Computer-Vision-based 3D Object Detection for Self Driving Cars

Manoj Rajagopalan
Stanford University
rmanoj@stanford.edu

Martin Freeman
Stanford University
emmerich@stanford.edu

Shoaib Lari
Stanford University
slari1@stanford.edu

Abstract

We present a method to detect and locate dynamic entities for autonomous vehicles using only stereo- and mono- computer vision techniques. Our method is a composition of geometric methods and deep learning. We attempt to compare our method to conventional approaches that employ LIDAR-based-detection. We show that our method produces reasonable object bounding boxes on entities of interest. Our measurement of runtimes on cloud platforms shows that our method is a promising start to extracting dynamic-entity ground-truth at scale.

1. Introduction

Although self driving cars have come a long way as a result of advancement of technologies, perception (identification and classification of objects around the vehicle) is still a key limiter: the vehicle gets a very small time-quantum within which to sense the environment, fuse different modalities and present a comprehension (classification and tracks) to the Planner. Safety is paramount and Evaluation at-scale is a huge challenge for the industry. Typical approaches rely heavily on hand-labeling of object labels and bounding boxes. These approaches outsource "ground-truth" perception to humans and therefore are very expensive while being imperfect. An alternative would be to analyze, offline, the full recording of all raw sensor data for the entire mission and to comprehend object labels, bounding boxes and tracks as ground truth to evaluate in-vehicle-perception against. Not only are vast amounts of time, compute- and storage-resources available to crunch the enormous amount of data (esp. in cloud environments), but such approaches also have knowledge of "the future" to be able to reason more accurately about the vehicle environment at any time.

In this project, we experimented with Computer Vision techniques for performance of offline 3D detection on industry-standard datasets. As a first step in this experimentation, we reproduced the detection and tracking results in [2], and explored the MEGVII[1] [12] detector utilized in the paper.

Subsequently, we explored generation of 3D object bounding boxes (OBB) using only stereo image-pairs and mono-/stereo- computer vision techniques, instead of using LIDAR points. LIDAR points are sparse and accurate. Stereo points are dense and less accurate than LIDAR.

2. Related Work

Most 3D object detection strategies for self driving cars can be generally divided into those that use Lidar and those that use Lidar + Image (RGB).

The Lidar-based 3D detection methods can be further classified into two categories in terms of point cloud representations, namely the grid-based methods and the point-based methods.

The grid-based methods generally transform the irregular point clouds to regular representations such as 3D voxels or 2D bird-view (BEV) maps, both of which are efficiently processed by 3D or 2D Convolutional Neural Networks (CNN) to learn point features for 3D detection. However, 3D convolution suffers from greater computational cost and higher latency than 2D convolutions. Examples of voxel-based methods are Voxelnet [14], Second [15], Fast point-rcnn [16], and Part-a2 net [17]. Examples of BEV implementations are AVOD [19], Pixor [20], HDNet [22], and PointPillars [23].

Point-based methods preserve the point cloud's spatial structure and directly extract discriminative features from raw point clouds for 3D detection. PointNet [26] generates spatial encodings for each point, which are then aggregated within grid cells via a symmetrical operation such as max(). The local point features are further aggregated into more global features while maintaining symmetry between points. Many grid-free implementations use the set abstraction (SA) layers introduced in PointNet++ [27]. Similar to convolution layers, SA layers can be stacked to store global features

[1] MEGVII, winner of the Lidar Track of the *NuScenes 3D Detection Challenge*, applies sparse 3D convolutions on input LIDAR point-clouds to extract features and generate detections.

into a few adaptively sampled keypoints, which can then be used as input to other SA layers. However, in general the SA layers are slower than a CNN backbone. The PointNet++ architecture infers global structure from local regions, performs multi-scale feature fusion, and propagates global information to local points. PointRCNN [28] utilizes two stages for 3D object detection. It uses PointNet++ to produce object bounding box proposals. Frustum ConvNet [29] uses sliding frustums to aggregate local pointwise features for 3D object detection. Sparse-To-Dense (STD) [30] improves latency by voxelizing the features for each of the proposals, yielding a more compact representation that is efficiently used by fully-connected (FC) layers in the second stage.

Generally, the grid-based methods are more computationally efficient, but the inevitable information loss degrades the fine-grained localization accuracy, while the point-based methods have higher computation cost, but can easily achieve larger receptive fields by the point set abstraction. With these tradeoffs, a common strategy is to use a hybrid approach by augmenting the grid-based backbone with a parallel branch. The current top performer on KITTI, PV-RCNN [32] is a two stage detector which uses a CNN to generate region proposals, while utilizing a parallel grid-free PointNet-based branch to aggregate features into keypoints. Because features are accumulated into these adaptively chosen key points, PV-RCNN preserves both point-cloud structure and localization information, thereby improving accuracy in the box refinement stage.

In comparison to Lidar-only methods, the Lidar + RGB fusion methods improve 3D detection performance, especially for smaller objects such as pedestrians or at long range objects located farther than 50 meters where Lidar data is often sparse. Proposal-based methods generate object proposals in RGB (e.g. F-Pointnet [25] which was a pioneering grid-free method) or BEV (e.g MV3D [18]). ContFuse [21] does BEV dense fusion. It generates BEV feature maps from RGB features, which are fused with the Lidar BEV features. Multi-task Multi-sensor Fusion (MMF) [24] uses Lidar points to fetch image features.

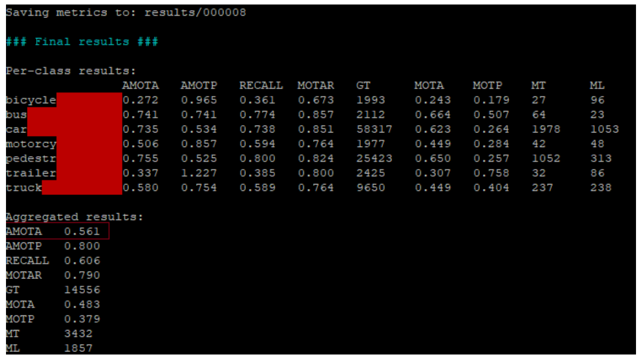
3. Approach

Our original plan was to use the Probabilistic 3D Multi-Modal, Multi-Object Tracking for Autonomous Driving paper [1] as the base implementation to experiment with. However, we were not able to access the code as there was still some work in progress. Therefore, after consulting with the teaching staff, we decided to base our efforts on an earlier paper Probabilistic 3d multi-object tracking for autonomous driving [2]. This paper is based on the nuScenes [3] dataset.

Our goal changed to reproducing this paper's results and extending it with our own approach to object detection using knowledge gained from CS231A. We also switched to using the popular KITTI [4] dataset.

3A. Reproduced Validation Set Results

The code for the paper *Probabilistic 3D Multi-Object Tracking for Autonomous Driving* [2] was made available to us. The work proposed a novel, online tracking method that took first place in the *NuScenes Tracking Challenge* at NeurIPS 2019. The code describes a full detection and tracking pipeline, with the detection module making use of MEGVII[12]'s detection results as input to the novel tracking method. Using the AMOTA (Average Multi-Object Tracking Accuracy) metric, we reproduced the validation set AMOTA results reported in [2] with the NuScenes dataset metadata (Fig. 1, 2).



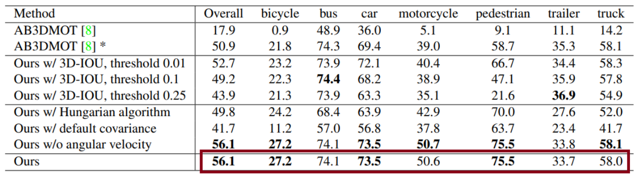
```
Saving metrics to: results/000008

### Final results ###

Per-class results:
                 AMOTA   AMOTP   RECALL  MOTAR   GT      MOTA    MOTP    MT      ML
bicycle[redacted]0.272   0.965   0.361   0.673   1993    0.243   0.179   27      96
bus              0.741   0.741   0.774   0.857   2112    0.664   0.507   64      23
car              0.735   0.534   0.738   0.851   58317   0.623   0.264   1978    1053
motorcy          0.506   0.857   0.594   0.764   1977    0.449   0.284   42      48
pedestr          0.755   0.525   0.800   0.824   25423   0.650   0.257   1052    313
trailer          0.337   1.227   0.385   0.800   2425    0.307   0.758   32      86
truck            0.580   0.754   0.589   0.764   9650    0.449   0.404   237     238

Aggregated results:
AMOTA    0.561
AMOTP    0.800
RECALL   0.606
MOTAR    0.790
GT       14556
MOTA     0.483
MOTP     0.379
MT       3432
ML       1857
```

Fig. 1: Validation Set Results Reproduction



Method	Overall	bicycle	bus	car	motorcycle	pedestrian	trailer	truck
AB3DMOT [8]	17.9	0.9	48.9	36.0	5.1	9.1	11.1	14.2
AB3DMOT [8] *	50.9	21.8	74.3	69.4	39.0	58.7	35.3	58.1
Ours w/ 3D-IOU, threshold 0.01	52.7	23.2	73.9	72.1	40.4	66.7	34.4	58.3
Ours w/ 3D-IOU, threshold 0.1	49.2	22.3	**74.4**	68.2	38.9	47.1	35.9	57.8
Ours w/ 3D-IOU, threshold 0.25	43.9	21.3	73.9	63.3	35.1	21.6	**36.9**	54.9
Ours w/ Hungarian algorithm	49.8	24.2	68.4	63.9	42.9	70.0	27.6	52.0
Ours w/ default covariance	41.7	11.2	57.0	56.8	37.8	63.7	23.4	41.7
Ours w/o angular velocity	**56.1**	**27.2**	74.1	**73.5**	**50.7**	**75.5**	33.8	**58.1**
Ours	**56.1**	**27.2**	74.1	**73.5**	50.6	**75.5**	33.7	58.0

Fig. 2: Original Validation Set Results

3B. Our Approach: Computer-Vision based Detection

Our approach to object detection is shown in Fig. 3. It is done per-frame for cars, motorcycles, bicycles, buses, trains and trucks. We first compute per-pixel disparities from rectified image-pairs using classical (SGBM) or learning-based (PSMNet) techniques. We then calculate per-pixel depth and use the inverse of the intrinsic calibration matrix to compute a "grid" of world-space 3D points for each (valid) pixel in the camera frame (point grid). We use the YOLOv5 classifier neural network to identify objects of interest, in the left image: we use the resulting bounding boxes to carve out object point-clouds from the scene point-grid. For each such object point-cloud (multiple per scene), we reject the ground-plane (based on height) and separate the foreground from the background using K-means clustering (K=2), and then run Principal Component Analysis (PCA) in order to establish a 3D object bounding box (OBB) for that object-instance. Finally, we reproject these 3D bounding boxes onto the left-camera image for visual evaluation.

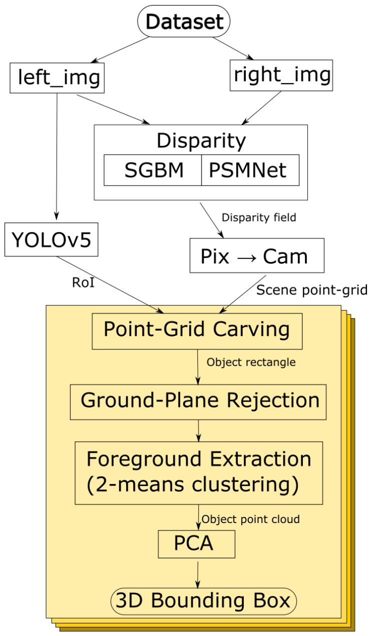


Fig. 3: Our object-detection pipeline. Yellow box shows steps performed per identified object.

We detail each step in the following subsections.

3C. Disparity Estimation

We first experimented with OpenCV's Stereo Global Block Matching (SGBM) with disparities in the range [4..192] and a block-size of 5. Larger block-sizes yielded sparser density for disparity values. The disparity range was chosen to match the defaults for PSMNet (next).

We also experimented with the Pyramidal Stereo Matching (neural) net (PSMNet) [10] for per-pixel disparities using pre-trained weights, using max-disparity of 192.

Fig. 5 and Fig. 6 show the results from these methods.

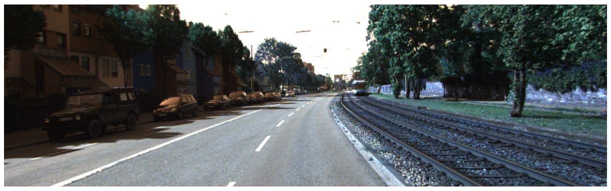


Fig. 4: Reference left-image for all results to follow

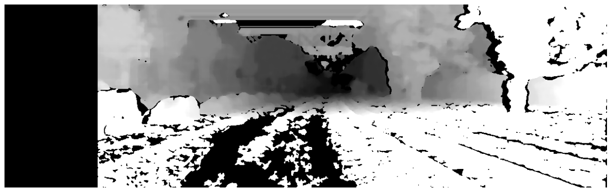


Fig. 5: OpenCV SGBM disparity field

Fig. 6: PSMNet disparity field (faint)

3D. Depth Field and Point-Grid Calculation

We computed the per-pixel depth field using the well known formula from epipolar geometry:

$$z_{ij} = b \times f / d_{ij}$$

where b is the stereo baseline, f is the focal length of the left camera, d_{ij} is the disparity at pixel (j,i).

We then calculated the 3D coordinates of each pixel in the camera frame using another well-known relation:

$$\mathbf{P}_{ij} = \mathbf{K}^{-1} \left(z_{ij} \, [j, i, 1]^{T} \right)$$

Fig. 7 shows a point grid calculated using PSMNet disparities.

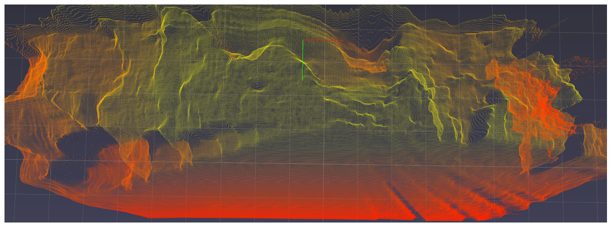


Fig. 7: Point-Grid for scene (car-line visible on left)

3E. Object Identification

We used the YOLOv5 [11] CNN detector, with pre-trained weights, to identify cars, trains, etc. Running it on color left images yields 2D axis-aligned bounding boxes and associated confidences for instances of object-classes of interest. We rejected identifications with confidences less than 60%. Fig. 8 shows detections on the input color image. Fig. 9 shows the result superimposed on the disparity field from SGBM.

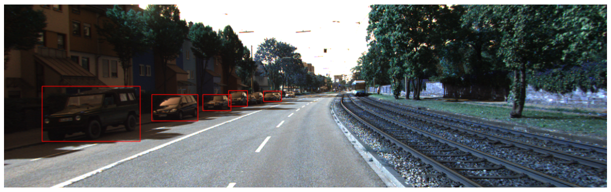


Fig. 8: YOLOv5 identification of cars (red bounding boxes)

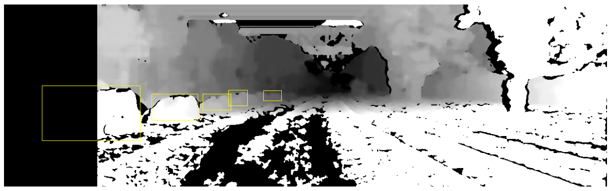


Fig. 9: YOLOv5 rects superimposed on disparity field

3F. Object Extraction and Bounding Box Calculation

We used YOLOv5 bounding boxes to carve out point clouds for individual objects in the scene (object point clouds).

To refine the object, we first rejected the ground-plane points by ignoring all whose *y*-coordinate > 1.6 m (KITTI cameras are at 1.65 m height from ground, and *y*-axis points towards gravity). Then we discarded background points after separating from the foreground using *K*-means clustering using *K*=2. We performed the clustering on an auxiliary dataset of scalar distances of each point from the camera, and used the resulting indices to separate the object point-cloud. Fig. 10 shows one such object after all these steps. We can see that it is well-isolated.

Finally, we ran Principal Component Analysis (PCA) to identify dominant directions for point-cloud distribution. By casting all points into the coordinate frame formed by the center and eigenvectors returned by PCA, we computed 3D bounding box orientation and extent.

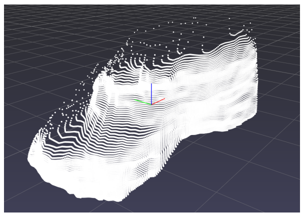


Fig. 10: Point cloud for the left-most car

3G. RANSAC: Attempt at Shape Extraction

We attempted to run RANSAC to extract the plane surfaces bounding car-volumes with the hope that within each plane we could fit polygons to the convex hull and extract a wireframe mesh for the car. But the object-point-cloud is so noisy that it is only possible to fit a few planes and many converge to oblique, transverse cuts across the object as shown in Fig. 10 regardless of the tolerances we chose. Hence, we abandoned this step.

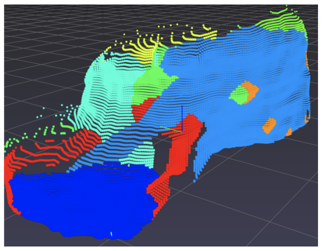


Fig. 10: RANSAC attempt: one color per plane

3H. YOLO vs. Point-cloud Approaches

Another angle of this project we wanted to explore was how our stereo-based detector approach would fare against popular LIDAR point-cloud based detectors. We chose the PV-RCNN detector [32], available in the OpenPCDet toolbox [33]. The pipeline is described in Fig. 12

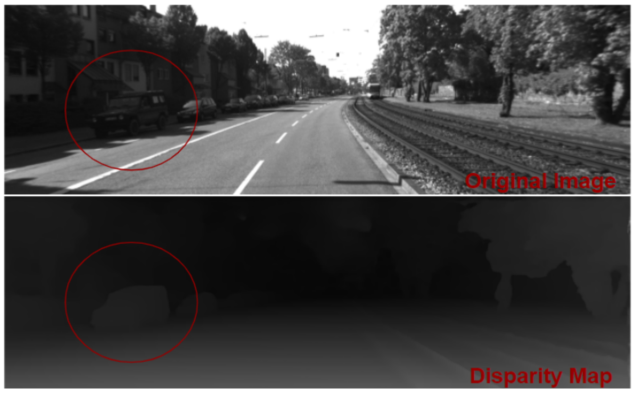


Fig. 11: YOLO vs. PCL Approaches Pipeline

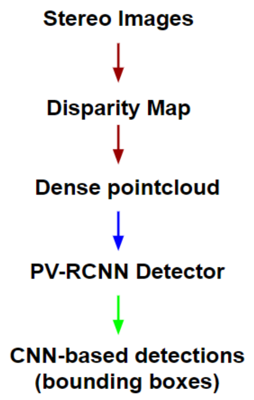


Fig. 12: YOLO vs. PCL Approaches Pipeline

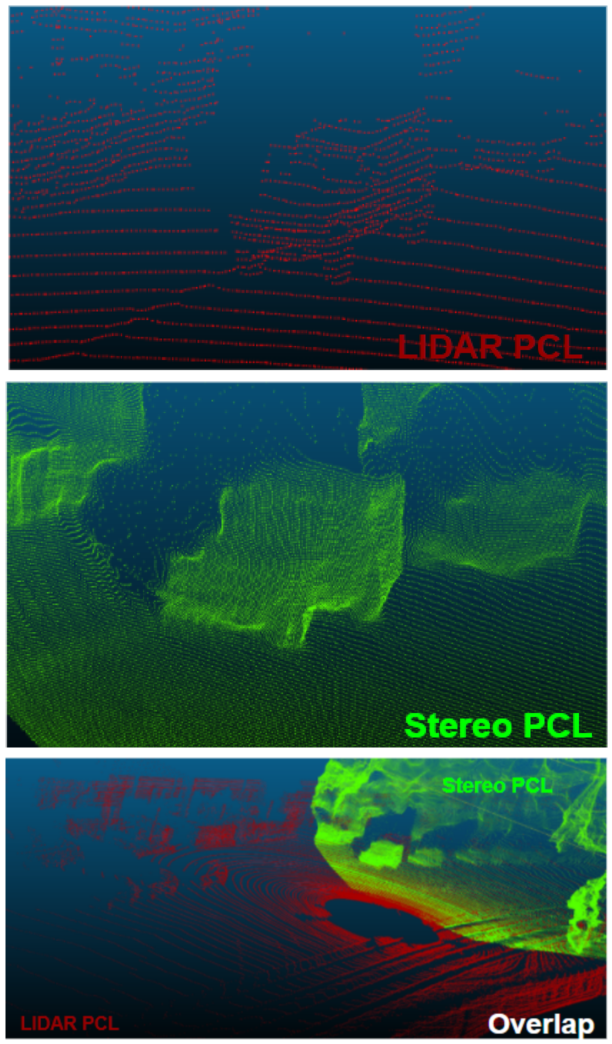


Fig. 13: Comparison of LIDAR and Stereo point-clouds

We fed our point-grid (Fig. 7) into PV-RCNN, and were able to generate vehicle detections for some particularly disparate point-grid frames, though this was mostly the exception not the norm. Most of the stereo-sourced point-clouds were noisy and could not compare in accuracy, field-of-view and consistency to the associated LIDAR point-cloud (Fig. 13). Ideally, if we want to quantitatively compare detection performance between the two clouds in the future, we would need to trim the LIDAR scans to the field of view and range of the stereo sourced point-clouds and possibly retrain the detectors.

4. Results and Analysis

For all reasons mentioned in Sec. 3H, we could generate OBB only using the YOLO+PCA approach. Our primary means of evaluation was visual inspection.

Fig. 14 shows object bounding boxes (OBB) reprojected onto the scene.

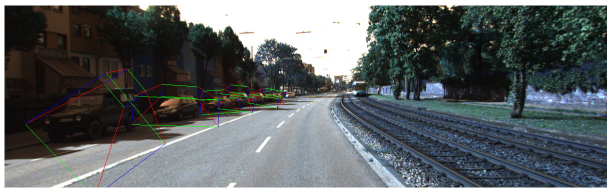


Fig. 14: Sample results: OBB placed on car-line on left

Figs. 15 and Fig. 16 show that trains and cyclists are also identified, in addition to cars in previous images.

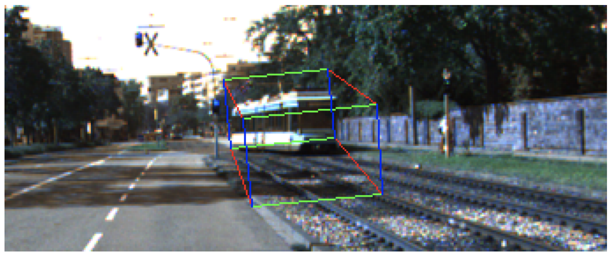


Fig. 15: OBB centered on train

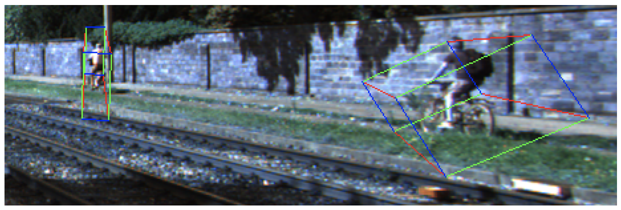


Fig. 16: Two cyclists with respective OBB.

3D bounding boxes were roughly centered on objects of interest. The fit is well-centered and consistent for farther objects and systematically degenerates as the object gets closer to the left of the image (smaller distance and greater azimuth) for both OpenCV and PSMNet. This is presumably due to noise in either method. It could also be due to small mismatches in intrinsic calibration, between the assumed pinhole model and the true camera, becoming pronounced at the image-periphery. Fig. 17 and Fig. 18 compare the 3D bounding boxes from OpenCV-SGBM and PSMNet, respectively. Both fail differently at the image's left due to unique noise-sources.

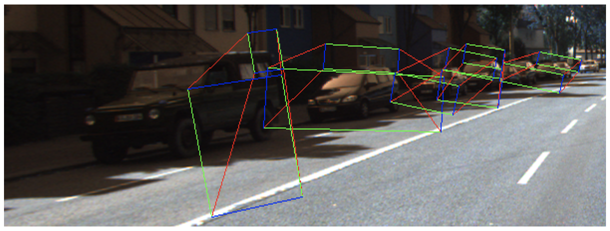


Fig. 17: OBB from OpenCV SGBM disparities

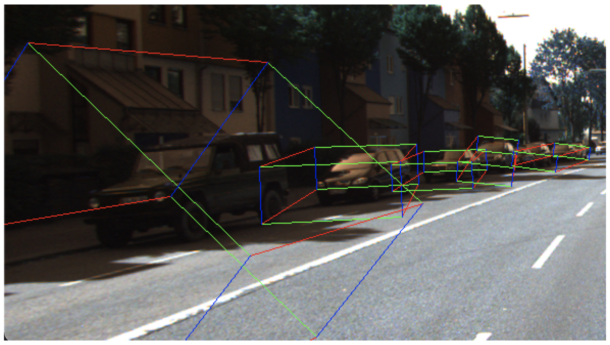


Fig. 18: OBB from PSMNet disparities

OpenCV is unable to compute disparities for a band of "max-disparity" pixels on the left of the image as seen in Fig. 9. Therefore increasing the value of this property compromises depth calculations for objects on the left. PSMNet does not have this problem. Due to rules-of-the-road, few objects appear on the right side and far enough for YOLOv5 detection to work properly. This made it difficult to check for behavior on the right side.

While generally centered on, or in the neighborhood of, objects of interest, our OBB are oblique and not aligned with the object: the boxes generally slope upwards and away from ego. This is expected because there is no constraint to anchor these boxes to the ground plane in "pure" PCA. Disabling ground-plane-rejection did not change the quality of the result because very few ground points are part of the YOLOv5 bounding boxes anyway.

Without foreground-background separation using K-means clustering (K=2), PCA yielded very loose and skewed 3D bounding boxes. In most cases, foreground and background cluster-centers were separated by 5-20 m.

Despite these limitations, the frame-to-frame behavior of our bounding boxes, in a video sequence, is quite good.

YOLOv5 can recognize many object classes (from COCO dataset) and its 2D bounding boxes are tight fits.

The accumulation of noise over all steps leads to curvy object outlines which frustrates RANSAC plane-fitting.

Runtimes for each frame, and for steps therein, are shown in Table 1. We used two platforms - **(1)** Macbook Pro with 2.9 GHz 6-core i9 processor with 32GB RAM running MacOS 11.1 (Big Sur), and **(2)** AWS g4dn instance with 3.1 GHz 4-core Xeon Platinum 8259CL processor and NVIDIA Tesla T4 GPU with 8GB VRAM. The Macbook Pro platform used CPU only whereas the AWS instance made use of GPU as well.

Table 1: Median runtimes for frames and steps (s)

	MacPro PSM	MacPro SGBM	AWS PSM	AWS SGBM
Frame	26.0	1.35	1.2	0.62
Disparity	24.3	0.35	0.97	0.37
Pt. Grid	0.03	0.03	0.02	0.02
YOLOv5	0.93	0.90	0.03	0.06
Carve → OBB	0.017	0.015	0.015	0.013
Misc (eg: IO)	0.72	0.04	0.15	0.16

It can be seen that the neural networks consume bulk of the processing time and that GPUs offer significant speedup. In cloud environments with powerful GPU and file-systems, our method is fast enough to provide a viable foundation for ground-truth calculation at-scale.

5. Possible Extensions

Quantitative comparison with KITTI Object Detection ground-truth is possible (we ran out of time) but this would require writing custom code to extract such information from the dataset (reference code is in MATLAB and not fully documented).

Future work could explore Hough Transform to see if points vote at least for the side-plane of cars, addressing a key limitation of RANSAC. Fusing LIDAR data with stereo data may lead to crisper and more accurate geometry in object point clouds. L-shape fitting on ground-plane-projections [31] could form the basis for best-fit 3D OBB aligned with ground surface. Overall, these improvements may provide good object-pose measurements to feed into state-estimators that use Extended Kalman Filters or Particle Filters. We were hoping to reach this point for this project with our PCA bounding boxes but the learning experience was good. Bayesian Smooth (for example, Extended Kalman smoothing) over the full history of object 3D locations would provide good ground-truth for validation studies.

6. Conclusions

We successfully applied techniques from stereo and mono computer vision, with a few extensions, to detect certain objects and mark their locations with 3D bounding boxes. Our approach is a combination of geometry and deep-learning. Starting with a rectified stereo image-pair, we show that it is possible to combine inferred classification with computed point clouds to carve out objects of interest and to locate them in 3D. Though our implementation is per-frame, our technique works smoothly across frames from a (stereo) video sequence. Limitations of our method can be overcome through multi-sensor fusion and more advanced computations. It is a promising foundation for extracting ground-truth trajectories for dynamic objects in the environment when combined with odometry and localization.

7. Contributions

- Shoaib Lari: Reproduced results of the baseline paper [2], wrote test programs to access the KITTI dataset, and performed Related Work research.
- Manoj Rajagopalan: Implementation of dataset → OBB pipeline shown in Fig. 3. Generated result images and performance table. RANSAC.
- Martin Freeman: Reproduced results of the baseline paper [2], experimented with PV-RCNN on our stereo point-clouds.

8. Code

Code to reproduce detection and tracking results in [2] is located at:
https://github.com/shoaiblari/stereo_based_tracking

Code for our object detection, from input stereo image-pairs to 3D OBB reprojection and RANSAC is located at:
https://github.com/manoj-rajagopalan/stanford_cs231a_project

9. References

[1] H.-k. Chiu, J. Li, R. Ambrus, J. Bohg, “Probabilistic 3D Multi-Modal, Multi-Object Tracking For Autonomous Driving”. arXiv:2012.13755, 2020

[2] H.-k. Chiu, A. Prioletti, J. Li, J. Bohg, “Probabilistic 3d multi-object tracking for autonomous driving,” arXiv:2001.05673, 2020.

[3] nuScenes. (2020) Nuscenes. Dataset. Available at https://www.nuscenes.org/

[4] The KITTI Vision Benchmark Suite. (2020). Available http://www.cvlibs.net/datasets/kitti/

[5] Pykitti package. Available at https://pypi.org/project/pykitti/

[6] The KITTI Vision Benchmark Suite Setup. (2020b). Available at http://www.cvlibs.net/datasets/kitti/setup.php

[7] Go Photonics FL2-14S3C-C camera setup. Available https://www.gophotonics.com/products/scientific-industrial-cameras/point-grey-research-inc/45-571-fl2-14s3c-c.

[8] Point Grey Flea 2 Camera. Available at https://www.surplusgizmos.com/assets/images/flea2-FW-Datasheet.pdf.

[9] opencv-python package. Available at https://pypi.org/project/opencv-python/.

[10] J-R. Chang, Y-S. Chen, “Pyramid Stereo Matching Network”, CVPR 2018.

[11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. “You only look once: Unified, real-time object detection.” CVPR 2016.
YOLO V5. Available at https://github.com/ultralytics/yolov5.

[12] B. Zhu, Z. Jiang, X. Zhou, Z. Li, G. Yu, "Class-balanced Grouping and Sampling for Point Cloud 3D Object Detection", WAD CVPR 2019.

[13] S. Song and J. Xiao. “Deep sliding shapes for amodal 3d object detection in RGB-D images”. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 808–816, 2016.

[14] Y. Zhou and O. Tuzel. Voxelnet: “End-to-end learning for point cloud based 3d object detection”. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4490–4499, 2018.

[15] Y. Yan, Y. Mao, and B. Li. Second: “Sparsely embedded convolutional detection”. *Sensors*, 18(10):3337, 2018.

[16] Y. Chen, S. Liu, X. Shen, and Jiaya Jia. “Fast point R-CNN”. In *Proceedings of the IEEE international conference on computer vision (ICCV)*, 2019.

[17] S. Shi, Z. Wang, X. Wang, and H. Li. “Part-a2 net: 3d part-aware and aggregation neural network for object detection from point cloud”. *CoRR*, abs/1907.03670, 2019.

[18] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. “Multi-view 3d object detection network for autonomous driving”. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[19] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. Waslander. “Joint 3D proposal generation and object detection from view aggregation”. *IROS*, 2018.

[20] B., W. Luo, and R. Urtasun. Pixor: “Realtime 3D object detection from point clouds”. In Proceedings of the *IEEE Conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018.

[21] M. Liang, B. Yang, S. Wang, and R. Urtasun. “Deep continuous fusion for multi-sensor 3D object detection”. In *ECCV*, 2018.

[22] B. Yang, M. Liang, and R. Urtasun. Hdnet: “Exploiting hd maps for 3d object detection”. In *2nd Conference on Robot Learning (CoRL)*, 2018.

[23] A.H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. “Pointpillars: Fast encoders for object detection from point clouds”. *CVPR*, 2019.

[24] M. Liang, B. Yang, Y. Chen, R. Hu, and R. Urtasun. “Multi-task multi-sensor fusion for 3D object detection”. In *CVPR*, 2019.

[25] C.R. Qi, W. Liu, C. Wu, H. Su, and L.J. Guibas. “Frustum pointnets for 3d object detection from rgb-d data”. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[26] C.R. Qi, H. Su, K. Mo, and L.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.

[27] C. Ruizhongtai Qi, L. Yi, H. Su, and L.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.

[28] S. Shi, X. Wang, and H. Li. Pointrcnn: “3D object proposal generation and detection from point cloud”. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–779, 2019.

[29] Z. Wang and K. Jia. Frustum convnet: “Sliding frustums to aggregate local point-wise features for amodal 3D object detection”. In *IROS*. IEEE, 2019.

[30] Z. Yang, Y. Sun, S. Liu, X. Shen, and J. Jia. “STD: sparse-to-dense 3d object detector for point cloud”. *ICCV*, 2019.

[31] S. Qu, G. Chen, C. Ye, F. Lu, F. Wang, Z. Xu, Y. Ge: “An Efficient L-Shape Fitting Method for Vehicle Pose Detection with 2D LiDAR”. *IEEE ROBIO*, 2018.

[32] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. “PV-RCNN: Point-Voxel feature set abstraction for 3d object detection”. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020

[33] OpenPCDet: LIDAR-based 3D object detection. Available at https://github.com/open-mmlab/OpenPCDet