BEV Object Tracking for LIDAR-based Ground Truth Generation

David Montero*, Nerea Aranjuelo*, Orti Senderos* and Marcos Nieto*
*Vicomtech, Mikeletegi 57, P. Tecnológico, 20009, San Sebastián, Spain
Email: see http://www.vicomtech.org

***Abstract*—Building ADAS (Advanced Driver Assistance Systems) or AD (Autonomous Driving) vehicles implies the acquisition of large volumes of data and a costly annotation process to create labeled metadata. Labels are then used for either ground truth composition (for test and validation of algorithms) or to set-up training datasets for machine learning processes. In this paper we present a 3D object tracking mechanism that operates on detections from point cloud sequences. It works in two steps: first an online phase which runs a Branch and Bound algorithm (BBA) to solve the association between detections and tracks, and a second filtering step which adds the required temporal smoothness. Results on KITTI dataset show the produced tracks are accurate and robust against noisy and missing detections, as produced by state-of-the-art deep learning detectors.**

I. Introduction

In recent years, huge improvements have been made in the field of Advanced Driver Assistance Systems (ADAS). Current ADAS are capable to perceive the surrounding environment of the vehicle by using sensors and take real-time decisions for safety (emergency braking, lane departure warning) or comfort (lane keeping system, automatic cruise control).

Most of these achievements have been possible thanks to the continuous and rapid advances in Deep Learning (DL) [1][2]. New DL models can process the data captured by different sensors, such as cameras or LIDARs, to obtain a precise estimation of the scene (e.g. other vehicles, pedestrians, traffic signs). However, the accuracy of these models depends almost entirely on the richness of the dataset used for training.

DL models require training with huge amounts of data precisely annotated. The annotation process is costly and slow, usually requiring large groups of human operators creating the labels with specific tools. This process is the main bottleneck for the improvement of DL and as a consequence, ADAS.

In order to reduce this cost and accelerate the process, many annotation tools, automatic and semi-automatic, have been developed using different approaches [3][4]. Though, automatic annotation tools are not perfect and they introduce errors or inaccuracies to the annotations, which need always to be corrected or, in the best case, validated as ground truth. In the case of object annotation, it is critical that the automated step produce labels which are not only accurate in space (e.g. a bounding box in 2D or cuboid in 3D), but also coherent in time, thus assigning single identifiers to objects through the sequence. In this paper we propose an offline 3D object tracking algorithm as a part of a semi-automatic annotation tool for LIDAR data.

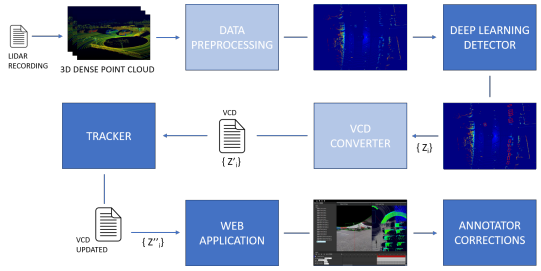

Fig. 1. Diagram of the annotation process.

Multi-object tracking is usually focused on solving how to associate incoming new detections to existing tracks [5], in an online process whose main requirement is to operate real-time. Offline processes, on the contrary, tend to find a graph with all possible associations and solve it in a joint optimized process, which result in very slow batch processes.

In our work we propose an approach which is as fast as online processes, by means of defining an online frame-level association problem, but solving it with a recursive function which ensures optimal association, plus additional post-processing steps which provide the necessary estimation smoothness. Our approach aims to obtain the highest possible accuracy in the least possible time in order to work effectively in a semi-automatic annotation process.

II. System overview

Figure 1 illustrates the pipeline of the annotation tool which includes the proposed tracking component. The annotation process is divided in a series of steps, starting with the generation of the recordings from the sensorized vehicles.

LIDAR streams (3D point cloud sequences) are then pre-processed to create bird's-eye view (BEV) images (also called top view images, see section III-A), which are then used as input for the Convolutional Neural Network (CNN) detectors (see section III). The detector outputs are cuboids in 3D space, defined as $\mathbf{Z}_{t,n} = (x, y, z, r_x, r_y, r_z, w, h, l, c, s)$, where t is the time step, n is the detected object number in that time step, $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ are the object center coordinates, $\mathbf{r_x}$, $\mathbf{r_y}$ and $\mathbf{r_z}$ are the object rotation angles, $\mathbf{w}$, $\mathbf{h}$ and $\mathbf{l}$ are it width, height, and length, $\mathbf{c}$ is the class number, and $\mathbf{s}$ is the confidence of the detection. This data will be converted into a standard format using the VCD converter[1], and will serve as the input

[1]https://vicomtech.box.com/v/vcd-library-linux-windows

978-9-0827-9703-9/19/$31.00 ©2019 IEEE

data for the tracker. The tracker will associate the input object detections between the different time steps, generate predictions where detections are missing and finally correct the input object properties by applying a post-process to the generated tracks (see section IV). As a result, the VCD payload describing the scene is updated with the tracking information and sent to a web application for the final annotation step carried out by human operators.

III. Object Detection

Using latest advances in object detection with DL algorithms [1], we have trained a CNN to generate oriented bounding box detections based on LIDAR point clouds.

A. 3D Point Cloud Representation

A Velodyne HDL-64E laser scanner produces about one million 3D points per second [6]. Applying detection algorithms directly on these point clouds is normally avoided because of their sparse nature and the large amount of data to be processed [7][8]. Due to the high dimensionality and sparsity of the data, we have adopted an approach based on BEV map data representation. The point cloud is projected to the ground plane and discretized into a 2D grid of cells with resolution of 0.1 x 0.1m. Only the points inside a range of [(-40, 40), (-40, 40), (-1.75, 1.25)] meters are considered, taking the LIDAR position as the origin of coordinates. For each cell, the minimum, average and maximum heights of all the contained points are stored. These 3 features are stored as a 3-channel matrix with size of 800×800 and fed into a CNN.

B. Deep neural network

The trained network takes the encoded feature maps as input and produces oriented 3D boxes for the considered object classes. The network is based on Faster R-CNN architecture [9], which consists on two main stages. First, a Region Proposal Network (RPN) generates bounding boxes with object candidates. In the second stage, for each box proposal, extracted features are used to classify it, as well as to regress the final box coordinates as well as its rotation angle. Non-maximum suppression (NMS) is used to reduce redundancy of highly overlapped boxes. We use a IoU (Intersection-over-Union) threshold of 0.7 for NMS. The backbone network we use for feature extraction is ResNet-101 [10].

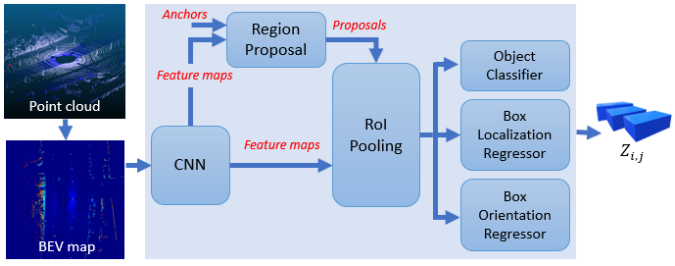

Fig. 2. Neural network architecture for oriented box detection.

The model is optimized for a multi-task loss function, which combines classification and bounding box regression losses [9]. Different from the original Faster R-CNN architecture, which does not predict the orientation of the boxes, an extra loss term is added [11] to minimize the error between the estimated rotated box and the ground truth.

The network is trained in an end-to-end fashion. Weights are initialized with pretrained weights on ImageNet data [12]. The training dataset is generated with the sequences published so far from the public driving dataset nuScenes [13]. Augmentation techniques are applied to augment samples containing the least frequent classes. The total amount of training samples is 20000, including 9 different object classes: car, pedestrian, cyclist, train, truck, bus, motorist, construction vehicle and trailer. Custom anchor boxes are designed for each class.

For each time step t, each object prediction is parameterized by $\mathbf{Z}_{t,d}$ as defined in previous section. The height of the bounding box is estimated based on the object class.

IV. Tracking

As mentioned in section II, the tracker is in charge of associating the input objects between the different time steps, generating missing tracking states in intermediate steps and correcting the input object properties by applying a post-process to the generated tracks. A diagram describing the tracking process can be found in Figure 3.

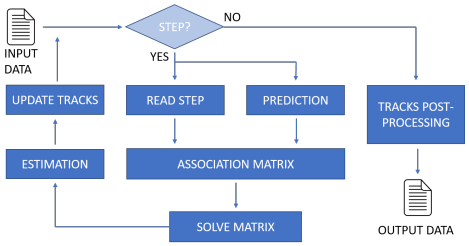

Fig. 3. Offline tracking dataflow where input data stands for the obtained CNN detections, and output data are the updated tracks.

The tracking process is divided into two main components. First, the online component is executed as a loop where, at each iteration, a new time step (i.e. frame) of the input data is processed. Tracks are then updated using predictions based on their previous information and the new input detections. In the correction step an association matrix is generated and solved in order to match existing tracks predictions with detected objects and the tracks are updated. Finally, a post-process adds smoothness to produced tracks both in position and rotation.

A. Linear prediction

It is assumed that detections are filtered by their score using a suitable user-defined threshold before feeding the tracker, so the variable of $\mathbf{Z}_{t,n}$ is not used during the tracking process.

After analyzing the data and the use case, the following conclusions were reached. z position, r_x and r_y is approximately constant in almost every case, therefore it does not provide relevant information. Also, r_z is a noisy variable and will be corrected in the post-process. So we decided to discard these variables from the prediction model. Similarly, the object size

and the class variables are assumed to be constants, so they are not considered for the prediction.

After this filter, the prediction variables are x, y and r_z. For solving this problem, it was decided to use a constant acceleration model. So, considering all the previous information, a track state will be represented by $\mathbf{S}_{t,m} = (x, y, \dot{x}, \dot{y}, \ddot{x}, \ddot{y}, z, r_x, r_y, r_z, w, h, l, c)$, where t is the time step, and m is the index of the track.

B. Association matrix

For the association between the tracks states and the detections an association matrix is defined. Let this matrix be $A_{m \times n}$ where M is the number of existing tracks, and N is the number of incoming detections.

Each entry of A encodes the association likelihood (between 0 and 1) of track m and detection n. Before computing the likelihood, it is checked that the track and the detection class belongs to the same group. Two groups are defined based on the detector output classes: a vehicle class (cars, trucks and buses) and a human class (pedestrians, cyclists and bikers). This is applied because classes of the same group are more likely to be confused by the detector.

Only the 2D position variables (x and y) are taken in account for the likelihood computing, due to the noise in the r_z, s_x and s_y. The likelihood function can be selected as any decaying function around the predicted state of the track. Therefore, we need first to define the distance between the centroids of the predicted track $\mathbf{p} = (p_x, p_y)^\top$ and the detection $\mathbf{q} = (q_x, q_y)^\top$, as $\mathbf{d}' = (\mathbf{p} - \mathbf{q})$.

The model must include the uncertainty of the prediction itself and the noise of the detections. For convenience, we use a bivariate normal distribution on a normalized distance $\mathbf{d}$ between the centroids of the predicted track and the detection. This distance will be rotated so the $\mathbf{x}$ axis aligned with the velocity vector, using $\theta = atan(v_y, v_x)$:

$$d' = d' \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix} \tag{1}$$

Thus, the distance is normalized as:

$$\mathbf{d} = \left(\frac{max(0, |d'_x| - \epsilon)}{d_{max}}, \frac{max(0, |d'_y| - \epsilon)}{d_{max}} \right)^\top \tag{2}$$

where ϵ is the detection noise, and d_{max} is the maximum association distance, which depends on the class group of the track. This normalization ensures that the distance measure includes the detection noise.

The likelihood function is then defined as:

$$\mathcal{L}(\mathbf{d}) = \frac{1}{\sqrt{(2\pi)^2 |\sigma|}} exp(-\frac{1}{2}(\mathbf{d}^\top \sigma^{-1} \mathbf{d})) \tag{3}$$

where σ is the covariance matrix which defines the shape of the likelihood function. As we need a likelihood function ranged between 0 and 1, we need to select $\sigma_x = \sigma_y = \frac{1}{\sqrt{2\pi}}$. Then, the likelihood is redefined as:

$$\mathcal{L}(\mathbf{d}) = exp(-(||\mathbf{d}||^2 \pi)) = exp(-d_x^2 \pi) exp(-d_y^2 \pi) \tag{4}$$

As we want to include the uncertainty of the prediction process, we truncate this function to a certain limit, defined by an ellipse aligned with the velocity vector of the track and which axis length will depend on the prediction certainty. The axes of the ellipse can be defined as:

$$\begin{aligned} s_x &= d_{max} + (1 + ||v||(\Delta t + 1))\gamma_x \\ s_y &= d_{max} + (1 + ||v||(\Delta t + 1))\gamma_y \end{aligned} \tag{5}$$

where $||\mathbf{v}||$ is the magnitude of the velocity vector of the track, Δt depends on the number of steps passed since the last association of a detection with the track, and γ_x and γ_y are factors which elongate the ellipse according to the expected object dynamics (e.g. for cars, we have found that $\gamma_x = 2.0$ and $\gamma_y = 0.3$ encode well its inertia).

Then, the ellipse is defined as $\mathbf{d} S^{-1} \mathbf{d} = 1$, where S is the matrix that defines the axes of the ellipse: $S = diag(s_x^2, s_y^2)$.

C. Association matrix resolution

Solving correctly the association matrix is crucial for the tracking, as a wrong or sub-optimal match in a time step propagates the error for the rest of the sequence. Two ways of solving it are considered and tested: (i) the Greedy Algorithm (GA)[14], used as baseline for comparison; and (ii) our proposed modified Branch and Bound Algorithm (BBA) [15].

GA works selecting individually the global maximum out of the matrix and setting to zero its corresponding row and column, and iterating finding next maxima until the matrix is emptied. BBA works, instead, in a recursive manner, finding maxima and then exploring the tree of possible suboptimal associations in order to find better joint likelihoods for the entire matrix. BAA guarantees that the sum of the likelihood of the produced associations is optimal, while GA does not.

GA is usually faster than BBA when there is a high number of candidates, but results using BBA are better in most of the cases. The two algorithms were implemented and tested in different scenarios. The results of some of those tests are presented in table I, where the errors are the number of steps in which the sum of the likelihood is worse using the GA, Max diff is the maximum difference of the BBA and GA likelihood sum registered, GA time is the average time of the GA implementation execution and the BBA time is the algorithm implementation execution, both in microseconds and per frame. Although this table shows relatively small errors of the GA results at frame-level, the impact of these errors is extremely significant as they are propagated to the rest of the sequence causing erroneous tracks.

TABLE I
ASSOCIATION ALGORITHMS COMPARISON

Scenario	Steps	Errors	Max diff	GA time	BBA time
Street road	611	10	0.423	20 μs	35 μs
Parking 1	189	5	0.703	29 μs	45 μs
Parking 2	404	6	0.559	110 μs	132 μs

Considering the tests results and the priority of the accuracy over the processing time it was decided to use a modified

BBA as the matrix solver. In the proposed implementation, the algorithm starts associating each track with the detection with the best likelihood. Then it looks for collisions (where a collision stands for the case where two tracks have been associated with the same detection). If it finds any, then a recursive function will be called until it finds the best collisions free combination. The algorithm can be found in algorithm 1, where candidates variable is the association vector.

Algorithm 1 B&B Recursive Function

procedure SOLVECOL($candidates, a, b$)
  $candidatesA \leftarrow candidates$
  $candidatesA(a) \leftarrow getNextCandidate(a)$
  $candidatesA \leftarrow checkCol(candidatesA)$
  $scoreA \leftarrow \sum_{i=0}^{n\_tracks-1} \mathcal{L}_{i,candidatesA(i)}$
  $candidatesB \leftarrow candidates$
  $candidatesB(b) \leftarrow getNextCandidate(b)$
  $candidatesB \leftarrow checkCol(candidatesB)$
  $scoreB \leftarrow \sum_{i=0}^{n\_tracks-1} \mathcal{L}_{i,candidatesB(i)}$
  if $scoreA \geq scoreB$ **then**
    return $candidatesA$
  return $candidatesB$
procedure CHECKCOL($candidates$)
  for $i \leftarrow 0; i < n\_tracks - 1; i \leftarrow i + 1$ **do**
    for $j \leftarrow i + 1; j < n\_tracks; j \leftarrow j + 1$ **do**
      if $candidates(i) = candidates(j)$ **then**
        $candidates \leftarrow solveCol(candidates, i, j)$
  return $candidates$

D. Estimation and tracks updating

Once the association matrix is solved, the tracks new state will be estimated. The position estimation will be calculated correcting the detection position using the equation 6, where $\mathbf{p}$ can be x or y, $dist_p$ is the difference between prediction and detection positions, $\mathbf{predn}_p$ is the prediction noise, which depends on the velocity, the number of steps without detections (Δt) and a noise coefficient, and $\mathbf{detn}$ is the detection noise.

$$p_{est} = p_{det} + (max(0, dist_p - max(predn_p - detn), 0)))/2$$
$$predn_p = (dist_p/||dist_p||)||v_p||\Delta t\gamma_p \quad (6)$$

The velocity in $\mathbf{x}$ and $\mathbf{y}$ is calculated by the difference between the new and the last estimated positions divided by the difference of steps, and smoothed using a linear interpolation with the last 3 velocity values registered. The same method is used for the acceleration in $\mathbf{x}$ and $\mathbf{y}$. For the rest of the state variables, the value of the detection variables will be assign, since they will be treated in post-processing.

After the new tracks states are computed, the missing tracks states from previous steps will be generated using a linear interpolation. Also, it will be checked if there are dead tracks (without an associated detection in the last $\mathbf{n}$ steps) and separate them from the active ones. Finally, new tracks will be generated using the unassociated detections.

E. Tracks post-processing

Once the online stage has finished, the post-processing stage will begin. The aim of this stage is to remove orphan tracks and to correct the detection noisiest variables, the z axis rotation angle and the class. They are considered as orphan tracks all the tracks that has less than $\mathbf{n}$ states, and they will be removed, since they have a high probability of being erroneous.

For the $\mathbf{r}_z$ correction, it is assumed that, in most cases, the detection value is right, and that it wont change drastically from one step to another, so it will be calculated using the mode between $\mathbf{s} - \mathbf{n}$ and $\mathbf{s} + \mathbf{n}$ steps, being $\mathbf{s}$ the current step number. Also, the track size will be corrected using the mode of every track states, as it should be constant in every step. Finally, for the track class variable, it is again assumed that, in most cases, the detection value is right, so it will receive the value of the mode between all track steps.

V. TESTS & DISCUSSION

In this section we present an experimental analysis of our method. We validate the tracker using the first 10 of the 22 training sequences with ground truth of the KITTI dataset with the tracking metrics proposed in [16].

In the ideal situation when detections are perfect, using the ground truth boxes from the KITTI dataset, we have observed that the tracker produces perfect tracks, without error.

A. Ablation study

We perform an ablation study on the input data in order to measure the impact of imperfect detections on the tracking results. We define three types of detection problems: (i) spatial noise; (ii) temporal sparsity; and (iii) a combination of both. The first test evaluates the effect of translation, rotation and size measurement noise in the detections. The temporal sparsity refers refers to the miss-detections of objects in specific frames. For the first test define a probability of 50% to apply randomly a noise between -20% and 20% to each detection in each frame. The experiment is repeated 10 times; the average results for each sequence are shown in Table II.

TABLE II
TRACKER RESULTS WITH SPATIAL NOISE

Seq	MOTA	F1	MT	PT	ML	FRAG
0	0.8626	0.9319	12.0	0.0	0.0	33.0
1	0.8561	0.9292	90.4	1.6	0.0	177.1
2	0.8684	0.9346	15.6	0.4	0.0	68.4
3	0.8485	0.9253	9.0	0.0	0.0	25.8
4	0.8511	0.9270	29.3	1.6	0.1	59.7
5	0.8402	0.9212	33.5	0.5	0.0	91.6
6	0.8383	0.9203	12.8	0.2	0.0	48
7	0.8520	0.9268	56.7	0.3	0.0	163.2
8	0.8459	0.9237	24.2	0.8	0.0	92.8
9	0.8664	0.9338	87.0	1.0	0.0	185.9

Temporal sparsity is analyzed following the same process and suppressing 20% of each object detections randomly across the sequence (see Table III).

For the third test we combine the spatial noise and the temporal sparsity to simulate a real detector (see Table IV). We can observe the tracker is able to solve temporal sparsity better than spatial noise, since, in the correction stage, the tracker assumes the position of detections are highly reliable.

TABLE III
TRACKER RESULTS WITH TEMPORAL NOISE

Seq	MOTA	F1	MT	PT	ML	FRAG
0	0.9897	0.9964	12.0	0.0	0.0	3.8
1	0.9940	0.9983	92.0	0.0	0.0	8.4
2	0.9984	0.9996	16.0	0.0	0.0	0.9
3	0.9907	0.9977	9.0	0.0	0.0	1.8
4	0.9987	0.9997	31.0	0.0	0.0	0.6
5	0.9985	0.9995	34.0	0.0	0.0	1.0
6	0.9991	0.9998	13.0	0.0	0.0	0.3
7	0.9981	0.9995	57.0	0.0	0.0	2.4
8	0.9997	0.9999	25.0	0.0	0.0	0.2
9	0.9913	0.9977	88.0	0.0	0.0	13.4

TABLE IV
TRACKER RESULTS WITH COMBINED NOISE

Seq	MOTA	F1	MT	PT	ML	FRAG
0	0.8770	0.9409	12.0	0.0	0.0	29.6
1	0.8778	0.9417	91.0	1.0	0.0	146.5
2	0.8935	0.9472	15.9	0.1	0.0	55.3
3	0.8454	0.9251	9.0	0.0	0.0	28.0
4	0.8735	0.9380	29.9	2.0	0.1	47.3
5	0.8640	0.9340	33.4	0.6	0.0	73.6
6	0.8693	0.9360	13.0	0.0	0.0	35.3
7	0.8689	0.9360	56.9	0.1	0.0	144.8
8	0.8771	0.9392	24.2	0.8	0.0	73.3
9	0.8783	0.9419	87.5	0.5	0.0	166.7

B. Study on generated detections

We finally analyze the results of the tracking when the input estimations are generated by the trained point cloud based object detector (see section III). The same evaluation metrics are computed for this experiment. Results are shown in Table V, where there is an extra column (F1D) with the F-Score obtained with the detector output. In this experiment the

TABLE V
TRACKER RESULTS WITH REAL DETECTOR

Seq	MOTA	F1	F1D	MT	PT	ML	FRAG
0	0.5476	0.7650	0.7468	5	6	1	14
1	0.5389	0.7631	0.7420	50	20	21	92
2	0.3094	0.6491	0.6340	6	5	0	20
3	0.6265	0.8095	0.7800	4	2	3	9
4	0.6814	0.8490	0.7965	20	9	1	25
5	0.6280	0.8150	0.7077	14	9	11	34
6	0.7438	0.8788	0.8382	8	3	2	15
7	0.7000	0.8542	0.8473	43	11	3	70
8	0.5069	0.7554	0.7290	12	8	4	26
9	0.3406	0.6487	0.6382	29	26	27	106

MOTA results with real detections are, as expected, worse than with ground truth detections due to their inherent noise. However, there is always an improvement of F-Score, ranging between $2-5\%$ thanks to the usage of the tracker.

VI. CONCLUSIONS

In this work, we have shown our implementation of a 3D object offline tracking technique, which is robust against noisy and sparse detections produced by deep learning detection frameworks. Our approach combines the benefits of online tracking schemas, and thus operating near real-time, and the reliability of batch processes, by means of applying a recursive implementation of the Branch and Bound algorithm (BBA) to optimally solve the association problem.

In our experiments we show the BBA algorithm is optimal with respect the usual Greedy Algorithm (GA) approach, while keeping its computation under real-time requirements. Additionally, we have explored the impact of the noise and sparsity of detections benchmarking our proposed work against the ground truth from the KITTI dataset.

Furthermore, the proposed tracker is integrated into a web-based annotation platform which takes the produced tracks and presents them to teams of human annotators which refine, correct and finally validate the annotations. Future work will include a refinement of the score function at track level, so that the interaction between the algorithm and the annotators can be extended to offer finer level of control to the users.

VII. ACKNOWLEDGMENTS

This work has received funding from the European Commission (EC) Horizon 2020 programme (grant agreement no. 688099, project Cloud-LSVA).

REFERENCES

[1] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *arXiv preprint arXiv:1809.02165*, 2018.
[2] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez, "A review on deep learning techniques applied to semantic segmentation," *arXiv preprint arXiv:1704.06857*, 2017.
[3] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, "Labelme: a database and web-based tool for image annotation," *International journal of computer vision*, vol. 77, no. 1-3, pp. 157–173, 2008.
[4] S. Bianco, G. Ciocca, P. Napoletano, and R. Schettini, "An interactive tool for manual, semi-automatic and automatic video annotation," *Computer Vision and Image Understanding*, vol. 131, pp. 88–99, 2015.
[5] H. S. Parekh, D. G. Thakore, and U. K. Jaliya, "A survey on object detection and tracking methods," *IJIRCCE*, vol. 2, pp. 2970–2979, 2014.
[6] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361, IEEE, 2012.
[7] Y. Zeng, Y. Hu, S. Liu, J. Ye, Y. Han, X. Li, and N. Sun, "Rt3d: Real-time 3-d vehicle detection in lidar point cloud for autonomous driving," *IEEE Robotics and Automation Letters*, vol. 3, pp. 3434–3440, 2018.
[8] B. Yang, W. Luo, and R. Urtasun, "Pixor: Real-time 3d object detection from point clouds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7652–7660, 2018.
[9] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.
[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
[11] Y. Jiang, X. Zhu, X. Wang, S. Yang, W. Li, H. Wang, P. Fu, and Z. Luo, "R2cnn: rotational region cnn for orientation robust scene text detection," *arXiv preprint arXiv:1706.09579*, 2017.
[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[13] "Nuscenes." https://www.nuscenes.org/. (Accessed on 25/02/2019).
[14] J. Edmonds, "Matroids and the greedy algorithm," *Mathematical programming*, vol. 1, no. 1, pp. 127–136, 1971.
[15] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Operations research*, vol. 14, no. 4, pp. 699–719, 1966.
[16] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler, "Mot16: A benchmark for multi-object tracking," *arXiv preprint:1603.00831*, 2016.