(C_i, T_{C_i})
      if F_{C_i} ≤ F_C then return P_C, F_C
    if min_i F_{C_i} > F_C then
      P_C ← ∪_i P_{C_i}
      F_C ← min_i F_{C_i}
  return P_C, F_C
```

Let $P_C$ be the segmentation that the algorithm produces for $C$ and let $F_C$ be its score. If $P_C$ were not optimal, there must exist a different segmentation $P'_C$ with score $F'_C$, s.t. $P'_C \neq P_C$ and $F'_C > F_C$. Moreover, $P'_C$ is either a trivial segmentation, i.e. $P'_C = \{C\}$ or the union of segmentations over each of $C$'s children nodes, i.e. $P'_C = \cup_i \{P'_{C_i}\}$.

First, $P'_C$ is not a trivial segmentation. If we assume $P'_C = \{C\}$, we will have $F'_C = f(C;\theta)$. Since $P_C \neq P'_C$, the algorithm chooses $P_C$ over $\{C\}$, therefore, $F_C > f(C;\theta)$. This clearly contradicts with $F'_C > F_C$.

Thus, $P'_C$ has to be the union of segmentations over each of $C$'s children node. According to the inductive hypothesis, the algorithm has the optimal segmentation over each of $C$'s children node, meaning $\forall i, F'_{C_i} \le F^*_{C_i}$ or concretely

$$\forall i, \min_{z \in P'_{C_i}} f(z;\theta) \le \min_{z \in P^*_{C_i}} f(z;\theta) \tag{4}$$

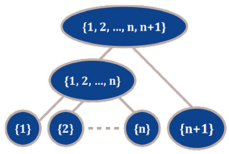
Here, $z$ represents an arbitrary local segment from a segmentation over $C_i$. By applying Lemma 1, we have

$$\min_{z \in \cup_i P'_{C_i}} f(z;\theta) \le \min_{z \in \cup_i P^*_{C_i}} f(z;\theta) \tag{5}$$

On one hand, $P'_C = \cup_i \{P'_{C_i}\}$ has a score of $F'_C = \min_{z \in \cup_i P'_{C_i}} f(z;\theta)$. On the other hand, the algorithm by design chooses the higher scoring one between $P_C = \{C\}$ with a score of $F_C = f(C;\theta)$ and $P_C = \cup_i P^*_{C_i}$ with a score of $F_C = \min_{z \in \cup_i P^*_{C_i}} f(z;\theta)$, ensuring that $F_C \ge \min_{z \in \cup_i P^*_{C_i}} f(z;\theta)$. With these and (5), we conclude $F_C \ge F'_C$, which contradicts the assumption $F'_C > F_C$. ■

*Generality:* Our analysis makes no assumptions about the objectness function $f(C;\theta)$ except the fact that it cannot be affected by the partitioning of other segments. In particular, this would allow objectness to depend on contextual arrangement of surrounding points outside $C$ - e.g., $f(C, X;\theta)$.

*Efficiency:* Given points $X$ and a tree $T_X$ with $N$ leaf nodes, Algorithm 1 guarantees to return the optimal worst-case segmentation after visiting every node in the tree. In practice, it might not visit all nodes. Instead, it skips the rest of sub-trees whenever one sub-tree exhibits lower score than the coarse segmentation




| C | {1} | {2} | ... | {n} | {n+1} | {1, 2, ..., n} | {1,2,...,n+1} |
|---|---|---|---|---|---|---|---|
| f(C) | 0.01 | 0.01 | ... | 0.01 | 0.99 | 0.001 | 0 |
| Fc* | 0.01 | 0.01 | ... | 0.01 | 0.99 | 0.01 | (0.99+0.001)/2 |
| Pc* | {{1}} | {{2}} | ... | {{n}} | {{n+1}} | {{1},{2},...,{n}} | {{1,2,...,n},{n+1}} |

Fig. 3. We illustrate why average-case segmentation does not have optimal substructure. We plot a tree on the left and show local objectness scores on the right. In this case, the optimal average-case segmentation of the root node, i.e. {{1,2,...,n},{n+1}} cannot be formed by the optimal average-case segmentations of its children nodes, i.e. {{1},{2},...,{n}} and {{n+1}}.

(line 9 in Algorithm 1). The algorithm's complexity is linear in $N$ despite the fact that the search space is exponential in $N$.

### B. Average-Case Segmentation

*Average-case* segmentation scores a global segmentation as the *average* objectness among its local segments:

$$F_{\text{avg}}(P_X;\theta) = \frac{1}{M}\sum_{i=1}^{M} f(C_i;\theta) \tag{6}$$

where $P_X \in S_{X,T}$, $P_X = \{C_1,\ldots,C_M\}$, and $C_i \subset X$. We define $P^*_{X,\text{avg}}$ as an *optimal average-case segmentation* if

$$P^*_{X,\text{avg}} = \underset{P_X \in S_{X,T}}{\text{argmax}}\, F_{\text{avg}}(P_X;\theta) \tag{7}$$

It turns out that the problem of finding the optimal *average-case* segmentation does not have optimal substructure, unlike *worst-case* segmentation, meaning a locally optimal partitioning might no longer be optimal when considering global partitioning. Formally speaking, Lemma 1 no longer holds once $\min$ is changed to $\text{avg}$.

Despite without optimal substructure, we apply a similar greedy searching algorithm. The main difference is how we aggregate local scores. Though greedily averaging local scores might lead to myopic decisions in certain situations (Figure 3), it performs well in practice (Section IV).

### C. Learning the Objectness Function

We have discussed segmentation algorithms under the assumption that we already have access to an objectness function $f(C;\theta)$, which predicts an objectness score for a given point cloud. We now introduce how to learn this function. Despite there has been a line of work that focuses on learning better representation, including Kd-networks [30], PointCNN [31], EdgeConv [32], PointConv [33], just to name a few, we choose a simple PointNet++ to parameterize such an objectness function as a proof of concept. Below, we talk about how to learn a PointNet++ model as a regressor to predict objectness score.

*Ground truth objectness:* First, we must define regression target, i.e. ground truth objectness, of a given segment $C$. Suppose we have ground truth segmentation $P^{gt} = \{C^{gt}_1,\ldots,C^{gt}_L\}$, where $L$ is the number of ground truth segments. We can define $C$'s target objectness as the largest point IoU between itself and any ground truth segment Eq. (8).

$$Objectness(C,P^{gt}) = \max_{l=1,\ldots,L} \frac{|C \cap C^{gt}_l|}{|C \cup C^{gt}_l|} \tag{8}$$

Such a definition of objectness is only reasonable if points are uniformly distributed in space. In practice, 3D sensors (e.g. LiDAR) tend to produce denser points near the sensor. In consequence, the objectness will be heavily influenced by the partitioning of points closer to the sensor. For example, imagine two objects are segmented into one segment. Suppose one object has $n_1$ points and the other has $n_2$. If we use vanilla IoU as objectness, this segment would score $\frac{\max(n_1,n_2)}{n_1+n_2}$. When $n_1 \gg n_2$, the score could be really close to 1 despite it clearly introduces an under-segmentation error. To compensate such bias towards nearby objects, we propose a simple modification to IoU as in Eq. (9).

$$Objectness(C,P^{gt}) = \max_{l=1,\ldots,L} \frac{\sum_{x\in C\cap C^{gt}_l} x^T x}{\sum_{x\in C\cup C^{gt}_l} x^T x} \tag{9}$$

where $x^T x$ represents a point $x$'s squared distance to sensor origin. Eq. (8) is a special case, where $x^T x$ is replaced with 1.

*Implementation:* We train a PointNet++ w/ multi-scale grouping (MSG) [4] for learning the objectness function. Starting from the off-the-shelf architecture, we replaced the classifier with a regressor that produces a real-value given an input point cloud. We applied a sigmoid function to convert the regression output to numbers between [0, 1]. Finally, we compute the mean-squared error between prediction and ground truth objectness and perform backprop. In terms of preprocessing, we follow [24] to make sure the input cloud is centered at origin and rotated based on the viewpoint. To facilitate batch processing, we follow the standard practice for PointNet++ and re-sample each segment to 1024 points.

### D. Building Tree Hierarchies

We have discussed segmentation algorithms under the assumption that we have access to a tree hierarchy. Now we introduce how to build such a tree hierarchy given a set of points $X$. One natural approach is agglomerative clustering. After we define a metric (i.e. pairwise distance between two points) and a linkage criteria (i.e. pairwise distance between two sets of points), we can start from $\{\{x_1\},\ldots,\{x_N\}\}$ and keep merging the closest pair of point sets by taking the union over them, until all points are merged into one set. Such an approach produces a tree in a bottom-up fashion.

This approach tends to create tree hierarchies with very fine granularity, e.g. one node may differ from another with only one point of difference. As we have mentioned, our segmentation algorithms need to evaluate the objectness of every node in the tree. From an efficiency point of view, we would like to build a coarser tree whose leaf nodes are segments rather than individual points. Moreover, adjacent nodes should differ from each other much more.

*Implementation:* We build tree hierarchies by applying Euclidean Clustering [9] recursively in a top-down fashion

with a list of decreasing $\epsilon$. Since Euclidean Clustering finds connected components w.r.t. a distance threshold $\epsilon$, we start with the largest $\epsilon$ that defines the most coarse connected components. Then, we apply Euclidean Clustering with a smaller $\epsilon$ within each connected component. This produces a multiple-tree top-down hierarchy. In our experiments, we use $\epsilon \in \{2\text{ m}, 1\text{ m}, 0.5\text{ m}, 0.25\text{ m}\}$ to build tree hierarchies for both training and testing. During training, we extract segments out of tree hierarchies built with the same parameters to form our training set for learning the objectness function. During testing, we apply the same learned objectness function in both worst-case semgentation and average-case segmentation.

## IV. Experiments

For evaluation, we repurpose the KITTI object detection benchmark for point cloud segmentation following the setup in [8]. In our case, 3D objects do not physically overlap with one another. Therefore, we use ground truth 3D bounding boxes to produce ground truth segmentation. To do so, we first remove all points outside ground truth 3D bounding boxes (Figure 1). Then we treat points within one ground truth 3D bounding box as the ground truth segment for the object. On KITTI, there exist ground truth 3D bounding boxes that overlap with each other. We ignore such segments during evaluation, since it is not clear how to define the ground-truth for the points in such bounding boxes [8]. We follow [34] for splitting data into training and validation.

*Evaluation protocol:* We follow evaluation metrics introduced by Held *et al.* [8], which consists of two errors, under-segmentation error and over-segmentation error. Given ground truth segmentation $P^{gt} = \{C_1^{gt}, \ldots, C_L^{gt}\}$, we compute under-segmentation error $U$ and over-segmentation error $O$ given an output segmentation $P = \{C_1, \ldots, C_M\}$ as:

$$U = \frac{1}{L}\sum_{l=1}^{L} \mathbf{1}\left(\frac{|C_{i^*} \cap C_l^{gt}|}{|C_{i^*}|} < \tau_U\right) \tag{10}$$

$$O = \frac{1}{L}\sum_{l=1}^{L} \mathbf{1}\left(\frac{|C_{i^*} \cap C_l^{gt}|}{|C_l^{gt}|} < \tau_O\right) \tag{11}$$

with

$$i^* = \text{argmax}_{i=1}^{M} |C_i \cap C_l^{gt}| \tag{12}$$

where $\mathbf{1}(\cdot)$ is an indicator function and $\tau_U, \tau_O$ are both constant thresholds. We set $\tau_U = 2/3$ and $\tau_O = 1$ following [8]. We ignore objects with 0 points inside their 3D boxes (about 1%). For objects with overlapping bounding boxes (about 2.5%), we ignore points that fall into the overlapped region. Other than these, we compute segmentation errors over objects at all distance and also errors that focus on nearby objects (15 m).

### A. Baselines

*Euclidean clustering:* We use Euclidean clustering with 4 different distance threshold $\{2\text{ m}, 1\text{ m}, 0.5\text{ m}, 0.25\text{ m}\}$ to build trees of segments, which defines the space of possible segmentations for our approach. Therefore, we include them as baselines and see if a better solution can be found.

*State-of-the-art 3D detectors:* We compare our approach to AVOD [18], PointPillars [21], PointRCNN [25], and SECOND [22]. We follow the off-the-shelf training and testing setting as closely as possible. For AVOD, we re-train a car detector and a people detector (pedestrian and cyclist) with LiDAR as the only input following the official implementation.[2] For PointPillars, we re-train a detector that simultaneously detects cars and people (pedestrian and cyclist) following an author-endorsed implementation.[3] For PointRCNN, we evaluate the official pre-trained car model as there are no available models or training configurations for other classes within its official repository.[4] For SECOND, since it is our best performing baseline, besides re-training the off-the-shelf model, we also explore various ways to improve its performance.

By design, these detectors output class-specific bounding box detection. To produce class-agnostic segmentations, we ignore the class label and follow a greedy procedure: We start with the highest scoring bounding box and group all points within the box as one segment. We then remove those points and move onto the next highest scoring detection. We repeat until exhausting either detections or 3D points. In the end, we might still not have every point assigned to a segment. A simple fix is grouping leftover points as a new segment. We discuss a much better alternative approach below.

*Detector++:* A better approach to handling missed detection is to fall back to clustering. Specifically, we apply Euclidean Clustering (EC) with a fixed $\epsilon$ on all leftover points, producing a set of leftover segments. For each leftover segment, we check if it can merged into an existing detection segment, using the criteria of whether the smallest pairwise distance between two segments is smaller than the threshold $\epsilon$. If so, we merge the leftover segment into the detection segment. We refer to such baselines as Detector++ (e.g. AVOD++ etc.).

*SECOND++:* To ensure an apples-to-apples comparison, we re-train and re-evaluate the best baseline, i.e. SECOND, with background removal. These baselines are marked with "+ BG Removal". In addition, we discover that, by extending SECOND's detection range from 50 m to 80 m, we significantly improve SECOND's performance. The affected baselines are marked with "+ Ext. Range". Finally, we re-train and re-evaluate SECOND on all 8 classes. The new baselines are labeled as "SECOND++(8)". In contrast, off-the-shelf SECOND baselines are labeled as "SECOND++(4)" as they are trained on 4 classes (car, pedestrian, cyclist, and van).

### B. Results

We first present qualitative examples of our approach segmenting rare objects on KITTI Val, as shown in Figure 4. For quantitative evaluation, we present both per-class and overall segmentation errors in Table I.

*Ours(min) vs. Ours(avg):* We label the optimal worst-case segmentation as Ours(min) and the average-case segmentation as Ours(avg). Ours(avg) consistently outperforms Ours(min)

[2][Online]. Available: https://github.com/kujason/avod
[3][Online]. Available: https://github.com/traveller59/second.pytorch
[4][Online]. Available: https://github.com/sshaoshuai/pointrcnn

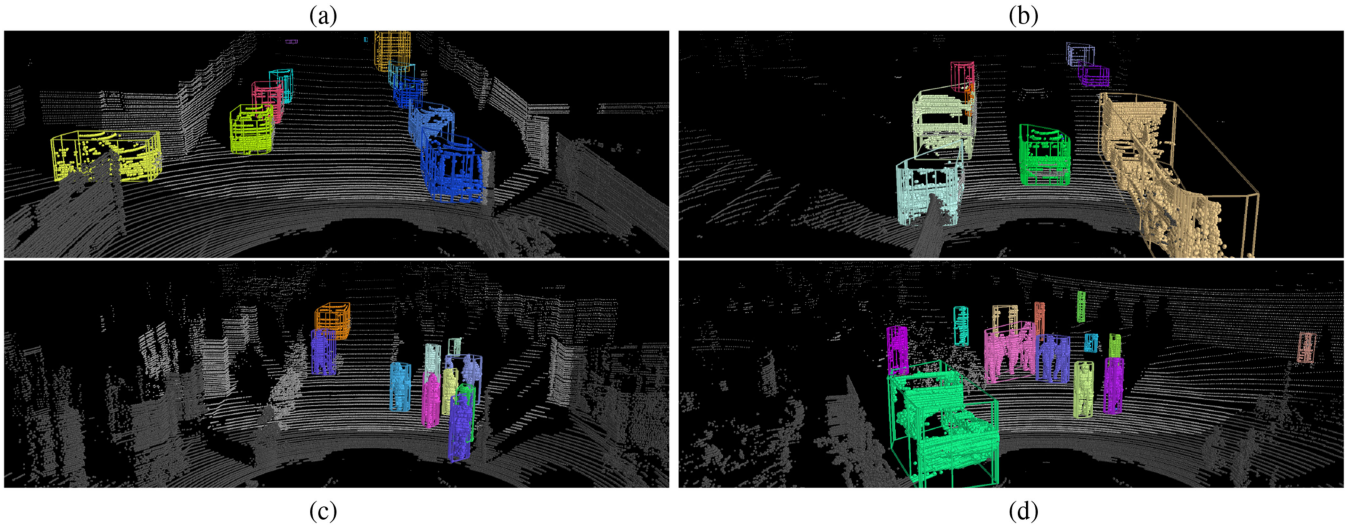



Fig. 4. We visualize more qualitative results of the proposed algorithm Ours(avg) on KITTI. In (a), we show a common scenario where there are parked cars on both sides of the road. In (b), we show a rare scenario where there is an oversized tank truck in the right lane. In (c), we show a scenario where a group of pedestrians walking in front of the autonomous vehicle. In (d), we show a typical failure case where pedestrians walk closely side by side. For such cases, there is often no perfect solution within the search space generated by EC.

TABLE I
SEGMENTATION ERRORS ON KITTI VAL. LEFT SHOWS UNDER-, OVER-SEGMENTATION, AND TOTAL ERROR. RIGHT SHOWS TOTAL ERROR ON A PER-CLASS BASIS

| Method | under | | over | | **total** | | car | | van | | truck | | pedestrian | | person sitting | | cyclist | | *tram* | | misc | | **mean** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | all | 15m | all | 15m | all | 15m | all | 15m | all | 15m | all | 15m | all | 15m | all | 15m | all | 15m | all | 15m | all | 15m | all | 15m |
| EC(2m) | 24.89 | 46.21 | 5.24 | 0.43 | 30.1 | 46.6 | 24.4 | 37.3 | 18.2 | 21.1 | **29.3** | **18.5** | 63.1 | 75.2 | 79.2 | 78.5 | 28.7 | 53.0 | 94.9 | 55.0 | 31.0 | 36.2 | 46.1 | 46.9 |
| EC(1m) | 11.26 | 26.31 | 24.89 | 7.09 | 36.1 | 33.4 | 31.4 | 21.2 | 44.5 | 22.1 | 51.6 | 59.3 | 48.9 | 60.7 | 74.0 | 74.3 | 17.4 | 35.6 | 121.2 | 100.0 | 39.5 | 36.2 | 53.6 | 51.2 |
| EC(0.5m) | 5.54 | 12.89 | 63.70 | 48.32 | 69.2 | 61.2 | 74.6 | 67.9 | 81.0 | 73.2 | 79.3 | 92.6 | 39.0 | 47.6 | 63.0 | 62.5 | 25.5 | 16.1 | 121.2 | 100.0 | 64.9 | 43.3 | 68.6 | 62.9 |
| EC(0.25m) | 3.02 | 7.47 | 89.61 | 78.70 | 92.6 | 86.2 | 97.1 | 98.8 | 98.8 | 99.1 | 98.7 | 100.0 | 58.9 | 54.1 | 74.7 | 75.0 | 87.5 | 56.4 | 119.1 | 100.0 | 94.1 | 82.7 | 91.1 | 83.3 |
| EC(all)* | *9.72* | *17.16* | *5.24* | *0.43* | *15.0* | *17.6* | *10.9* | *11.6* | *13.6* | *5.6* | *29.3* | *18.5* | *26.9* | *33.6* | *48.7* | *48.6* | *10.4* | *14.1* | *94.5* | *55.0* | *15.8* | *7.1* | *31.3* | *24.3* |
| AVOD | - | - | - | - | - | - | 81.8 | 85.5 | - | - | - | - | 85.4 | 92.0 | - | - | 88.3 | 87.9 | - | - | - | - | - | - |
| AVOD++ | - | - | - | - | - | - | 12.5 | 10.7 | - | - | - | - | 36.6 | 46.4 | - | - | 13.1 | 18.8 | - | - | - | - | - | - |
| PointPillars++ | - | - | - | - | - | - | 22.4 | 22.7 | - | - | - | - | 58.6 | 63.6 | - | - | 44.8 | 36.2 | - | - | - | - | - | - |
| PointRCNN++ | - | - | - | - | - | - | 7.6 | 5.2 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| SECOND++(4) | - | - | - | - | - | - | 8.1 | 4.2 | 22.5 | 7.5 | - | - | 34.4 | 41.6 | - | - | 10.6 | 14.1 | - | - | - | - | - | - |
| + Ext. Range | - | - | - | - | - | - | **7.2** | 4.5 | 18.4 | 7.0 | - | - | 34.2 | 41.8 | - | - | 10.0 | **10.1** | - | - | - | - | - | - |
| + BG Removal | - | - | - | - | - | - | 9.9 | **4.0** | 25.9 | 8.9 | - | - | 33.2 | 39.1 | - | - | 11.9 | 15.4 | - | - | - | - | - | - |
| + Both | - | - | - | - | - | - | 9.3 | **4.0** | 20.6 | 8.0 | - | - | 32.9 | **38.7** | - | - | 12.0 | 14.1 | - | - | - | - | - | - |
| SECOND++(8) | 4.07 | 7.42 | 11.79 | 9.25 | 15.9 | 16.7 | 8.1 | 5.1 | 23.2 | 8.0 | 43.2 | 51.9 | 33.2 | 39.5 | 70.8 | 70.8 | 9.6 | 13.4 | 119.5 | 100.0 | 28.7 | 18.9 | 42.0 | 38.5 |
| + Ext. Range | 4.28 | 7.85 | 10.48 | 9.10 | **14.8** | 17.0 | 7.4 | 4.9 | 17.6 | 8.5 | 34.0 | 48.1 | 34.3 | 40.8 | 72.7 | 72.9 | **9.5** | 12.1 | 115.3 | 100.0 | 27.8 | 22.8 | 39.8 | 38.8 |
| + BG Removal | 3.89 | 6.74 | 13.34 | 9.74 | 17.2 | 16.5 | 10.0 | 5.0 | 24.7 | 5.6 | 45.8 | 63.0 | **32.8** | 40.0 | 63.0 | 62.5 | 11.7 | 14.8 | 116.1 | 95.0 | 26.6 | 21.3 | 41.3 | 38.4 |
| + Both | 3.84 | 6.66 | 12.28 | 9.66 | 16.1 | **16.3** | 9.4 | 4.8 | 20.1 | 6.1 | 38.2 | 63.0 | 33.2 | 40.1 | 63.0 | 62.5 | 11.5 | 13.4 | 112.7 | 95.0 | 23.4 | 21.3 | 38.9 | 38.3 |
| Ours(min) | 15.09 | 25.70 | 5.57 | 0.58 | 20.7 | 26.3 | 15.9 | 17.9 | 15.4 | 11.3 | **29.3** | **18.5** | 39.2 | 48.0 | 68.8 | 67.4 | 17.9 | 26.2 | 94.9 | 55.0 | 23.2 | **15.0** | 38.1 | 32.4 |
| Ours(avg) | 10.54 | 17.41 | 7.87 | 4.60 | 18.4 | 22.0 | 13.8 | 14.6 | 14.8 | 7.0 | 30.1 | 29.6 | 33.4 | 40.6 | 61.7 | 61.1 | 16.4 | 19.5 | 94.9 | 55.0 | 22.3 | **15.0** | 35.9 | 30.3 |
| Ours(avg) w/ | | | | | | | | | | | | | | | | | | | | | | | | |
| (2.7, 0.9, 0.3)m | 13.41 | 19.65 | 6.16 | 5.13 | 19.6 | 24.8 | 15.8 | 17.1 | 13.9 | 13.6 | 23.6 | 25.9 | 36.2 | 43.3 | 63.6 | 63.2 | 18.3 | 26.2 | 72.0 | 20.0 | 20.2 | 19.7 | 33.0 | 28.6 |
| (2.4, 1.2, 0.6, 0.3)m | 11.26 | 16.65 | 5.94 | 5.69 | 17.2 | 22.3 | 12.9 | 15.0 | 11.5 | 7.5 | 27.2 | 29.6 | 34.7 | 41.2 | **59.7** | **59.0** | 15.7 | 19.5 | 80.9 | 25.0 | **18.4** | 19.7 | 32.6 | 27.1 |
| (3.2, 1.6, 0.8, 0.4, 0.2)m | 12.36 | 15.10 | 4.67 | 5.36 | 17.0 | 20.5 | 12.8 | 12.7 | **10.7** | **4.2** | **21.5** | 22.2 | 35.8 | 41.2 | **59.7** | **59.0** | 17.4 | 16.8 | **67.8** | **10.0** | 19.0 | 17.3 | **30.6** | **22.9** |

in terms of the total error. Ours(min) produces a much lower over-segmentation error but a much higher under-segmentation error, suggesting it makes more mistakes of grouping different objects into one segment and less mistakes of splitting points from one single object into multiple segments. The cause of such behavior might be due to the risk-averse objective of optimal worst-case segmentation. However, current evaluation does not emphasize the worst-case performance, instead, it measures the average performance over all objects. We observe that if we evaluate the worst-case objectness (Section IV-C), Ours(min) does outperform both Ours(avg) and AVOD++.

*Ours vs. Euclidean Clustering:* We label Euclidean Clustering as "EC($\epsilon$)," where $\epsilon$ represents the distance threshold (meter). All together, they define a segment hierarchy. We construct a pool of segments that contains every node (segment) in the hierarchy and call this "EC(all)*". This serves as a *unreachable* upper-bound, since segments from such a pool overlap with each other, which violates the non-disjoint constraint of a valid partition. Nonetheless, it shows that there gap between our proposed method and the upper bound is relatively small (3–4%), suggesting plenty of room left for improvement in creating better hierarchies.

*Detector++ vs. Detector:* We focus on AVOD to demonstrate the improvement of Detector++ over Detector. AVOD produces much larger oversegmentation errors, likely due to imprecisely localized 3D bounding boxes. For example, when a 3D bounding box is predicted smaller than it should be, the resultant segment might miss points on the edge, leading to oversegmentation. AVOD++ is designed to fix this issue and dramatically improves the oversegmentation error. The undersegmentation errors also improves significantly from AVOD to AVOD++, likely due to successfully segmenting objects that are completely missed by detections.

TABLE II
INSTANCE SEGMENTATION AP[@.5:.95:.05] ON KITTI VAL

| | car | van | trk | ped | psit | cyc | tram | misc | **mean** |
|---|---|---|---|---|---|---|---|---|---|
| AVOD | 64.4 | - | - | 31.1 | - | 15.5 | - | - | - |
| AVOD++ | 91.6 | - | - | 51.6 | - | 41.6 | - | - | - |
| PointPillars++ | 91.4 | - | - | 55.2 | - | 55.9 | - | - | - |
| PointRCNN++ | 95.2 | - | - | - | - | - | - | - | - |
| SECOND++(4) | 95.1 | 68.9 | - | 68.6 | - | 65.9 | - | - | - |
| + Ext. Range | 95.8 | 75.4 | - | 70.2 | - | 68.1 | - | - | - |
| + BG Removal | 95.3 | 74.0 | - | 77.5 | - | 71.8 | - | - | - |
| + Both | **96.0** | **82.0** | - | **78.1** | - | **72.9** | - | - | - |
| SECOND++(8) | 95.3 | 70.3 | 30.0 | 71.8 | 2.6 | 69.6 | 10.2 | 33.9 | 48.0 |
| + Ext. Range | 95.9 | 78.3 | **63.4** | 71.0 | 2.9 | 71.9 | 13.4 | 39.6 | 54.5 |
| + BG Removal | 95.1 | 73.5 | 30.3 | 76.2 | 9.0 | 71.4 | 13.1 | 47.3 | 52.0 |
| + Both | **96.0** | 81.3 | 61.7 | 76.5 | 8.6 | 72.4 | 16.4 | **55.4** | **58.5** |
| Ours(min) | 86.0 | 80.4 | 61.6 | 62.3 | 12.9 | 66.3 | **21.9** | 53.0 | 55.6 |
| Ours(avg) | 89.8 | 81.1 | 58.6 | 69.2 | **14.0** | 68.2 | 19.8 | 51.0 | 56.5 |
| Ours(avg) w/ | | | | | | | | | |
| (2.7, 0.9, 0.3)m | 87.5 | 78.6 | 57.6 | 66.7 | **14.0** | 66.9 | 20.8 | 49.7 | 55.2 |
| (2.4, 1.2, 0.6, 0.3)m | 89.6 | 81.9 | 59.4 | 67.9 | 13.7 | 69.2 | 21.5 | 52.1 | 56.9 |
| (3.2, 1.6, 0.8, 0.4, 0.2)m | 89.0 | 79.0 | 56.3 | 67.6 | 13.2 | 67.2 | 18.7 | 49.0 | 55.0 |

*Ours vs. Detector++:* SECOND++ performs the best among all Detector++ baselines and also achieves the lowest overall total error among all methods. However, if we break down total segmentation errors on a per-class basis, our approaches perform much better than SECOND++. Such difference is due to a skewed data distribution. For example, 68% objects are labeled as car while only 3% are labeled as misc. SECOND++ performs better on common classes such as car and ours perform better on rare ones such as misc.

*Runtime analysis:* Our algorithm requires running PointNet++ on every candidate segment in order to compute its objectness. In practice, one frame from KITTI Val, which contains $68(\sigma = 42)$ segments on average, takes about $0.19\ s(\sigma = 0.06\ s)$ to process on a single GTX 1080.

### C. Additional Evaluation Protocols

*Class-agnostic instance segmentation:* The evaluation protocol we adopt comes from the robotics community [15]. It differs from the standard evaluation in computer vision, i.e. per-voxel instance segmentation in ScanNet [35]. One key difference is that 3D instance segmentation does not require the output segmentation to be a valid partition. Instead, it treats the task as retrieval and evaluates the tradeoff between precision and recall. Here we take a similar approach as ScanNet, but modify the evaluation protocol to be class-agnostic and per-point instead of per-voxel.

As we can see in Table II, the observations are consistent with what we see in Table I: SECOND++(8) with both modifications outperforms our segmentation approach on common classes such as *car*, but falls short on rarer classes (such as person sitting and tram) by a large margin. Overall, the best SECOND approach outperforms the best variant of our approach by 1.6% in mAP.

*How objectness generalizes:* To evaluate how well our learned objectness model generalizes, we apply it onto ground truth segments from the validation set. In Figure 5, we plot the average objectness score for each class and the standard deviation. We also show the percentage of objects for each class within the training set. As the number of training data decreases

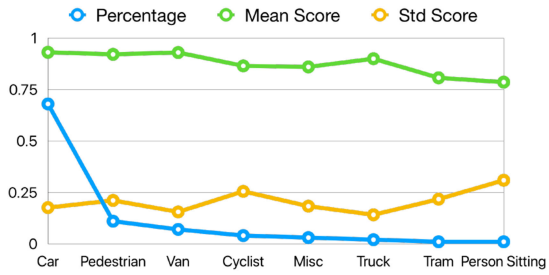



Fig. 5. How the learned objectness model generalizes in the tail.

TABLE III
SEGMENTATION ERRORS ON KITTI VAL

| Method | Under (%) | | Over (%) | | Total (%) | |
|---|---|---|---|---|---|---|
| | all | 15m | all | 15m | all | 15m |
| Ours(min) - vanilla | 13.91 | 22.40 | 5.58 | 0.60 | 19.5 | 23.0 |
| Ours(min) - weighted | 13.13 | 21.42 | 5.65 | 0.60 | **18.8** | **22.0** |
| Ours(avg) - vanilla | 10.30 | 15.44 | 7.11 | 3.13 | 17.4 | 18.6 |
| Ours(avg) - weighted | 8.64 | 12.75 | 7.89 | 4.73 | **16.5** | **17.5** |

dramatically, the average score tends to drops slightly and the variance tends to rise slightly.

*Worst-case evaluation:* In Tables I and II, we see Ours(avg) outperforms Ours(min) despite the latter is provably optimal. We have briefly discussed the reason: current protocols do not evaluate worst-case performance. Here, we score the worst IoU between a set of local segments and the ground truths, as Eq. (13) shows, where $\{P_1 \dots P_N\}$ and $\{P_1^{gt} \dots P_N^{gt}\}$ represents predicted and ground truth segmentation in each of the $N$ frames. We found Ours(min) scores a mean-worst IoU of 72.2%, 4.2% higher than Ours(avg).

$$\text{score} = \sum_{i=1}^{N} \frac{1}{N} \min_{C \in P_i} \max_{C^{gt} \in P_i^{gt}} \frac{|C \cap C^{gt}|}{|C \cup C^{gt}|} \tag{13}$$

### D. Additional Diagnostics

*Sensitivity analysis:* Our objectness function is learned on segments from a EC hierarchy generated with 4 distance thresholds {2 m, 1 m, 0.5 m, 0.25m}. To analyze how robust our algorithm is to change of hyper-parameters, we test the learned objectness function on different hierarchies. In Tables I and II, we find that having a deeper hierarchy significantly reduces segmentation errors. Comparing to hard-thresholded segmentation errors, there are only slight changes in multi-threshold instance segmentation mAP.

*Weighted vs. vanilla IoU:* Here, we empirically compare weighted IoU and vanilla IoU in terms of defining the training target for our objectness model. As we see in Table III, for both worst-case and average-case segmentation, the objectness model trained with weighted IoU perform slightly better than the one trained with vanilla IoU. Note "Ours(min) - vanilla" and "Ours(avg) - vanilla" share the exact same underlying objectness model.

## V. Conclusion

We present an approach for class-agnostic point cloud segmentation. The approach efficiently searches over an exponentially large space of candidate segmentations and return one where individual segments score well according to a data-driven point-based model of "objectness". We prove that our algorithm is guaranteed to achieve optimality to a specific definition. On KITTI, we demonstrate our approach significantly outperforms past bottom-up approaches and top-down object-based algorithms for segmenting point clouds.

## Appendix

There is a video (https://www.cs.cmu.edu/~peiyunh/seg/slides.mp4link) that illustrates the main ideas of this letter. In addition, there are more videos (links: https://www.cs.cmu.edu/~peiyunh/seg/kitti/2011_09_26_drive_0009.mp41, https://www.cs.cmu.edu/~peiyunh/seg/kitti/2011_09_26_drive_0035.mp42, https://www.cs.cmu.edu/~peiyunh/seg/kitti/2011_09_28_drive_0021.mp43) that highlight the advantages and limitations of our approach.

## References


[1] C. Urmson *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *J. Field Robot.*, vol. 25, no. 8, pp. 425–466, 2008.

[2] M. Montemerlo *et al.*, "Junior: The stanford entry in the urban challenge," *J. Field Robot.*, vol. 25, no. 9, pp. 569–597, 2008.

[3] B. Alexe, T. Deselaers, and V. Ferrari, "What is an object?" in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2010, pp. 73–80.

[4] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5099–5108.

[5] Q.-Y. Zhou and U. Neumann, "A streaming framework for seamless building reconstruction from large-scale aerial lidar data," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2009, pp. 2759–2766.

[6] L. McInnes, J. Healy, and S. Astels, "hdbscan: Hierarchical density based clustering," *J. Open Source Softw.*, vol. 2, no. 11, 2017, Art. no. 205.

[7] J. Strom, A. Richardson, and E. Olson, "Graph-based segmentation for colored 3D laser point clouds," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2010, pp. 2131–2136.

[8] D. Held, D. Guillory, B. Rebsamen, S. Thrun, and S. Savarese, "A probabilistic framework for real-time 3D segmentation using spatial, temporal, and semantic cues," *Robotics: Science and Systems*, 2016.

[9] R. B. Rusu, "Semantic 3D object maps for everyday manipulation in human living environments," *KI-Künstliche Intelligenz*, vol. 24, no. 4, pp. 345–348, 2010.

[10] T. Hackel, J. D. Wegner, and K. Schindler, "Fast semantic segmentation of 3D point clouds with strongly varying density," *ISPRS Ann. Photogrammetry, Remote Sens. Spatial Inf. Sci.*, vol. 3, no. 3, pp. 177–184, 2016.

[11] X. Weng and K. Kitani, "A baseline for 3D multi-object tracking," 2019, *arXiv: 1907.03961*.

[12] J. Ku, A. D. Pon, and S. L. Waslander, "Monocular 3D object detection leveraging accurate proposals and shape reconstruction," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2019, pp. 11 867–11 876.

[13] B. Douillard *et al.*, "On the segmentation of 3D lidar point clouds," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 2798–2805.

[14] K. Klasing, D. Wollherr, and M. Buss, "A clustering method for efficient segmentation of 3D laser data," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2008, pp. 4043–4048.

[15] D. Held, J. Levinson, S. Thrun, and S. Savarese, "Combining 3D shape, color, and motion for robust anytime tracking," *Robotics: Science and Systems*, 2014.

[16] A. Teichman, J. Levinson, and S. Thrun, "Towards 3D object recognition via classification of arbitrary object tracks," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 4034–4041.

[17] A. Azim and O. Aycard, "Detection, classification and tracking of moving objects in a 3D environment," in *Proc. IEEE Intell. Veh. Symp.*, 2012, pp. 802–807.

[18] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander, "Joint 3D proposal generation and object detection from view aggregation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 1–8.

[19] W. Luo, B. Yang, and R. Urtasun, "Fast and furious: Real time end-to-end 3D detection, tracking and motion forecasting with a single convolutional net," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 3569–3577.

[20] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3D object detection," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 4490–4499.

[21] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2019, pp. 12697–12705.

[22] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, 2018, Art. no. 3337.

[23] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, Aug. 2000.

[24] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3D object detection from RGB-D data," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 918–927.

[25] S. Shi, X. Wang, and H. Li, "Pointrcnn: 3D object proposal generation and detection from point cloud," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2019, pp. 770–779.

[26] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," in *Proc. IEEE Trans. Pattern Anal. Mach. Intell.*, 1999, vol. 1, pp. 377–384.

[27] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *Int. J. Comput. Vision*, vol. 59, no. 2, pp. 167–181, 2004.

[28] M. G. Uzunbas, C. Chen, and D. Metaxas, "An efficient conditional random field approach for automatic and interactive neuron segmentation," *Med. Image Anal.*, vol. 27, pp. 31–44, Jan. 2016.

[29] A. Ošep, W. Mehner, P. Voigtlaender, and B. Leibe, "Track, then decide: Category-agnostic vision-based multi-object tracking," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 1–8.

[30] R. Klokov and V. Lempitsky, "Escape from cells: Deep kd-networks for the recognition of 3D point cloud models," in *Proc. Trans. Pattern Anal. Mach. Intell.*, 2017, pp. 863–872.

[31] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "Pointcnn: Convolution on x-transformed points," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 820–830.

[32] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ACM TOG*, vol. 38, no. 5, 2019, Art. no. 146.

[33] W. Wu, Z. Qi, and L. Fuxin, "Pointconv: Deep convolutional networks on 3D point clouds," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2019, pp. 9621–9630.

[34] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3D object detection network for autonomous driving," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2017, pp. 1907–1915.

[35] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3D reconstructions of indoor scenes," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2017, pp. 5828–5839.