2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Dynamic Object Tracking for Self-Driving Cars Using Monocular Camera and LIDAR

Lin Zhao[1] , Meiling Wang*[1] , Sheng Su[1], Tong Liu[1] , and Yi Yang[1]

Abstract— **The detection and tracking of dynamic traffic participants (e.g., pedestrians, cars, and bicyclists) plays an important role in reliable decision-making and intelligent navigation for autonomous vehicles. However, due to the rapid movement of the target, most current vision-based tracking methods, which perform tracking in the image domain or invoke 3D information in parts of their pipeline, have real-life limitations such as lack of the ability to recover tracking after the target is lost. In this work, we overcome such limitations and propose a complete system for dynamic object tracking in 3D space that combines: (1) a 3D position tracking algorithm based on monocular camera and LIDAR for the dynamic object; (2) a re-tracking mechanism (RTM) that restore tracking when the target reappears in camera's field of view. Compared with the existing methods, each sensor in our method is capable of performing its role to preserve reliability, and further extending its functions through a novel multimodality fusion module. We perform experiments in the real-world self-driving environment and achieve a desired 10Hz update rate for real-time performance. Our quantitative and qualitative analysis shows that this system is reliable for dynamic object tracking purposes of self-driving cars.**

I. INTRODUCTION

Dynamic Object tracking is an important problem in robotics and has wide applications range from mobile robot navigation to autonomous driving. The detection and tracking of traffic participants such as pedestrians, cars, and bicyclists play an important role in decision-making and intelligent navigation for self-driving cars. Given the image of an arbitrary target of interest, the aim of object tracking is to estimate its position in all the subsequent frames with the best possible accuracy.

Although recent advances in object detection and tracking start to approach a matured state [1], [2], there are still several open problems in vision-based tracking approaches. The majority of existing methods often perform tracking in the image domain. In the absence of depth measurements or strong priors, a single view does not provide enough information to estimate the 3D layout of a scene accurately. Yet, in mobile robotics and autonomous driving scenarios, precise 3D localization and trajectory estimation is of fundamental importance. In order to prevent collisions, it is crucial to be aware of the relative position of objects in world space.

Practical applications usually face the requirements of online program execution and interaction with users. Therefore, the tracked target specified by the user can be represented as a simple rectangular bounding box in the first frame, which will result in the object representation including pixels belonging to the background. In this case, effective depth estimation algorithms are not guaranteed to be accurate when estimating the depth of the target object.

In this paper, we propose a real-time spatial position tracking system based on the combination of a monocular camera and a LIDAR sensor to realize the tracking of specified targets in actual traffic scenes. The tracking system consists of three modules: mask generation, depth estimation, and re-tracking mechanism. After given an arbitrary rectangle bounding box in the first frame of the color image sequences captured by the monocular camera, the mask generation module tracks the selected object and generates a binary segmentation mask that expresses whether or not a pixel belongs to the target during the tracking process. The generated mask provides pixel-level information for the depth estimation module. Then based on the results of the joint calibration between monocular camera and LIDAR, we project the LIDAR point cloud onto the image plane to obtain the sparse two-dimensional depth map. According to the segmentation results of the target, the spatial position of the target is estimated from the sparse depth map. Due to the rapid movement, the target may leave the camera's field of view. In this paper, we develop a re-tracking mechanism that maintains an object detection program and restores tracking when the target reappears.

This tracking system is further developed based on the existing excellent end-to-end target tracking networks. We focused on the 3D position tracking of a single target and re-tracking problem. In summary, our main contributions in this paper are threefold:

1) A re-tracking mechanism (RTM) is developed to restore tracking after losing the tracked object. When the object returns to the camera's field of view, RTM can quickly re-identify the original object and continue tracking.
2) A strategy to fuse monocular camera data and LIDAR data is developed. Using the geometry information provided by LIDAR, the spatial position of the target can be estimated.
3) We conduct a qualitative and quantitative evaluation of the system on our self-driving car platform.

*This work was partly supported by Program for Changjiang Scholars and Innovative Research Team in University of Ministry of Education of China in 2016 (IRT_16R06, T2014224), General Program of National Natural Science Foundation of China (Grant No. NSFC 61173076), Training Program of the Major Research Plan of the National Natural Science Foundation of China (Grant No. NSFC 91120003). (Corresponding author: Meiling Wang.)

[1]The authors are with the Integrated Navigation and Intelligent Navigation laboratory, School of Automation, Beijing Institute of Technology, China (email: zhaolin_email@qq.com).

II. RELATED WORK

In this section, we provide an overview of the most representative techniques for the problems tackled in this paper.

A. *Object Tracking*

In general, tracking algorithms can be categorized into generative or discriminative based on their model-construction mechanism [3]. Generative methods [4], [5] model the association problem as a certain form of optimization problem on graphs, which model the target region in the current frame, and look for the region most similar to the model in the next frame to predict the location. Discriminative methods [6], [7] are also called tracking-by-detection, where the target object is first obtained by an object detector and then linked into trajectories via data association.

Tracking-by-detection has been the prevailing solution to object tracking in the past decade. Especially, the Correlation Filter rose to prominence as a particularly fast and effective strategy for tracking-by-detection [8]. Performance of Correlation Filter-based methods has then been notably improved with the adoption of spatial constraints [9], multi-channel formulations [10], and deep features [11], [12].

B. *Semantic Segmentation*

Most modern object tracking frameworks use a rectangular bounding box both to initialize the target and to estimate its position in the subsequent frames. However, a simple rectangle often fails to properly represent an object, this can lead to incorrectly estimate the depth of the object. This motivated us to produce segmentation masks during the tracking progress while still only relying on an initial bounding box.

The goal of semantic segmentation is to classify each pixel of an image into a predefined class. A popular image segmentation model is the encoder-decoder structure. The encoder part reduces the spatial resolution of the input through downsampling to generate a low-resolution feature map, which has high computational efficiency and can effectively distinguish different categories. The decoder samples the feature description up and restores it to the full resolution segmentation graph. Similar to the object detection task, there is a trade-off between accuracy and runtime. Therefore, approaches like convolutional factorization [13], [14], quantization [15], pruning [16], and dilated convolutions [17] came up.

C. *Depth Estimation*

Depth estimation is a part of 3D reconstruction in computer vision. The monocular depth estimation based on deep learning is to fit a function that maps the image into a depth map. However, there may be far fewer pixels in the object than in the background, so the effective depth estimation algorithms cannot ensure that the depth estimation of the target object is accurate. A traditional usage of LIDAR point cloud for tracking is to measure distances [18], provide 2.5D grid representation [19], [20] or to derive some hand-crafted features [21]. Recent studies [22], [23] have demonstrated the value of using 3D point cloud as perception features in autonomous driving. There are also methods [24] which projects the point cloud to a sphere thus 2D CNN can be applied for the segmentation task.

III. MASK GENERATION

The first step of mask generation is to select the Region of Interest (ROI) where the target is located in the first frame. To allow simple initialization, the ROI object is only represented by a bounding box. The visual single object tracking is often considered as a similarity learning problem, which is usually addressed by Siamese architectures [25], [26], [27]. Bertinetto et al. [25] train a fully-convolutional Siamese network, which is referred to as SiamFC, on the offline datasets. Their approach achieves very competitive performance in modern tracking benchmarks at speeds that far exceed the frame-rate requirements. Inspired by the success of the fast-tracking approach, we similarly employ a fully-convolutional Siamese neural network within the mask generation module to obtain object tracking information in real-time.

A. *Network Architecture*

The fully-convolutional Siamese network requires inputting an exemplar image X_1 and a search image X_2, which are the ROI object we selected and a larger search area, respectively. The Siamese network applies an identical transformation to both inputs, which means that the exemplar image and the search image are computed by the same function $F_\theta(\cdot)$, generating two feature maps $F_\theta(X_1)$ and $F_\theta(X_2)$. After that, the two features embeddings $F_\theta(X_1)$ and $F_\theta(X_2)$ are combined with a cross-correlation layer to generate a response map:

$$C_\theta(X_1, X_2) = F_\theta(X_1) \star F_\theta(X_2) \tag{1}$$

where $\star$ denotes cross-correlation operation, θ indicates the parameters of the Siamese network, and Resnet-50 [28] is employed to the identity transformation of shared weights in this paper.The response map $C_\theta(X_1, X_2)$ is actually a score map, and each spatial element on it encodes the similarity measure between X_1 and all candidate sub-windows within X_2.Therefore, the most responsive part of $C_\theta(X_1, X_2)$ corresponds to the location of the target in the search image. We can track the position of the most similar part on the search image X_2 against the sample image X_1 based on the response map.

However, the response map only indicates position information on the image plane. It does not provide pixel-level semantic information of the target object. Based on the work of SiamFC, Wang Qiang et al. [1] propose a multi-task learning method that combines video object segmentation and visual object tracking by extending a branch for training a segmentation task, which is called SiamMask. The Siamese network is simultaneously trained in siammask for three tasks: the corresponding bounding box, object score,

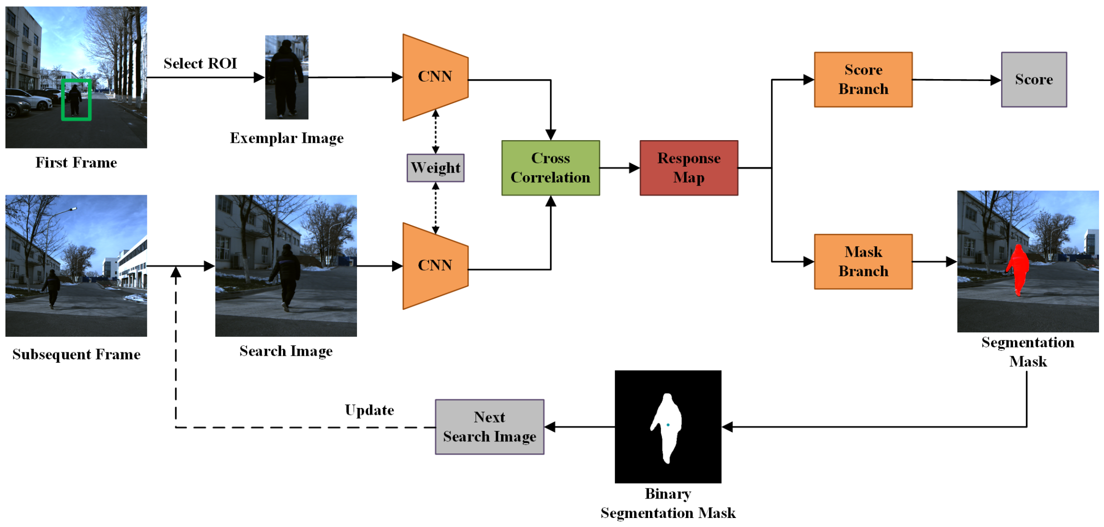

Fig. 1. The mask generation and object tracking network architecture.

and segmentation mask of the tracked target object. Since the basic task of mask generation is to obtain the pixel-level information of the tracked object, its bounding box is not utilized in the depth estimation module. Our method only adopts the two-branch variant of SiamMask, that is, two branch networks are added after the response map $C_\theta(X_1, X_2)$, which are used to predict the object score and segmentation mask respectively. The score branch outputs the score of tracking results, which can be used to judge whether the target is lost. The mask branch is used to output a binary segmentation mask, which is used to express whether or not a pixel belongs to the target object within the search image. Both the binary segmentation mask and the object score are implemented by a two-layer neural network:

$$\begin{aligned} O_{mask} &= M_\phi(C_\theta(X_1, X_2)) \\ O_{score} &= S_w(C_\theta(X_1, X_2)) \end{aligned} \tag{2}$$

where M_ϕ denotes the two-layers CNN for producing segmentation masks with learnable parameter ϕ, S_w denotes the two-layers CNN used to output scores with learnable parameter w.

The architecture of the mask generation algorithm is illustrated in Fig. 1. After obtaining the segmentation mask, the center position of the search image X_2 in the next frame can be inferred and updated. The update strategy takes the last estimated position of the target as the center of the search image X2, that is, the center point o_i of the binary segmentation mask.

B. Loss Function

The Siamese network in SiamFC is trained on positive and negative pairs extracted from a large-scale video dataset with annotations. Each pair of training samples consists of an exemplar image and a corresponding search image extracted from the same video. A logistic loss is adopted in this training process, which can be referred to as L_s:

$$L_s = \log(1 + e^{-sy}) \tag{3}$$

where each pair is labeled with a ground-truth label $y \in \{+1, -1\}$ and s is its real-valued score. In the case of positive pairs($y = 1$) in SiamMask, a binary logistic regression loss L_m is adopted for each ground-truth mask c of size $w \times h$:

$$L_m = \tfrac{1+y}{2wh} \textstyle\sum_{ij} \log\left(1 + e^{-c^{ij} m^{ij}}\right) \tag{4}$$

where m is the predicted mask and both the superscript (i, j) denotes the pixel coordinate of the corresponding mask. In this way, the two-branch Siamese network optimizes a two-branch losses $L_{two-branch}$:

$$L_{two-branch} = \lambda_1 L_m + \lambda_1 L_s \tag{5}$$

where hyperparameters λ_1 and λ_2 are set to 32 and 1 respectively, according to the configurations in SiamMask.

IV. DEPTH ESTIMATION

The output of the previous section is a binary segmentation mask of the search image. In this section, we describe how to estimate the spatial position of the target object with respect to the self-driving car from the binary segmentation mask and the LIDAR point clouds. To describe the framework combining LIDAR and Monocular camera, the coordinate frames of the self-driving car is first introduced, namely the LIDAR frame L and the camera frame C(see Fig. 2). For convenience, the LIDAR coordinate system can be seen as the car coordinate system, that is, the origin of the LIDAR coordinate system represents the position of the vehicle.The three-dimensional position of the target object in the LIDAR frame is represented by $P_o = (x_o, y_o, z_o)$.

Before the data fusion between LIDAR and camera, it is necessary to know the transformation matrix between the corresponding coordinate systems of both sensors to provide redundant information in the same reference system. Dhall et al. [30] propose a novel LIDAR-camera extrinsic calibration solution by using 3D-3D point correspondences. Their approach uses ArUco tags stuck on the cardboards with a point extraction algorithm to obtain 3D point correspondences in the LIDAR as well as camera frames. The two sets of 3D points can be used to find an accurate rigid-body transformation between LIDAR frame L and camera frame C. With the above calibration pipeline, the LIDAR point clouds can be projected into the camera frame:

$$\mathbf{p}_C = \mathbf{R}_{CL}\mathbf{p}_L + \mathbf{t}_{CL} \tag{6}$$

where $\mathbf{p}_L = [x_L, y_L, z_L]$ denotes the 3D coordinates of the point clouds in the LIDAR frame, and $\mathbf{p}_C = [x_C, y_C, z_C]$ denotes the 3D coordinates in the camera frame. According to the pinhole camera model [29], there is a mathematical relationship between the coordinates of a point in the three-dimensional space and its projection onto the image plane:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{z_C}\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x_C \\ y_C \\ z_C \end{bmatrix} = \frac{1}{z_C}\mathbf{K}\mathbf{p}_C \tag{7}$$

where $\mathbf{K}$ indicates the camera intrinsic matrix. $\mathbf{p}_i = (u, v)$ indicates the coordinates of a point on the image plane. From Eq 7, LIDAR point clouds are projected onto the image plane. Since the segmentation mask of the target object is known, namely we know which pixels on the image plane whether or not belong to the target. The problem of 3D position estimation can be simplified to calculate the depth of the target object z_{obj} in the camera coordinate system. In an ideal situation, we can directly calculate the 3D position of the target object based on the LIDAR point clouds that overlap with the segmentation mask on the image plane.

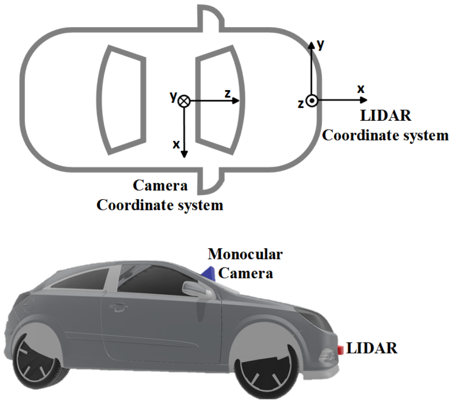

Fig. 2. The coordinate systems of the self-driving car.

However, due to the sparsity of LIDAR point clouds, when the distance between the dynamic target and the self-driving car is relatively large, the number of LIDAR point clouds that overlap with the segmentation mask on the image plane will be small. Besides, there are cases of noise in LIDAR point clouds or rough segmentation mask, etc. Considering the above problems, a choice strategy is adopted to obtain an accurate three-dimension position of the target object. After the LIDAR point clouds are projected onto the image plane, we take the following steps for each point $\mathbf{p}_{uv}$:

1) Choose the projected LIDAR points in the search image as a set $\mathbb{F}$. Based on whether or not the points are inside the object area (that is, the area with a pixel value of one in the binary segmentation mask), we divide the set $\mathbb{F}$ into a set $\mathbb{F}_b$ and a set $\mathbb{F}_o$, where $\mathbb{F}_o$ indicates the projected LIDAR points located inside the object area, and $\mathbb{F}_b$ indicates the points located outside the object area, namely the background area.
2) Count the number of the projected points in the point set $\mathbb{F}_o$ and record it as Γ. In the case where the number of points Γ is greater than a given threshold. Taking the center point of the object area as the origin o_i (see Section III-A), the pixel distances d_p between the projected points of $\mathbb{F}_o$ and o_i are calculated in order. Then the depth of the target object z_{obj} can be calculated:

$$z_{obj} = \frac{\sum_{i\in\Gamma} \frac{1}{d_p^{(i)}} z_C^{(i)}}{\sum_{i\in\Gamma} \frac{1}{d_p^{(i)}}} \tag{8}$$

where $d_p^{(i)}$ indicates the pixel distance between the i-th projected point of $\mathbb{F}_o$ and o_i, $z_C^{(i)}$ indicates the coordinate value of the i-th projected point in the z-axis direction of the camera frame C. $z_C^{(i)}$ indicates the depth of the i-th projected point in the camera frame C.

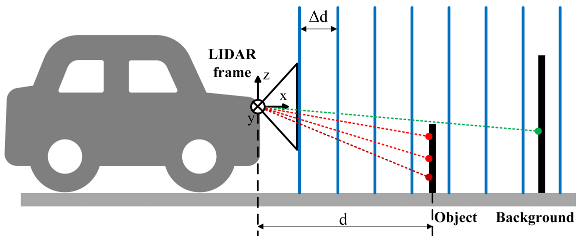

Fig. 3. The depth histogram for projected points

3) In the case where the number of points Γ is less than the given threshold, we calculate a depth histogram of the projected points in the set $\mathbb{F}_o$ with a fixed bin width of $\Delta d = 0.2m$ (see Fig. 3). Elements in $\mathbb{F}_o$ are inserted into the depth histogram. The maximum probability interval in the depth histogram is considered to be the vertical plane in which the object is located.

For a self-driving car, the object tracking is also a moving object on the ground. According to the operating principle of point clouds captured by LIDAR scanning, the point cloud scanned onto the object can be approximated as a local vertical plane. We consider the connection of points within the object area and its neighborhood. Points inside the object area and points in the neighborhood of the object area will fall in different intervals of the depth histogram. A jump in depth from the object to background corresponds to a gap in bin occupancy. According to this constraint, the interval of the object's local vertical plane can be estimated. If the number of projected points in the object area is less than the given threshold and the above constraint relationship does not exist. In this case, the object is considered to be lost.

V. RE-TRACKING MECHANISM

Based on the score results generated by the mask generation module and the state given by the depth estimation module, we can decide whether to convert the tracked state to the state of loss. In the case of a lost state, the self-driving car will follow its original planned trajectory.

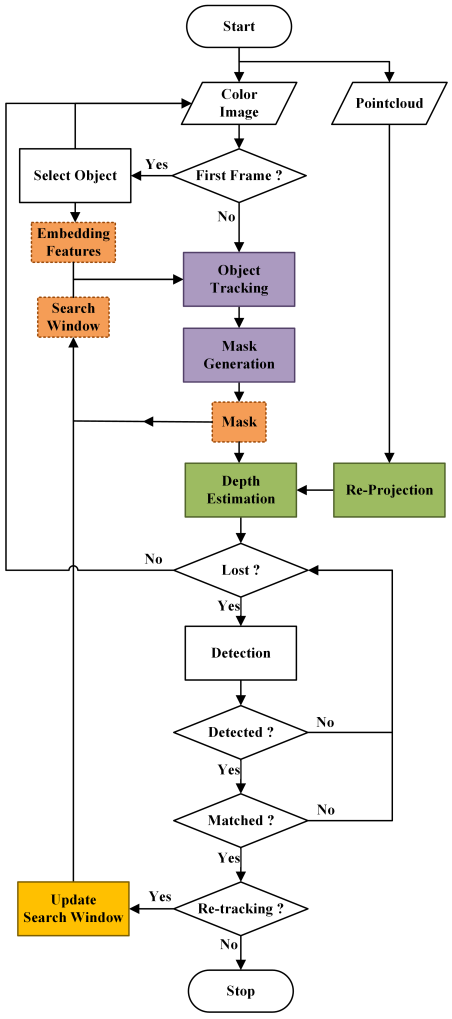

Fig. 4. The whole process of RTM.

Generally, most target loss occurs when the target makes a quick turn and disappears from the camera's field of view. When the lost target returns to the camera view, RTM can deal with the tracking recovery problem. The workflow of RTM is shown in Fig. 4. The universal fast object detection framework YOLO [2] is used to output a rough target detection on the input image. Since the LIDAR sensor has a wide 360-degree surround scan range, the lost target will still be scanned in the field of view of the LIDAR. When target loss begins to occur, RTM records the target category selected in the first frame (e.g. pedestrians, vehicles) and the direction angle θ_l of the target with respect to the self-driving car. RTM keeps running the YOLO detection algorithm according to the recorded category to detect whether the specified object category appears again in subsequent images.

When the target of the specified category is detected again, RTM calculates a target azimuth θ_r in the LIDAR data according to the depth estimation module and compares it with the azimuth θ_l of the target when it is lost to determine whether the re-detected target is in the vicinity. Besides, RTM crops a temporary search image according to the position of the target in the current image and re-inputs it into the fully convolutional Siamese network. In order to reduce computing resources, RTM only runs the score-branch network. The output score of the network measures the similarity between the previous exemplar image and the temporary search image. Given a certain score threshold and neighboring azimuth threshold (e.g. $|\theta_l - \theta_r| < \Delta\theta$), RTM determines whether the previously tracked object reappears. If the re-detected target satisfies the conditions of similarity score judgment and adjacent azimuth judgment at the same time, RTM resumes tracking and updates the search window based on the bounding box of the detected object. When the next image frame appears, the system continues to track the specified target based on the updated search window.

VI. EXPERIMENTS

In this section, we present the experimental evaluation of our 3D position tracking system for dynamic objects.

A. Experimental Setup

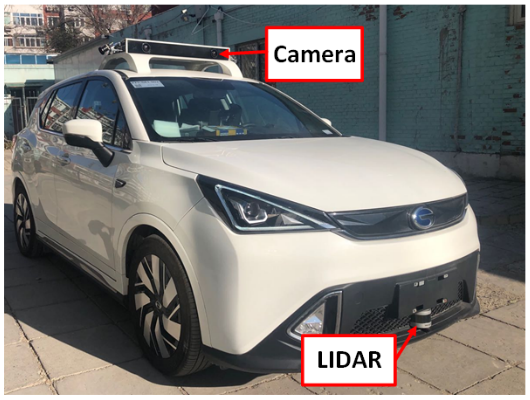

Fig. 5. Our self-driving car platform, equipped with monocular cameras and a LIDAR sensor.

Unlike other object tracking experiments based on offline annotated datasets, we evaluate the performance of the 3D position tracking system through our self-driving car platform in real-world driving scenarios. The self-driving car platform as shown in Fig. 5, is equipped with monocular cameras and a LIDAR sensor that is the Velodyne's VLP-16. The Velodyne VLP-16 LIDAR sensor has a 30° vertical field of view, as well as a 360° horizontal field of view, and the 16 planes represent an angular resolution of 2° in the vertical direction. The monocular camera is installed in the front of the car, which has a 135° field of view. We only use a monocular camera to deploy our tracking algorithm as well as a VLP-16 which is also installed in the front of the car. The monocular camera is calibrated with respect to the LIDAR frame using a LIDAR-Camera calibration pipeline [30] to find a rigid-body transformation between a LIDAR and a camera for autonomous vehicles applications. We integrated the entire three-dimension position tracking system of dynamic objects into the self-driving car. All the software runs on a PC with an NVIDIA GeForce GTX 2080.

B. *Re-Tracking Experiment*

To conduct dynamic object tracking experiments in the case where the tracked object is lost, we artificially design a driving environment with a 90-degree turn path as shown in Fig. 6. During the 90-degree path turning process, the self-driving car is prone to encounter the blind area at close range due to the limited field of view of the monocular camera. At this point, the tracked object is easy to move out of the camera's field of view. Although the object cannot be detected, there is a heading hold function on the self-driving car that allows it to keep turning based on its planned historical trajectory. When the self-driving car enters the straight area after the turn is completed, the target will return to the camera's field of view. We deploy advanced visual tracking algorithms such as Deep SORT [31], [32] or SiamMask [1] on our self-driving car to compare whether these algorithms can restore tracking under the same experimental conditions. Table I shows the results of the re-tracking experiment. It can be seen that our algorithm can continue to re-track the target after the target is lost ("Changed" indicates that the ID of the target re-tracked is different from the original target). The experimental results of missing the target in a turn show that RTM is an effective object re-tracking strategy for the target loss problem during driving with blind areas on a curve.

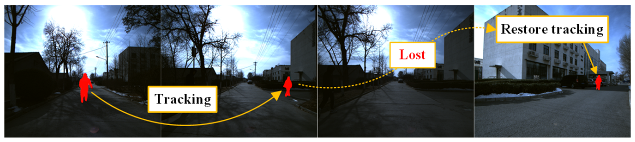

Fig. 6. Re-tracking experiments

TABLE I
RE-TRACKING RESULTS

Method	Deep SORT	SiamMask	**RTM**
Status	Changed	Lost	**Re-Tracking**

C. *Runtime Performance*

We create multiple parallel threads to achieve a desired 10-Hz update rate for real-time performance: mask generation thread, depth estimation thread, and detection thread for RTM. The detection thread and mask generation thread are mutually exclusive. To manage intra-process communication between different threads and sensors, the Robot Operating System (ROS) was used. ROS provides already-implemented sensor drivers for extracting point cloud data and color image data from the VLP-16 and monocular camera, respectively. The calculation time of different algorithm modules in our system and the capture time of different sensor data is shown in Fig. 7. Since the LIDAR data capture is run by a separate program thread, the total time taken to track the 3D position of the target is 83.3ms + 15.2ms = 98.5ms on average under the condition that the target is not lost.

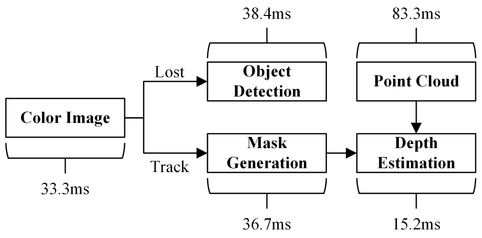

Fig. 7. Process time for each step

D. *Qualitative and Quantitative Results*

Qualitative results of the 3D position tracking system for dynamic objects are shown in Fig. 9. The intermediate results are listed in detail according to the processing logic of the system. We validate the adaptability of our system by tracking different objects in different environments. The maximum speed of the self-driving car does not exceed 10km/h during the experiments.

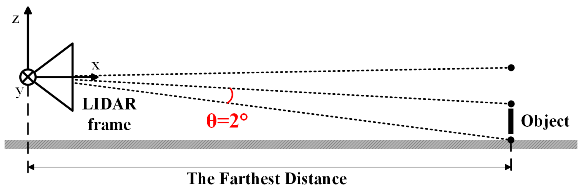

Fig. 8. The farthest tracking distance derived from the resolution of LIDAR, $\theta = 2°$ denotes the angular resolution of LIDAR in the vertical direction.

Besides, we performed the farthest distance experiments of the tracking algorithm. Ideally, the farthest distance of the three-dimensional position tracking algorithm is constrained by the image resolution and the angular resolution of the LIDAR in the vertical direction. When the target is too far away from the vehicle, the target occupies fewer pixels in the image plane and is difficult to track. Moreover, the interval of the lidar's scan lines increases with distance. When the interval is larger than the space occupied by the target, the target cannot be scanned. With the resolution of the camera allowed, we calculate the farthest distance of the 3D position tracking algorithm. As shown in Fig. 8, we approximate the farthest distance of our tracking algorithm when the height of the ground object is equal to the interval between scan lines of the LIDAR:

$$d_{max} = \frac{h}{tan(\frac{\theta}{2})} \tag{9}$$

where h indicates the height of the target. For a pedestrian with a height of $h = 1.7$m, The farthest distance of our tracking system reaches 39.23m in real-world driving experiments.

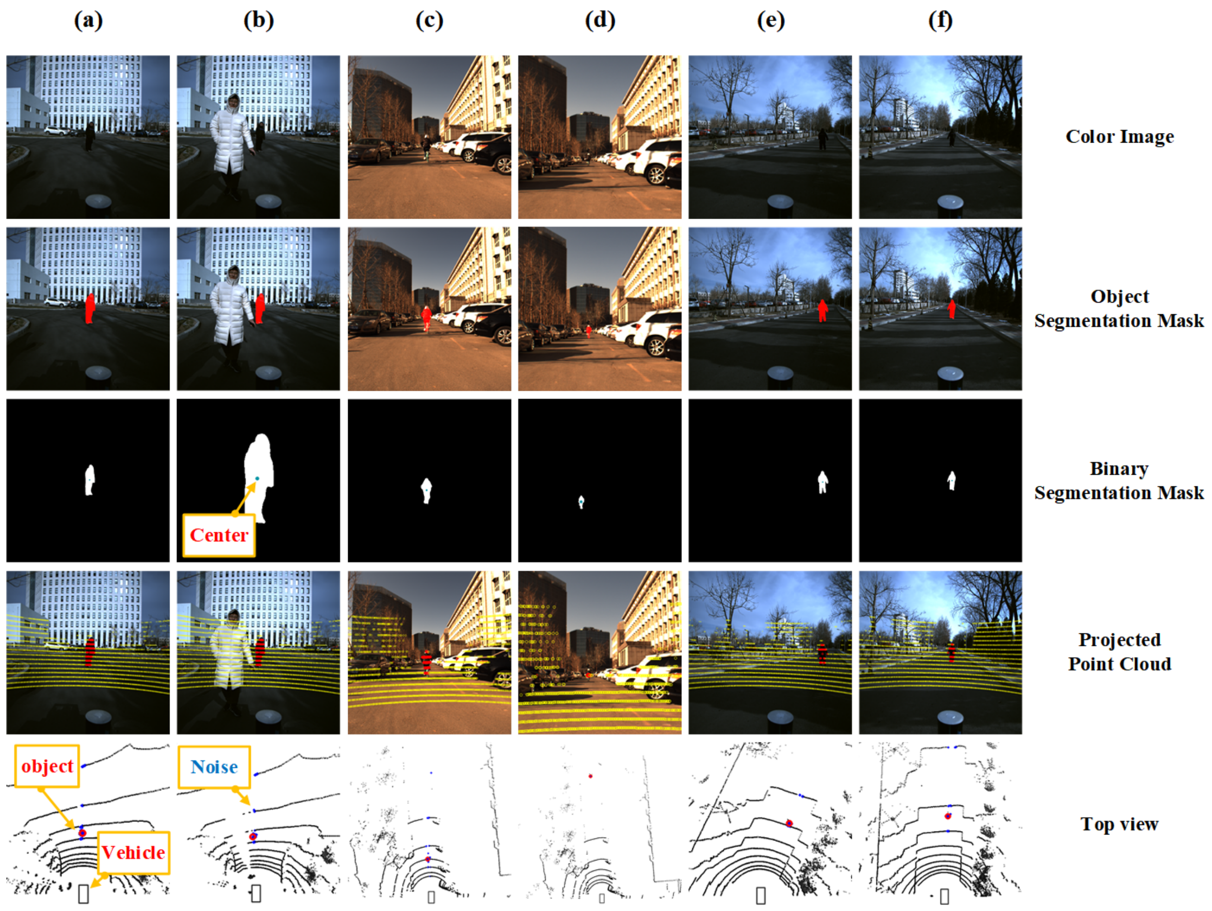

Fig. 9. Qualitative results of our system. All experimental scenario data is derived from our self-driving car platform. (a) and (b) test the performance of object tracking when obstacles appear in the search image. (c) and (d) test tracking a pedestrian riding bicycles. (e) and (f) test dynamic tracking performance. The bottom row is a point cloud map showing the top view. We project the 3D position of the target onto a 2D plane in the bird's-eye view to assist the decision-making planning of the self-driving car. We use the red circle to indicate the position of the target in the two-dimensional top-view map constructed by the point cloud. Note that the blue points in the map represent the noise, which will be filtered by our proposed position estimation algorithm during the calculation of the three-dimensional position of the target.

VII. CONCLUSION AND FUTURE WORK

In this paper, we propose a 3D position tracking pipeline for self-driving cars. In particular, by fusing a LIDAR with a monocular camera, our system achieves the goals of tracking the target in images and estimating its depth. Furthermore, we propose a re-tracking mechanism (RTM) to resume tracking after the object is lost.

Despite the promising real-world vehicle experimental results, our system still has limitations in the case of simultaneous tracking of multiple objects. As future work, we would like to consider more practical issues related to dynamic object tracking to improve the performance of self-driving cars.

REFERENCES

[1] Wang, Qiang, et al. "Fast online object tracking and segmentation: A unifying approach." Proceedings of the IEEE conference on computer vision and pattern recognition. 2019.
[2] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
[3] Li, Xi, et al. "A survey of appearance models in visual object tracking." ACM transactions on Intelligent Systems and Technology (TIST) 4.4 (2013): 1-48.
[4] Wen, Longyin, et al. "Multiple target tracking based on undirected hierarchical relation hypergraph." Proceedings of the IEEE conference on computer vision and pattern recognition. 2014.
[5] Kim, Chanho, et al. "Multiple hypothesis tracking revisited." Proceedings of the IEEE International Conference on Computer Vision. 2015.
[6] Bolme, David S., et al. "Visual object tracking using adaptive correlation filters." 2010 IEEE computer society conference on computer vision and pattern recognition. IEEE, 2010.
[7] Henriques, João F., et al. "High-speed tracking with kernelized correlation filters." IEEE transactions on pattern analysis and machine intelligence 37.3 (2014): 583-596.
[8] Bolme, David S., et al. "Visual object tracking using adaptive correlation filters." 2010 IEEE computer society conference on computer vision and pattern recognition. IEEE, 2010.
[9] Kiani Galoogahi, Hamed, Terence Sim, and Simon Lucey. "Multi-channel correlation filters." Proceedings of the IEEE international conference on computer vision. 2013.
[10] Kiani Galoogahi, Hamed, Terence Sim, and Simon Lucey. "Correlation filters with limited boundaries." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.
[11] Danelljan, Martin, et al. "Eco: Efficient convolution operators for tracking." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.
[12] Valmadre, Jack, et al. "End-to-end representation learning for corre-

lation filter based tracking." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017.

[13] Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).

[14] Mehta, Sachin, et al. "Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation." Proceedings of the european conference on computer vision (ECCV). 2018.

[15] Rastegari, Mohammad, et al. "Xnor-net: Imagenet classification using binary convolutional neural networks." European conference on computer vision. Springer, Cham, 2016.

[16] He, Yihui, Xiangyu Zhang, and Jian Sun. "Channel pruning for accelerating very deep neural networks." Proceedings of the IEEE International Conference on Computer Vision. 2017.

[17] Yu, Fisher, and Vladlen Koltun. "Multi-scale context aggregation by dilated convolutions." arXiv preprint arXiv:1511.07122 (2015).

[18] Rangesh, Akshay, and Mohan Manubhai Trivedi. "No blind spots: Full-surround multi-object tracking for autonomous vehicles using cameras and lidars." IEEE Transactions on Intelligent Vehicles 4.4 (2019): 588-599.

[19] Asvadi, Alireza, Paulo Peixoto, and Urbano Nunes. "Detection and tracking of moving objects using 2.5 d motion grids." 2015 IEEE 18th International Conference on Intelligent Transportation Systems. IEEE, 2015.

[20] Choi, Jaebum, et al. "Multi-target tracking using a 3d-lidar sensor for autonomous vehicles." 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013). IEEE, 2013.

[21] Song, Shiyang, Zhiyu Xiang, and Jilin Liu. "Object tracking with 3D LIDAR via multi-task sparse learning." 2015 IEEE International Conference on Mechatronics and Automation (ICMA). IEEE, 2015.

[22] Bai, Min, et al. "Deep multi-sensor lane detection." 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2018.

[23] Casas, Sergio, Wenjie Luo, and Raquel Urtasun. "Intentnet: Learning to predict intention from raw sensor data." Conference on Robot Learning. 2018.

[24] Wu, Bichen, et al. "Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud." 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018.

[25] Bertinetto, Luca, et al. "Fully-convolutional siamese networks for object tracking." European conference on computer vision. Springer, Cham, 2016.

[26] Zagoruyko, Sergey, and Nikos Komodakis. "Learning to compare image patches via convolutional neural networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[27] Li, Bo, et al. "High performance visual tracking with siamese region proposal network." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018.

[28] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[29] Hartley, Richard, and Andrew Zisserman. Multiple view geometry in computer vision. Cambridge university press, 2003.

[30] Dhall, Ankit, et al. "LiDAR-camera calibration using 3D-3D point correspondences." arXiv preprint arXiv:1705.09785 (2017).

[31] Wojke, Nicolai, and Alex Bewley. "Deep cosine metric learning for person re-identification." 2018 IEEE winter conference on applications of computer vision (WACV). IEEE, 2018.

[32] Wojke, Nicolai, Alex Bewley, and Dietrich Paulus. "Simple online and realtime tracking with a deep association metric." 2017 IEEE international conference on image processing (ICIP). IEEE, 2017.