Real-Time Bird's Eye View Multi-Object Tracking system based on Fast Encoders for Object Detection

Carlos Gómez-Huélamo[1], Javier Del Egido[1], Luis M. Bergasa[1], Rafael Barea[1], Manuel Ocaña[1], Felipe Arango[1], Rodrigo Gutiérrez-Moreno[1]

***Abstract*— This paper presents a Real-Time Bird's Eye View Multi Object Tracking (MOT) system pipeline for an Autonomous Electric car, based on Fast Encoders for object detection and a combination of Hungarian algorithm and Bird's Eye View (BEV) Kalman Filter, respectively used for data association and state estimation. The system is able to analyze 360 degrees around the ego-vehicle as well as estimate the future trajectories of the environment objects, being the essential input for other layers of a self-driving architecture, such as the control or decision-making. First, our system pipeline is described, merging the concepts of online and real-time DATMO (Deteccion and Tracking of Multiple Objects), ROS (Robot Operating System) and Docker to enhance the integration of the proposed MOT system in fully-autonomous driving architectures. Second, the system pipeline is validated using the recently proposed KITTI-3DMOT evaluation tool that demonstrates the full strength of 3D localization and tracking of a MOT system. Finally, a comparison of our proposal with other state-of-the-art approaches is carried out in terms of performance by using the mainstream metrics used on MOT benchmarks and the recently proposed integral MOT metrics, evaluating the performance of the tracking system over all detection thresholds.**

keywords: 3D Multi-Object Tracking, ROS, Real-Time, Evaluation Metrics.

I. INTRODUCTION

One of the key concepts when developing safe Autonomous Driving Systems (ADS) is the perception of the environment. Furthermore, the reliability of the Collision Avoidance System (CAS) lies on the performance of the environment detector and its ability to predict future situations. In that sense, a real-time Multi-Object Tracking (MOT) system is essential for self-driving, representing the most important module of the perception layer in a fully-autonomous driving architecture. The improvements in object detection in the last years have allowed the research community, specially those groups related to ADS, to focus on MOT techniques, yielding higher accuracy at the cost of computational cost and complexity, making its use prohibitive in real-time systems.

MOT systems aim to estimate the orientation, location and scale of all the objects in the environment over time. While object detection only captures the information of the environment in a single frame, a tracking system must take temporal information into account, filtering outliers (a.k.a false positives) in consecutive detections and being robust to partial or full occlusions. When travelling throughout a route programmed by the path-planner, the vehicle may detect an undetermined number of unforeseen objects over which the MOT module of the ADS should consider only the most relevant from a safety point of view (such as pedestrians, cyclists or cars) to predict and monitor their trajectories. Then, the vehicle can use the evolution of the scene over time to infer driving behaviour and motion patters for improved forecasting.

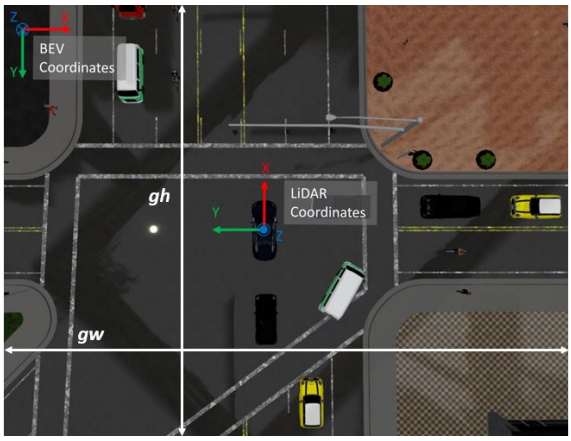

Fig. 1. LiDAR to BEV coordinates transformation illustrated in CARLA simulator

In this work, we approach the MOT problem with a simple yet accurate combination of traditional techniques such as Kalman Filter (KF) [1] and Hungarian algorithm (HA) [2] for state estimation and data association respectively. Nevertheless, most MOT approaches [3] [4] model the state of each obstacle with its 3D position, scale, orientation and their corresponding linear and angular velocity. These approaches introduce an unnecessary complexity and computational cost to the system since most traffic scenes can be described in terms of 2D position, angular and linear velocity, apart from the orientation and scale of the resulting bounding box, that is, a Bird's Eye View (BEV), as depicted in Fig. 1. The prediction step is featured by a constant linear and angular velocity, being the unknown accelerations modelled as Gaussian random variables. The input of the update step of the KF is fed with the output of our 3D object detector (PointPillars) [5] as measurements. PointPillars is a state-of-the-art 3D object detector based on fast encoders that achieves accurate

[1]Carlos Gómez-Huélamo, Javier del Egido, Luis M. Bergasa, Rafael Barea, Manuel Ocaña, Felipe Arango and Rodrigo Gutiérrez-Moreno are with the Electronics Department, University of Alcalá (UAH), Spain. cram3r95@gmail.com, {javier.egido, luism.bergasa, rafael.barea, manuel.ocanna}@uah.es, {juanfelipe.arango, rodrigo.gutierrez}@edu.uah.es

detections at a high frame rate taking advantage of 2D convolutions and Graphical Processing Unit (GPU) to extract clusters by using the 3D pointcloud of the environment around the vehicle. In a similar way to the prediction model, the noise associated to the model measurement is featured by Gaussian random variables. Regarding data association between the actual and predicted object detections, we use the 2D Intersection-Over-Union (2D-IoU) in BEV plane instead of using the 3D-IoU version applied in the AB3DMOT baseline [3] and other previous works. The affinity matrix of the HA is then computed using the BEV-IoU between every pair of detection and predicted trajectories. Moreover, we exploit the concepts of standard communication in robotics using the Robot Operating System (ROS) [6] and lightweight Linux containers for consistent software development and deployment using Docker [7].

II. RELATED WORKS

A. 3D Multi-Object Tracking

A MOT system is basically divided in two sequential stages: First, an object detector must obtain the most relevant obstacles in the scene. In that sense, the scene is mainly analyzed by using cameras (2D object detectors), LiDAR (3D object detectors) or a combination based on their fusion.Second, a tracking module, based on a combination of data estimation and association techniques, is used to monitor the obstacles throughout the scene.

Both 2D and 3D MOT systems can be split into two branches based on the way data association is performed: Online and batch methods. While batch methods aim to find the global optimal solution by using the whole sequence, using network flow graphs that can be solved by minimum cost flow algorithms [8] [9], online methods take into account the data association as a bipartite graph matching problem traditionally solved by a HA.

On the other hand, regarding motion estimation and trajectory prediction, analyzing the scene with a 2D object detector usually designs the appearance and motion models in BEV/3D space adding perspective distortion. Some works attempt to solve this problem from different approaches. [10] uses an Unscented Kalman Filter (UKF) in the BEV space to estimate both the linear and angular velocity of the obstacles. [11] approaches the object detection to an image-based method that estimates the obstacles location in image plane in addition to their 3D distance to camera. Then, a Poisson Multi-Bernoulli Mixture (PMBM) filter is applied to estimate the velocity of the obstacles in the 3D space. As observed, previous works use relatively complicated filters to predict, in an accurate way, the spatial features of the obstacles in the scene. In this work, we use [3] as our baseline, which employs a 3D KF for tracking, where each obstacle state includes the 3D centroid position, rotation angle and 3D bounding box dimensions, excluding the angular velocity. According to the original system pipeline, tracking is fed by the results of the 3D object detector throughout the whole scene, which does not match with real-time requirements. In our case, we conduct real-time tracking-by-detection system, creating or removing tracklets along the sequence in order to get a better perspective of what is happening around the ego-vehicle.

B. 3D Object Detection

As stated above, detecting the bounding boxes directly in 3D has the potential to design the appearance and motion models in 3D space without perspective distortion [3], instead of detecting the obstacles in image plane and then retrieving their remaining 3D information. Naturally, the quality of the detected bounding boxes is essential for the final tracking accuracy. Modern LiDAR based 3D object detection usually belong to one of two branches [4]: Point- or Voxel-based methods. Regarding voxel-based methods, they first divide the input point cloud (3D space) into equally-sized 3D voxels to generate 3D feature tensors based on the points inside each voxel. Then, the feature tensors are fed to 3D CNNs to predict the position of the bounding boxes. On the other hand, point-based methods do not required this quantization step, but they directly apply PointNet++ [12] on the input point cloud for detecting the objects in the 3D space. In this work we use PointPillars [5], a voxel-based state-of-the-art 3D object detector, due to its computational efficiency. This end-to-end network analyzes the raw point cloud by clustering it into upright columns to process the data as a 2D pseudo-image in such a way that highly efficient 2D convolutions can be applied on GPU, not requiring performing a fine-tuning process to improve the quality of the network parameters. We trained it by using the KITTI [13] Multi-Object Tracking dataset, containing information of vehicles, cyclists and pedestrians in arbitrarily complex urban environments.

III. OUR APPROACH

As commented above, we use AB3DMOT [3] as our baseline. Even though their MOT architecture attempts to obtain a real-time 3D MOT system, in practice this is not true because it uses offline tracking.

In this approach, we propose to merge the concepts of online and real-time DATMO (Deteccion and Tracking of Multiple Objects), ROS (Robot Operating System) [6] and Docker [7] to enhance the integration of this MOT system in fully-autonomous driving architectures [14]. Although [3] bases its tracking module in the SORT (Simple Online and Real-time Tracking) algorithm [15], its potential has not been yet exploited, since so far, it has been offline executed as in batch tracking methods, feeding the tracking module with a single text file containing all detections in the scene over time, and then generating an unique identifier by using an affinity matrix [2] and 3D-IOU. Instead, based of the bounding boxes retrieved by our 3D object detector and the ROS network, our modified SORT algorithm transforms the 3D bounding boxes features, which are in LiDAR system coordinates and in real units (m), to BEV image coordinate system in image units (pixels), translating the origin from the ego-vehicle to the top-left corner of the grid analyzed by the 3D object detector. After the tracking stage is performed,

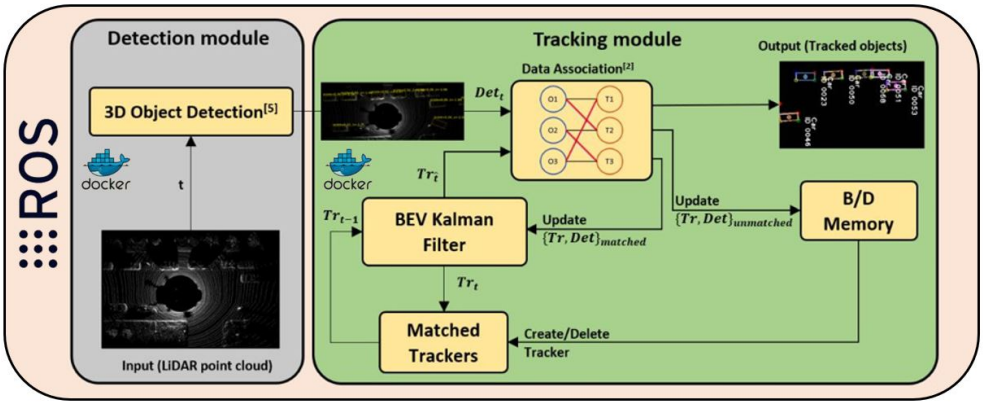

Fig. 2. **BEV MOT system pipeline**: **(1)** 3D object detection module provides the detected bounding boxes at frame t from the raw LiDAR pointcloud using ROS communications; **(2)** The coordinates are transformed into BEV image plane, so a BEV Kalman filter predicts the state of trajectories in frame *t-1* to current frame $\hat{t}$ throughout the prediction step; **(3)** the detections at frame t and predicted trajectories at $\hat{t}$ are matched using the Khun-Munkres (a.k.a Hungarian) algorithm; **(4)** matched trajectories is updated based on their corresponding matched detections to obtain update trajectories at frame t; **(5)** Unmatched trajectories and detections are used to delete disappeared trajectories or create new ones respectively; **(6)** Matched predicted trajectories are returned to the system using ROS communications.

the BEV information of the obstacle in LiDAR coordinate system is retrieved, which can be stored in static files for tracking validation (as in KITTI dataset) or published in the corresponding autonomous driving architecture using ROS facilities to provide the required information for other layers of the vehicle, such as the control or the decision-making [14]. For this purpose, we have integrated the object detector and the tracking module in two different Docker images, enhancing the portability, isolation and flexibility of the work. In this way, both stages of the MOT problem (detection and tracking) can be abstracted as two MIMO (Multiple Inputs-Multiple Outputs) systems, whose input and outputs are clearly defined in such a way both the tracking algorithm and the object detector (in addition to additional modules and requirements) can be substituted, improved or modified by similar approaches providing the same format for the inputs and outputs of both MIMOs. Our MOT system pipeline, as well as its stages, is illustrated in Fig. 2.

A. 3D Object Detection

The first step our MOT algorithm must carry out is to detect the bounding boxes of the most relevant obstacles in the environment around the vehicle. As discussed before, to avoid perspective distortion, we use PointPillars, a voxel-based state-of-the-art 3D object detector. At a given frame t, the detections provided by PointPillars are given in the following form:

$$\mathbf{Det}_t = [\mathbf{Det}_t^1, \mathbf{Det}_t^2, ..., \mathbf{Det}_t^f] \tag{1}$$

Where f is the number of detected 3D bounding boxes at a given frame and threshold. At this point, instead of using all the 3D information of the object [4] [3], we only take its projection on the floor plane (BEV information), as discussed above, to reduce the complexity and computational cost of tracking stage, specially in those urban scenarios full of vehicles, based on the assumption that height (z-dimension) is not as important as other coordinates (x-axis, y-axis) in a context of self-driving navigation. Then, each detection in eq. 1 is represented as the tuple:

$$\mathbf{Det}_t^i = [x_m, y_m, w_m, l_m, \theta, type, score] \tag{2}$$

Where x_m, y_m correspond to the object centroid in LiDAR coordinates (m), w_m and l_m correspond to the width and length of the object respectively (m), θ its orientation angle around the LiDAR Z-axis, the object type (according to KITTI format) and detection confidence. Fig. 1 shows the transformation from the source coordinate system (LiDAR), measured in m and placed at the ego-vehicle, to the target coordinate system (BEV), measured in *pixels* and placed on the top-left corner of the grid, which is the most common way to work with images in computer vision.

Eq. 3 and eq. 4 show the transformation matrix between both coordinate systems, including both the rotation and the translation, where a $\mathbf{LiDAR}_{point} = [x_m, y_m, z_m, 1]^T$ is given as the column vector in homogeneous coordinates.

$$\mathbf{T} = \begin{bmatrix} 0 & -1 & 0 & \frac{grw}{2} \\ -1 & 0 & 0 & \frac{grh}{2} \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$\mathbf{BEV}_{point} = \mathbf{T} \cdot \mathbf{LiDAR}_{point} \tag{4}$$

At this point, each detection is represented by the tuple shown in eq.2, but now x_m, y_m represent the obstacle centroid in BEV image perspective. Furthermore, the resolution of the BEV image can be modified, in such a way a width value in pixels is given to the algorithm and the height is calculated according to the aspect ratio of the real world grid with respect to the width of the image in pixels

Where gpw and gph correspond to the width and height of the BEV image perspective in pixels. To convert a point from the real word units (m) to camera units $(pixels)$, we apply the corresponding scale factor to each coordinate:

$$\begin{bmatrix} x_{px} \\ y_{pc} \end{bmatrix} = \begin{bmatrix} \frac{gpw}{grw} & 0 \\ 0 & \frac{gph}{grh} \end{bmatrix} \begin{bmatrix} x_m \\ y_m \end{bmatrix} \quad (5)$$

However, it is very common to have different scales for x and y axis in BEV since the grid is not usually squared. It is more interesting to have a further view in x LiDAR axis rather than a large side sweep in terms of y LiDAR axis. Considering this hypothesis, the right way to obtain the width and length of the BEV LiDAR bounding box in (*pixels*) is to obtain the corners of the rotated bounding box in pixels and then perform the Euclidean distance among the corresponding corners to obtain the width and length in (*pixels*). Nevertheless, the object detector provides the rotation angle of the obstacle (featured as θ) according to its own coordinate system and not around the ego-vehicle coordinate system. Regarding this constraint, to calculate the dimensions of the bounding box in pixels, three steps must be followed.

First, we assume a horizontal bounding box (θ = 0) at the BEV image coordinate system origin, where c1 corresponds to the top-left corner (c2, c3 and c4 are placed clockwise .

Then, using eq. 4 and eq. 5 for each corner, the Euclidean distance is applied between c1 and c2 to obtain the width in pixels, in the same way that the Euclidean distance is applied between c1 and c4 to obtain the length in pixels.

Finally, the first four variables of the detection tuple shown in eq. 2 are converted into pixels, in such a way the SORT algorithm deal with these bounding boxes in the BEV image by using the following tuple:

$$\mathbf{Det}_t^i = [x_{px}, y_{px}, w_{px}, l_{px}, \theta, type, score] \quad (6)$$

B. BEV Kalman Filter - Object State Prediction

Once we have each BEV detection as shown in eq. 6, a BEV Kalman Filter is used to track the objects. Since the average frame rate of PointPillars is over 50 fps, real-time can be considered at the detection module, so the inter-frame displacement of the objects can be approximated by using the constant velocity model, which is independent of other objects in the scene and of the LiDAR motion. Regarding this, the estimation of the measured variables in the following frame are:

$$x_{px}(\hat{t}) = x_{px}(t) + v_x \quad ; \quad y_{px}(\hat{t}) = y_{px}(t) + v_y$$

$$s(\hat{t}) = s(t) + v_s \quad ; \quad \theta(\hat{t}) = \theta(t) + v_\theta$$

On the other hand, due to we use the SORT algorithm to carry out the online MOT, some additional variables are included in the object state, such as the aspect ratio and the scale of the bounding box. The aspect ratio can be defined as the relation between the width and the length of the obstacle. Likewise, the scale represents the area of the target bounding box. Then, the state of each object trajectory is modelled as:

$$\mathbf{Tr}_t^j = [x_{px}, y_{px}, s, r, \theta, x_{px}^{'}, y_{px}^{'}, s^{'}, \theta^{'}] \quad (7)$$

Note that the angular velocity $\theta^{'}$ is used in the state space to improve the prediction of the obstacle in later frames. Furthermore, as shown in [15], the aspect ratio of the bounding box is considered to be constant. As observed in Fig. 1, at every frame, a tuple $\mathbf{Tr}_t = [\mathbf{Tr}_t^1, \mathbf{Tr}_t^2, ..., \mathbf{Tr}_t^g]$, with length g is calculated, where each element correspond to an association between a detection and a trajectory tracker. Then, based on the associations of the previous frame and the constant velocity model, the tuple $\mathbf{Tr}_{\hat{t}}$ is calculated, where each element corresponds to the predicted trajectory ($\mathbf{Tr}_{\hat{t}}^j$) in the current frame t expressed as:

$$\mathbf{Tr}_{\hat{t}}^j = [x_{px}(\hat{t}), y_{px}(\hat{t}), s(\hat{t}), r, \theta(\hat{t}), x_{px}^{'}, y_{px}^{'}, s^{'}, \theta^{'}] \quad (8)$$

This tuple of predicted trajectories based on the previous frame associations, in addition to the current frame detections, represents the inputs to the data association algorithm at frame t.

C. Data association

In order to associate the detections $\mathbf{Det}_t$ and the predicted trajectories $\mathbf{Tr}_{\hat{t}}$, a simple but accurate data association algorithm [2] is applied. The resulting affinity matrix presents f rows (number of detections at frame t) and g columns, which correspond to the number of predicted trajectories based on the information of frame $t-1$. Each element of the matrix corresponds to the BEV-IoU between every pair of predicted trajectory and detection. Then, we solve the bipartite graph matching problem using the Hungarian algorithm, rejecting the matching if the BEV-IoU is lower than a given hyperparameter IoU_{th}, giving rise to a set of matched detections ($\mathbf{Det}_{matched}$) and predicted trajectories ($\mathbf{Tr}_{matched}$) (both with the same number of elements, h, that is, the number of matches), as well as a set of unmatched detections ($\mathbf{Det}_{unmatched}$), where $n = f - h$ is the number of unmatched detections, and a set of unmatched trajectories ($\mathbf{Tr}_{unmatched}$), where $m = g - h$ is the number of unmatched detections.

D. BEV Kalman Filter - Object State Update

As observed in Fig. 1, once we have the corresponding sets of matched detections and trajectories, based on the Kalman Filter prediction-update cycle, we update the state space of each trajectory based on its corresponding matched detection. To do that, we use the weighted average between the matched detection values and the state space of the trajectory tracker, according to [1]. On the other hand, in the same way that [3], we appreciate that this state update step does not work properly for obstacle orientation. The reason is simple: Since the object detector is based on point cloud and no vision information is included, the object detector cannot distinguish if the obstacle is rotated 0 or π, $\frac{\pi}{2}$ and $\frac{3\pi}{2}$, and so on, around its Z-axis. That is, the orientation may differ by π in two consecutive frames. Then, if no orientation correction is applied, the Kalman Filter associated to the tracker can get easily confused, since it tries to adapts itself to the new orientation value rotating the object by π in following frames,

giving rise to a low BEV-IoU between new detections and predicted trajectories. However, regarding the assumption that obstacles must move smoothly and its orientation cannot be modified by π in one frame (0,02 s according to our object detector), when this happens the orientation of the corresponding matched detection or matched tracker can be considered wrong. To solve this problem, the detection module only considers angle from 0 to π (that is, if an angle exceeds π, it is substracted to the provided angle). Then, if the difference of orientation between a given matched detection and its corresponding matched trajectory is greater than $\frac{\pi}{2}$, as stated before, either the orientation of the detection or the orientation of the tracker is wrong. Then, we add π to the orientation of the tracker with the aim to be consistent with the matched detection.

E. Deletion and Creation of Track Identities

When obstacles leave and enter the LiDAR grid, unique identities must be destroyed or created accordingly. In most tracking algorithms it is known as the B/D (Birth and Death) Memory, which is based on the set of unmatched trackers and detections provided by the data association algorithm, where the unmatched trackers represent potential objects leaving the LiDAR grid, in the same way that unmatched detections represent potential objects entering in the analyzed environment. In order to avoid tracking of false positives (that is, clusters in the point cloud that actually do not represent a relevant obstacle, such a vehicle or a pedestrian), a new trajectory is not created until the unmatched detection has been continuously detected in the next f_{min} frames. Then, the tracker is initialised with the features of the detected bounding box, and the associated velocities set to zero. Note that, as stated in [15], since the velocity associated to the measured variables is unobserved at this moment (i.e., tracker initialization), the covariance initialises the value of the velocities (in the present work, velocity of the x_{px}, y_{px} centroid, scale s and rotation angle θ) with large values, reflecting their uncertainty. To avoid removing true positives trajectories from the scene, they are not terminated unless they are not detected during consecutive a_{max} frames. This assumption prevents an unbounded growth in the number of localisation errors and trackers due to predictions over long duration where the object detector does not provide any correction. Note that since this work does not consider object re-identification for simplicity, an object should leaves the scene and then reappers, according to the SORT algorithm, if it is initialized with a new tracker under a new identity. As shown in Fig. 1, the inputs to the Matched Trackers module are the updated matched trajectories from the BEV Kalman Filter and a set of created and deleted trackers, which jointly represent the input trajectories for the prediction step in the following frame.

IV. EXPERIMENTAL RESULTS

In order to evaluate our proposed MOT system pipeline, we carry out the evaluation in the KITTI MOT benchmark based on the method proposed by [3]. The KITTI MOT benchmark is composed of 29 testing and 21 training video sequences, where each sequence is provided with the corresponding RGB images (left and right camera of the stereo pair), LiDAR point cloud and the corresponding calibration file. Since KITTI does not provide any annotation (i.e., the groundtruth) for the testing split, we decided to evaluate our system in the training/validation split, which contains 636 and 30,601 annotated trajectories and objects respectively.

Mainstream metrics applied to MOT systems are extracted from CLEAR MOT metrics [18], such as MOTA (Multi-Object Tracking Accuracy), MOTP (Multi-Object Tracking Precision), ML/MT (Number of Mostly Lost/Tracked trajectories), IDS (Number of identity swutches), FRAG (Number of fragmentations generated by false negatives) and FN/FP (Number of false negatives/positives). Nevertheless, these metrics analyze the MOT system performance at a given threshold, not considering the confidence of the object detector. That means they do not take into account the full spectrum of precision and accuracy over different thresholds. Moreover, these traditional metrics evaluate the performance of the MOT system on the image plane (by projecting the detected 3D bounding box onto the image plane), which does not demonstrate the full strength of 3D DATMO. In that sense, the baseline [3] followed by the present work presents a 3D extension of the KITTI 2D MOT evaluation, known as KITTI-3DMOT, which focuses on the dimensions, orientation and centroid position of the 3D bounding box instead of the projection onto the image plane to evaluate the performance of the MOT system. Moreover, two new integral MOT metrics are introduced in order to solve the problem of evaluating the MOTA and MOTP of the system across all thresholds, known as AMOTA and AMOTP (Average MOTA and MOTP), as shown in eq. 9:

$$AMOTA = \frac{1}{L} \sum_{\{\frac{1}{L}, \frac{2}{L}, \ldots, 1\}} (1 - \frac{FP + FN + IDS}{num_{gt}}) \qquad (9)$$

Where L is the number of different recall values. Note that IDS, FP and FN are modified according to the results of each threshold value. Likewise, AMOTP can be estimated by integrating MOTP across all recall values.

Our final system configuration is as following: We use PointPillars trained over 1,187,840 training steps using the KITTI MOT benchmark database, the BEV Kalman Filter formulated in the previous section, an IoU_{th} = 0.25 in the data association module, and f_{min} = 1, a_{max} = 1 values for the birth and death module. We evaluate our system using the KITTI-3DMOT evaluation tool proposed by [3], obtaining the results summarized in Table I. In this table, we compare our numbers with the obtained by the representative state-of-the-art MOT system, AB3DMOT [3], with the following parameters: IoU_{th} = 0.1 in the data association module, f_{min} = 3 and a_{max} = 2, and for two different object detectors: Pointrcnn [17] and Monocular 3D [16]. Additionally, we include our previous proposal, which uses PointPillars [5] as object detector, with an IoU_{th} = 0.1 in the data association module, and f_{min} = 1, a_{max} = 3. Best results are coloured

TABLE I

ABLATION STUDY IN KITTI MOT BENCHMARK VALIDATION SPLIT USING KITTI-3DMOT. WE BOLD THE BEST RESULTS IN **BLACK** AND THE SECOND BEST IN **BLUE** FOR EACH METRIC

Method	AMOTA (%)	AMOTP (%)	MOTA (%)	MOTP (%)	IDS	FRAG	FP	FN
[3] using [16] as object detector	**28.84**	51.28	56.80	68.43	**3**	**73**	2788	7713
[3] using [17] as object detector	**39.44**	**74.60**	**76.47**	78.98	**0**	**58**	**1804**	**3859**
Previous work using [5] as object detector	27.30	63.45	59.26	75.24	131	186	4310	5364
Current approach	28.57	60.36	**73.57**	**79.77**	57	224	**3**	**2698**

in black and the second best in blue. It can be appreciated the individual effect of using different 3D object detectors as well as using different hyperparameters in terms of tracking configuration. We get the best performance in three evaluated parameters, as well as the second best results in terms of MOTA, significantly improving our previous results for all parameters. Even though we do not overcome the results obtained by AB3DMOT using [17] as object detector, these are promising results since PointPillars configurations run at least twice faster with respect to remaining configurations, dealing with the real-time requirement in terms of autonomous driving.

V. CONCLUSIONS AND FUTURE WORKS

This work illustrates the system pipeline and validation of our Real-Time Bird's Eye View MOT system based on 3D object detector on KITTI MOT benchmark using the recently proposed KITTI-3DMOT evaluation tool. The implemented system merges the concepts of ROS, that offers a robotic standard to integrate the MOT system in an easier way; Docker, in order to provide flexibility and isolation in terms of software development; and real-time DATMO, based on fast encoders for object detection and a simple but accurate and real-time algorithm to perform data prediction and association of multiple objects, such as BEV Kalman Filter and Hungarian algorithm. Our system establishes a state-of-the-art performance on 3D-MOT, as well as the possibility to be integrated in fully-autonomous driving architecture in an easier way due to its plug-and-play design, only requiring a slight fine-tuning. As future works, the system will be tested on the CARLA simulator using the corresponding groundtruth and validation tool in order to test the ability of the system to track multiple objects in arbitrarily complex urban scenarios. This strategy will wide the concept of training and validation regarding the current MOT benchmarks based on static recorded sequences, opening the possibility to build new online challenging sequences adapted to the users' needs.

ACKNOWLEDGMENT

This work has been funded in part from the Spanish MICINN/FEDER through the Techs4AgeCar project (RTI2018-099263-B-C21) and from the RoboCity2030-DIH-CM project (P2018/NMT- 4331), funded by Programas de actividades I+D (CAM) and cofunded by EU Structural Funds.

REFERENCES

[1] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[2] H. W. Kuhn and B. Yaw, "The hungarian method for the assignment problem," *Naval Res. Logist. Quart*, pp. 83–97, 1955.

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

[4] H.-k. Chiu, A. Prioletti, J. Li, and J. Bohg, "Probabilistic 3d multi-object tracking for autonomous driving," *arXiv preprint arXiv:2001.05673*, 2020.

[5] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 12697–12705, 2019.

[6] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[7] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[8] S. Schulter, P. Vernaza, W. Choi, and M. Chandraker, "Deep network flow for multi-object tracking," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6951–6960, 2017.

[9] L. Zhang, Y. Li, and R. Nevatia, "Global data association for multi-object tracking using network flows," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE, 2008.

[10] A. Patil, S. Malla, H. Gang, and Y.-T. Chen, "The h3d dataset for full-surround 3d multi-object detection and tracking in crowded urban scenes," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 9552–9557, IEEE, 2019.

[11] S. Scheidegger, J. Benjaminsson, E. Rosenberg, A. Krishnan, and K. Granström, "Mono-camera 3d multi-object tracking using deep learning detections and pmbm filtering," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 433–440, IEEE, 2018.

[12] C. R. 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*, pp. 5099–5108, 2017.

[13] 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.

[14] C. Gómez-Huelamo, L. M. Bergasa, R. Barea, E. López-Guillén, F. Arango, and P. Sánchez, "Simulating use cases for the uah autonomous electric car," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 2305–2311, IEEE, 2019.

[15] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 3464–3468, IEEE, 2016.

[16] X. Weng and K. Kitani, "Monocular 3d object detection with pseudo-lidar point cloud," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 0–0, 2019.

[17] 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*, pp. 770–779, 2019.

[18] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: the clear mot metrics," *EURASIP Journal on Image and Video Processing*, vol. 2008, pp. 1–10, 2008.