Edge-assisted Multi-vehicle Cooperative Perception: an Approach Based on Relative Pose Estimation

Journal:	*Transactions on Mobile Computing*
Manuscript ID	TMC-2023-10-1207
Manuscript Type:	Regular
Keywords:	Edge Computing, Cooperative Perception, Pose Estimation, I.4.3.d Registration < I.4.3 Enhancement < I.4 Image Processing and Computer Vision < I Computing Methodologies

Edge-assisted Multi-vehicle Cooperative Perception: an Approach Based on Relative Pose Estimation

Yu Zhao, He Sun, *Student Member, IEEE,* Mingjun Xiao, *Member, IEEE,*
Jie Wu, *Fellow, IEEE,* Junjie Shao, and Jinbo Cai

Abstract—Cooperative perception is a crucial technology to improve the safety and robustness of autonomous driving systems. Existing works mainly utilize a pre-constructed 3D map or roadside infrastructures to estimate poses for cooperative perception. However, the pre-constructed map should be updated frequently, and roadside infrastructures can only be employed on specific roads. To overcome these limitations, we exploit the inherent features of external shapes of vehicles for relative pose estimation and design an Edge-assisted multi-Vehicle cOperative perception system, called EVO. EVO adopts a suit of approaches to ensure the accuracy and real-time performance of cooperative perception: (1) a novel approach is proposed to estimate relative poses without requiring any pre-constructed 3D map or roadside infrastructures; (2) a hierarchical registration module is adopted to speed up pose estimation; (3) a Regions of Interest (RoIs) prediction module based on Dlinear and a registrations reduction module are designed to reduce the computation latency. We evaluate EVO on a CARLA-based dataset and demonstrate that EVO can perform multi-vehicle cooperative perception in real-time and maintain centimeter-level accuracy in pose estimation. Further, EVO extends the maximum perception distance from 32-39m to 38-67m and achieves about 2.43× improvement in perception area.

Index Terms—Cooperative Perception, Pose Estimation, Edge Computing, Registration

1 Introduction

With the development of computer vision, artificial intelligence, sensors, and network communication technologies, autonomous driving is gradually entering people's lives [1], [2]. Currently, high-level autonomous driving can be achieved in specific scenarios, such as in closed ports and industrial parks, while consumer-grade cars remain at the level of assisted driving. Widely reported accidents show that consumer-grade self-driving cars are unreliable, and their safety needs to be improved urgently. Multi-vehicle cooperative perception, which enables to sense distant or obscured objects in advance, is an effective way for vehicles to make correct decisions and improve the safety of autonomous driving [3], [4].

In general, cooperative perception consists of three major components. First, the poses of vehicles need to be determined. Specifically, cooperative perception requires a transformation matrix, which indicates the changes of position and orientation between two vehicles. Second, the vehicle receives the transformation matrices and perception data (e.g., point clouds, images, etc.) from others in real-time by using the wireless transmission between vehicles directly or via edge server indirectly. Third, the vehicle aggregates the received data into a complete view through fusion, as shown in Fig. 1. Currently, numerous efforts have investigated the cooperative perception among multiple vehicles and roadside infrastructures. For simplicity, we divide these works into the following two categories.

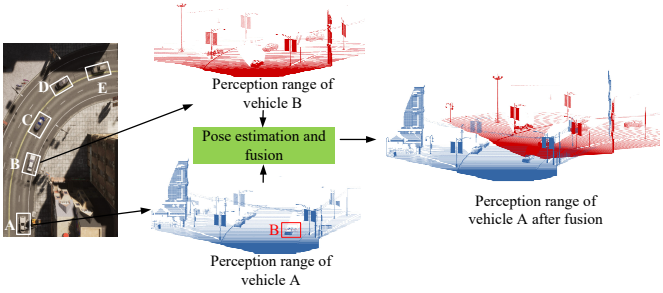


Fig. 1. We simulate real road scenes in the CARLA simulator, equipping each vehicle with a RGB-D camera or LiDAR with a perception distance of 40m and a field of view (FOV) of 110 degrees. Due to line-of-sight occlusion and limited perception distance, vehicle A cannot perceive vehicles C, D, and E. By fusing the point clouds perceived by vehicle A and vehicle B, vehicle A can observe vehicles C,D, and E in advance.

First, some works generally assume that the poses of vehicles together with sensors are known in advance [5]–[8], or identified by a 3D map [9]–[11] that is constructed through Simultaneous Localization and Mapping (SLAM) [12]–[14]. Consequently, they pay more attention to transmission or

- *Y.Zhao, H.Sun, M.Xiao(Corresponding author), J.Shao, and J.Cai are with the School of Computer Science and Technology / Suzhou Institute for Advanced Research / State Key Laboratory of Cognitive Intelligence, University of Science and Technology of China (USTC), Hefei, China. E-mail: {zhaoyu0624@mail, hesun@mail, xiaomj@, fdssjj@mail, SA21011121@mail}.ustc.edu.cn.*
- *J. Wu is with the Center for Networked Computing, Temple University, Philadelphia, PA 19122, USA. E-mail: jiewu@temple.edu.*

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grants 62172386, 61872330, and 61572457, in part by the Natural Science Foundation of Jiangsu Province in China under Grants BK20231212, BK20191194, and BK20131174, and in part by the Teaching Research Project of the Education Department of Anhui Province in China (No. 2021jyxm1738).

fusion issues while ignoring pose estimation. For example, EMP focuses on how to partition multi-vehicle point clouds from LiDAR based on the locations of vehicles and network bandwidth [5]. AutoCast proposes an infrastructure-free approach to maximize information sharing through wireless channels, where SLAM is employed to estimate poses [10]. However, in the real world, the pose from the Global Navigation Satellite System (GNSS) has low accuracy in urban environments due to signal occlusion and multipath affections, which will greatly degrade the accuracy of point clouds fusion. The pose estimation based on SLAM needs to construct 3D maps in advance and update them frequently, which will result in expensive costs.

Second, another line of work estimates the poses of vehicles based on the data perceived by roadside infrastructure [15]–[19]. Roadside infrastructure can observe vehicles from different perspectives, so fusing the perceptions from roadside infrastructures and connected vehicles can effectively complement the vehicles' view. For instance, VI-Eye performs pose estimation by aligning the vehicle-perceived point clouds with the infrastructure-perceived point clouds to obtain the transformation matrix [16]. These methods can work well with the assistance of roadside infrastructures, but they cannot cope with the absence of roadside infrastructures. While roadside infrastructure is gradually playing a more important role in urban mobility, its popularity will take time.

Motivated by the challenges encountered by 3D maps and roadside infrastructure, we design an Edge-assisted multi-Vehicle cOoperative perception system, called EVO, which does not need the assistance of roadside infrastructures and the 3D map pre-constructed by the SLAM approach. In EVO, we propose a novel 6 degrees of freedom (DoF) pose estimation method. Unlike existing works, we construct point clouds of the external shape of each vehicle, called a prior *point-cloud model*. Then, key points are extracted from each vehicle's point-cloud model as the prior knowledge for pose estimation. Because the external shape of each vehicle remains unchanged, the features of these key points are relatively stable. By performing the registrations of these key points with the real-time point clouds, we can obtain transformation matrices among sensors from different vehicles. Real-time point clouds, unlike point-cloud models obtained in advance, are perceived in real-time by each vehicle. However, due to the computational intensity of registration, it is hard to complete real-time pose estimation among multiple vehicles. To address this problem, we propose a hierarchical registration module that consists of two stages: coarse registration and fine registration. The coarse registration aligns real-time point clouds and point-cloud models to get pairwise point clouds. Based on the coarse registration, the fine registration only needs to select the partial point clouds to perform the registration and obtain the transformation matrices. To further reduce the computational latency, EVO employs a Regions of Interest (RoIs) prediction module based on Dlinear [20]. If the prediction is successful, EVO can omit the coarse registration of the current frame. In addition, since coarse registration produces some redundant pairwise point clouds, we optimize the results with the registrations reduction module.

Contributions. In summary, our work makes the following contributions: (1) We design a cooperative perception system, i.e., EVO, which does not need roadside infrastructure or pre-constructed 3D map. EVO employs an novel approach of pose estimation by utilizing the structural features of vehicles, which is able to achieve centimeter-level accuracy. (2) We propose a hierarchical registration module, which speeds up the pose estimation. (3) We design a RoIs prediction module based on Dlinear and a registrations reduction module to reduce computing overhead so that EVO can achieve real-time performance.

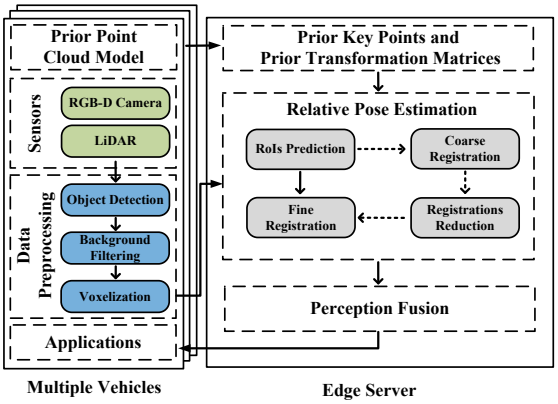


Fig. 2. The EVO system overview.

We evaluate EVO in a multi-vehicle dataset based on the CARLA simulator, which provides open digital assets and flexible APIs for sensor suites. In this simulator, we construct several scenarios, each containing 4-10 moving vehicles. Each scenario consists of continuous scenes for a while. Our results demonstrate that EVO achieves the average end-to-end latency about 60ms, meeting the real-time requirements of autonomous driving. Compared with the ego-vehicle, EVO extends the maximum perception distance from 32m-39m to 38-67m and achieves about $2.43\times$ improvement in the perception area for connected vehicles. In addition, the pose estimation accuracy is significantly higher than state-of-the-art baselines.

2 EVO Overview

EVO mainly consists of a data preprocessing component, a relative pose estimation component, and a perception fusion component, as shown in Fig. 2. The data preprocessing is performed on the vehicle side, while relative pose estimation and perception fusion are deployed on the edge server. Each connected vehicle maintains its own prior point-cloud models, which are extracted features as key points. For every set of key points, there is a matrix as a prior transformation matrix. The edge server maintains a local database of prior key points and prior transformation matrices. Connected vehicles offload key points and matrices within the coverage of the edge server. For clarity and intuition, we use the point-cloud model to represent key points in the following. More specifically, three main components are presented as follows:

Data Preprocessing. This component preprocesses the RGB-D images or point clouds acquired by the sensors on vehicles (such as LiDAR or RGB-D cameras) to reduce the

amount of data transmitted to the edge server. Light object detection is employed to identify regions of interest. EVO then utilizes a background filtering module based on the depth information to remove irrelevant points. Next, semantic information from object detection is used to generate sparse 3D point clouds through the voxelization module. Further details are presented in Sect. 3.1.

Relative Pose Estimation. The relative pose estimation component of EVO takes the fine-grained point clouds generated by the data preprocessing module as input. Then, EVO downsamples the fine-grained point clouds to generate coarse-grained point clouds. EVO performs registration to obtain the transformation matrices of the sensors among vehicles (Sect. 3.2). However, aligning multiple objects observed by each vehicle with multiple prior point-cloud models separately is computationally challenging. Therefore, we design a hierarchical registration module, including coarse registration and fine registration. The coarse registration mechanism identifies pairwise point clouds between the observed real-time point clouds and the prior point-cloud models, and fine registration solves the exact pose transformation matrices by an optimization (Sect. 3.2.1). In addition, to reduce the computation latency, we adopt a RoIs prediction module and a registrations reduction module. Based on the results obtained by the coarse registration, the registrations reduction module constructs the graph structure and determines the pairwise point clouds of the fine registration (Sect. 3.2.2). The RoIs prediction module predicts the position of objects in the current frame based on the past consecutive frames and determines whether the results of the previous coarse registration can be inherited (Sect. 3.2.3).

Perception Fusion. Based on the transformation matrices obtained from the relative pose estimation, EVO fuses the perceptions of multiple vehicles at the edge server. Then, the edge server delivers the fused perception to each vehicle for downstream applications, such as route planning and prediction. Perception fusion can extend the vehicle's perception range and make the autonomous or assisted driving system more reliable.

3 Implementation Details

The EVO system is composed of a data preprocessing component, a relative pose estimation component, and a perception fusion component, in which the relative pose estimation component consists of the hierarchical registrations module, the registrations reduction module, and the RoIs prediction module.

3.1 Data Preprocessing

EVO preprocesses the data to reduce the amount of data transmitted from the vehicle side to the edge server, consisting of object detection, background filtering, and voxelization. The EVO system processes the RGB images and depth information acquired by sensors on vehicles. There are many ways to acquire depth information, LiDAR, RGB-D camera, or sparse LiDAR point clouds for depth complementation. EVO supports any combination of sensors that can provide depth information.

Object Detection identifies potential objects in the RGB image, such as vehicles, bicycles, and people. Two types of detectors are commonly used for 2D object detection: two-stage and single-stage detectors [21]. Two-stage detection typically follows a coarse-to-fine processing paradigm, which can achieve high accuracy but heavy latency. Single-stage detection can retrieve all objects in a single inference step which has the advantage of real-time processing and easy deployment. Because of the limited onboard computing resources and the real-time requirements of the task, we choose the single-stage detector, YOLOv5 [22], as the detector. Furthermore, we introduce the EagleEye [23] algorithm to prune the neural network and reduce computation latency. After object detection, the RoIs in the RGB-D image or LiDAR point clouds are cropped and delivered into the background filtering module.

Background Filtering further reduces redundant points. These redundant points exist because the detection box obtained from object detection is rectangular, which contains background information. Moreover, depth estimation is inaccurate around the object boundaries, leading to the long-tail problem in the pseudo-LiDAR point clouds. These redundant points may affect the accuracy of relative pose estimation and increase the amount of data to be transmitted. We design a simple but effective background filtering module to address this issue. First, we randomly sample the depth of points on the point clouds or depth images. After removing a certain number of extreme values, these depth values are averaged to obtain the average depth, denoted as Dep_{avg}. Specially, for RGB-D data, EVO converts the depth image as 3D point clouds through back projection, as shown in Eq. (1). Here, we convert the points with coordinates (u, v) on the depth image obtained by the RGB-D camera to 3D point (X, Y, Z), where the f_x, f_y, c_x, c_y are the camera intrinsics.

$$\begin{aligned} X &= (u - c_x) \times Dep_{u,v} \,/\, f_x, \\ Y &= (v - c_y) \times Dep_{u,v} \,/\, f_y, \\ Z &= Dep_{u,v}. \end{aligned} \tag{1}$$

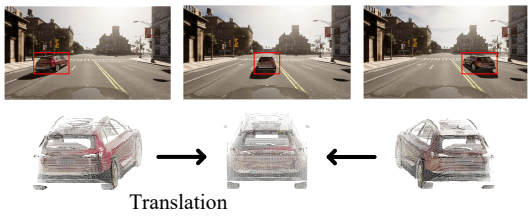
Then, based on the average depth, EVO determines whether each point in detected label is retained or not. This process is as follows:

$$f_{uv} = \begin{cases} 1, & d_{uv} \in [Dep_{avg} - Dep_{label}, Dep_{avg} + Dep_{label}], \\ 0, & else. \end{cases} \tag{2}$$

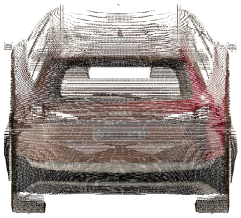
Here, $f_{uv} = 1$ means that the depth information in row u and column v is retained, and vice versa. d_{uv} denotes the distance of the point (u, v) from the sensor position. Dep_{label} denotes the depth range corresponding to the label. For example, for the point clouds labeled as a vehicle, the points in the range of $4m$ before and after the average depth are retained, i.e., $Dep_{car} = 4m$.

Compared to semantic segmentation algorithms, the background filtering module can remove redundant points effectively and achieve segmentation quickly. Then, the background filtering module passes the point clouds to the voxelization module.

Voxelization segments point clouds into different voxels based on their semantic information. However, voxelization



(a) Three different views to collect point clouds of the vehicle.



(b) Prior point-cloud model.

Fig. 3. The process of building a prior point-cloud model.

may cause the points of multiple objects to be fused in a single voxel if they are nearby. These voxels, which contain 3D points from multiple objects, will not reflect the external shapes of any entity. To solve this problem, EVO counts the number of semantic points in each voxel and only retains the points with the highest percentage of semantic points to prevent the fusion of multiple points from different objects. In addition, sporadic points are voxelized to magnify their effect on the 3D structure. We set a threshold based on the voxel size and let the voxels with the number of 3D points less than the threshold be empty.

Following data preprocessing, the real-time point clouds are offloaded to an edge server for computation. We represent perception as relatively sparse 3D points, each containing an additional integer byte of semantic note. EVO retains only the number of voxels in each object so that each point contains only three coordinates to reduce data transmission from the vehicle to the edge server.

3.2 Relative Pose Estimation

In this subsection, we propose the basic pose estimation approach, leaving the entire implementation of the relative pose estimation component to be presented in Sects. 3.3-3.5. First, the pose estimation could be formalized as a registration problem, i.e., how to determine a transformation matrix whereby EVO could align two point clouds as closely as possible. Specifically, given two point clouds $\mathcal{P} = \{p_i \mid i = 1, ..., P\}$ and $\mathcal{Q} = \{q_j \mid j = 1, ..., Q\}$, where p_i and q_j are the 3D coordinates of the i_{th} and j_{th} points in the two point clouds, respectively. We assume that $\mathcal{P}$ and $\mathcal{Q}$ have K pairs of correspondence $(a_k,\ b_k)$, where $k = 1, ..., K$ and $K \leq min(P, Q)$. Our goal is to find the rigid transformation matrix TM, i.e.

$$TM = \begin{bmatrix} R & t^T \\ 0_{1\times 3} & 1 \end{bmatrix}. \tag{3}$$

In Eq. (3), $R \in \mathcal{SO}(3)$ is the rotation matrix and $t^T \in \mathbb{R}^3$ is the transposition of the translation vector t, where $\mathcal{SO}(3)$ is a special orthogonal group containing the rotation matrix

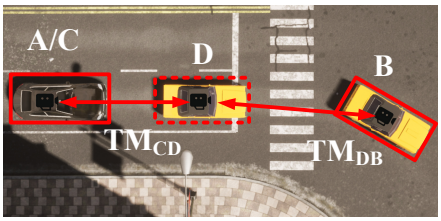


Fig. 4. Relative pose estimation overview.

R, and $\mathbb{R}^3$ is a set of the real number. Then, the registration problem is defined as follows:

$$\underset{R\in\mathcal{SO}(3),t\in\mathbb{R}^3}{arg\ min} \sum_{k=1}^{K} \|p_{a_k} - (Rq_{b_k} + t)\|_2^2 \tag{4}$$

Here, Eq. (4) means minimizing the alignment error between two point clouds. Registration generally includes feature extraction, correspondence searching, and transformation estimation to solve this problem. Feature extraction refers to the extraction of key points and features in the point clouds for correspondence searching. The metric of correspondence searching uses point-to-point feature distances to find the pairwise corresponding points, while transformation estimation uses these correspondences to estimate the transformation matrices.

The pose estimation component utilizes the prior point-cloud models of the vehicles. Fig. 3 shows the process of building a prior point-cloud model for the rear of a vehicle with an RGB-D camera. We carry out the acquisition of the prior point-cloud model of the vehicle at three fixed positions, i.e., left, middle, and right. This step is shown in Fig. 3a. The middle position is called the *reference position* of this point-cloud model. The prior point-cloud models acquired at the three positions are stitched together to obtain a more comprehensive prior point-cloud model, as shown in Fig. 3b. Further, we record the position of the sensors on the vehicle and the reference position. The transformation matrix between these two positions is obtained as a *prior transformation matrix*. Each vehicle stores its prior transformation matrix and prior point-cloud model, which are uploaded to the nearby edge server. When another vehicle observes this vehicle, the real-time point clouds are offloaded to the edge server. EVO performs pose estimation by registration between the prior point-cloud model and the real-time point clouds. However, the result obtained by the registration is not the transformation matrix between the two vehicles. We must combine this result with the prior transformation matrix.

As shown in Fig.4, the sensor on vehicle A observes vehicle B in real-time, and offloads the partial point clouds to the edge server after the data preprocessing module. The objective of relative pose estimation is to find the rigid transformation matrix of the sensors on vehicle A and the sensors on vehicle B, denoted by TM_{AB}. Specifically, we detail the process of pose estimation. First, EVO extracts key points and features from the point clouds perceived by vehicle A. Second, we place the prior point-cloud model of vehicle B directly in front of vehicle A, denoted as a virtual vehicle D. The edge server stores the key points and features of vehicle D. At this point, the sensor position of

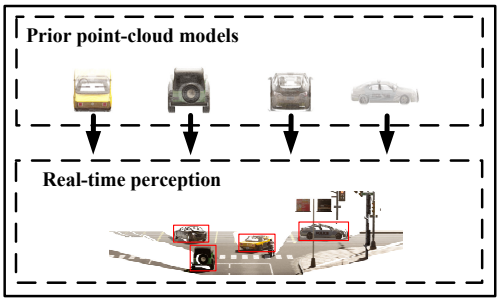


Fig. 5. Real-time perception of a vehicle need to be separately aligned with the nearby prior point-cloud models.

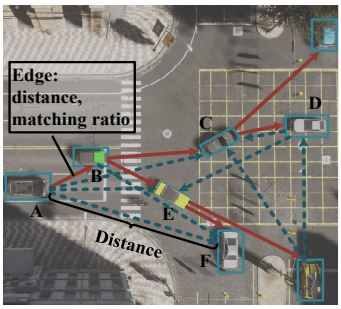


Fig. 6. Computational reduction through registrations reduction module.

vehicle A coincides with position C, which is the reference position of point-cloud model D. So, the transformation matrix TM_{AB} can be decomposed into TM_{AC}, TM_{CD}, and TM_{DB}. Then, correspondence searching is executed between the real-time point clouds perceived by vehicle A and the prior point-cloud model D. The registration is performed based on these correspondences, i.e., the transformation matrix TM_{DB} is estimated. The matrix TM_{CD} is the prior transformation matrix which is obtained in advance. Therefore, TM_{AB} is calculated as shown in Eq.(5).

$$TM_{AB} = TM_{AC} \times TM_{CD} \times TM_{DB}. \quad (5)$$

Since position C coincides with position A, TM_{AC} is an identity matrix. Since the matrices TM_{AC} and TM_{CD} are fixed, the accuracy of the TM_{AB} only depends on the accuracy of the TM_{DB}. Moreover, the external shapes of vehicles are relatively stable compared to 3D map and therefore do not require to be updated frequently.

3.2.1 Hierarchical Registration

We propose a hierarchical registration module that performs real-time relative pose estimation to address the challenge of aligning multiple detected objects with multiple prior point-cloud models. As shown in Fig. 5 and Fig. 6, vehicle A observes four vehicles (B, C, E, F). So the point clouds perceived by vehicle A in real-time need to be aligned with the point-cloud models of these four vehicles, respectively. Due to the error in GNSS positioning, it is difficult to determine the correspondences between each part of the real-time point clouds and multiple point-cloud models. Therefore, it is necessary to align each point-cloud model with all of the point clouds perceived by vehicle A separately. Since the edge server needs to process data from multiple vehicles simultaneously to ensure the system's real-time performance, we design a hierarchical registration module, including coarse registration and fine registration, to reduce redundant calculations. The details of coarse registration and fine registration are as follows.

Coarse Registration. When the real-time point clouds are passed to the relative pose estimation component, downsampling will be applied for these to form coarse-grained point clouds, which have a larger voxel size, fewer points, and faster registration. These coarse-grained point clouds are suitable for finding the pairwise point clouds quickly. EVO identifies nearby vehicles based on rough GNSS information and selects the nearby point-cloud models on the edge server. Unlike previous works that must depend on extremely precise GNSS information, EVO only relies on relatively rough GNSS information to determine the approximate positions and point-cloud models of nearby vehicles. The average error of real-world GNSS positioning is around 4-5m in urban canyon environments [15], [24], which cannot meet the requirements of positioning for autonomous driving. EVO receives the real-time point clouds offloaded by the vehicle and coarsely aligns them with the point-cloud models of nearby vehicles.

The coarse registration module is based on the RANSAC algorithm [25] and is described as follows. First, the coarse registration extracts features from the coarse-grained point clouds. Then, correspondence searching is executed to find correspondences between the coarse-grained point clouds and the point-cloud model. EVO accelerates correspondence searching by building the KDTree structure [26]. Third, EVO randomly selects several pairwise corresponding points as correspondences. The distances among pairwise corresponding points cannot be too small. EVO computes the transformation matrix between the two point clouds based on these correspondences by solving Eq. (4). After a certain number of iterations, we obtain the transformation matrix with the most prominent *fitness* value, which evaluates the overlapping area between the two point clouds. The overlapping area is calculated by the inlier correspondences and the number of points [27]. Finally, with the inlier correspondences, we select the inlier corresponding points in the coarse-grained point clouds and count their semantic notes. The real-time point cloud is divided according to the semantic notes. Among them, the partial point clouds with the largest number of corresponding semantic notes are selected for fine registration with that point-cloud model. EVO performs the above steps for all point-cloud models of nearby vehicles.

Fine Registration. The fine registration module receives the results of coarse registration and extracts features from the partial point clouds. The fine registration calculates the transformation matrices between real-time point clouds and point-cloud models by solving Eq. (4).

The fine registration module is described as follows. First, as the coarse registration, the features are extracted from the point clouds. Then, the correspondence searching is performed. Finally, the transformation estimation is performed based on the results of correspondence searching. Considering accuracy and run time, we introduce the Teaser algorithm [28] for transformation estimation. Teaser uses the

truncated least squares method to discard measurements with large residuals, which allows the problem to exclude false corresponding points. The Teaser algorithm decouples scale, rotation, and translation so that the three transformations can be solved in a cascade. Considering a real driving scene without scaling between pairwise point clouds, we remove the scale estimation to simplify the execution process of Teaser. EVO can filter some of the results with significant errors based on the relation between the normal vector of the ground plane and the rotation matrices.

3.2.2 Registrations Reduction

The results of the coarse registration often have redundant pairwise point clouds between real-time point clouds and point-cloud models, which will increase the computational effort of the fine registration. The registrations reduction module aims at minimizing the computational workload while ensuring high accuracy. It constructs a graph based on the results of coarse registration and identifies the pairwise point clouds that need to be fine registered. EVO assigns weights to the edge based on distance and the correspondence ratio. Based on this graph, EVO uses a minimum spanning tree algorithm to retain the necessary edges and remove the redundant edges to form a connected graph. The retained edges are the pairwise point clouds that need to be fine-registered.

In the instance as shown in Fig. 6, we use RGB-D cameras with a FOV of 110 degrees. The connected lines are the result of the coarse registration. In contrast, the red solid lines are the preserved edges that connect all nearby vehicles with guaranteed accuracy. The transformation matrix between vehicles that are not directly connected can be obtained by matrix multiplication. We assume that the transformation matrix between vehicle A and vehicle B is TM_{AB}, and the transformation matrix between vehicle B and vehicle C is TM_{BC}. So, the transformation matrix from vehicle C to vehicle A defines: $TM_{AC} = TM_{AB} \times TM_{BC}$.

In addition, the distance between A and C is significantly larger than that between A and B. Further distance means less precise depth information and less dense point clouds, which can decrease registration accuracy. If the transformation matrix of A and C is calculated directly, the accuracy of the fine registration may be reduced. By designing the weights of the edges, the pose estimation between distant vehicles is avoided as much as possible. We construct the directed graph structure based on the results from coarse registration. The set of directed edges is denoted by $\mathcal{E} = \{Edge_{ij} |\ i = 1, ..., N,\ j = 1, ..., N,\ i \neq j\}$. When one of the vehicles perceived by vehicle v_e corresponds to the point-cloud models of vehicle v_n, we set the weight of $Edge_{en}$ based on tuple $(v_e,\ v_n,\ l,\ cr_l,\ d_l)$ as follows:

$$Edge_{en} = \frac{\sqrt{d_l}}{cr_l}. \tag{6}$$

where l means the semantic note of the partial point clouds perceived by vehicle v_e. The weight of edges indicates that EVO tends to select the pairwise point clouds with closer distances and greater correspondence ratio.

The process of the registrations reduction module is described as follows. After building the directed graph, EVO converts the directed graph into an undirected one. If there is a directly connected edge between two vertices, we modify this edge to an undirected edge. If two directly connected edges with opposite directions between two vertices, we keep the edge with a smaller weight and change it to an undirected one. Subsequently, we use the Prim algorithm to obtain the edges in the undirected graph. The pairwise vertexes of these edges are passed into the fine registration module for relative pose estimation. For each vertex, EVO retains the paths below a certain number of hops between the remaining vertices and this vertex to get the transformation matrices. The number of hops is limited to prevent the accumulation of pose estimation errors.

3.2.3 Regions of Interest Prediction

The module takes advantage of the fact that objects may have similar positions in successive videos. There is a correlation between the coarse registration results of the previous and current frames in a successive video. However, we cannot determine this correlation directly based on the position at the current frame because multiple objects may have similar positions at close frames. We can infer the object's movement from the positions at the historical frames and thus predict the position at the current frame on the RGB image. EVO calculates the deviation of the object's current position from the predicted position. If the deviation is less than a specific threshold, the prediction is successful, and the results of the previous coarse registration can be utilized. We can omit the computation of coarse registration to directly determine the pairwise point clouds that need to be finely registered. Therefore, we design the RoIs prediction module to predict the current position of an object based on historical data.

Deep learning techniques have been widely used for multiple object tracking [29], [30]. Based on the evaluation in [31], the RNN neural network using LSTM as the base unit is significantly better than the CNN-based network in terms of real-time performance. Different from previous works, we employ a set of linear models for real-time RoIs prediction inspired by LSTF-Linear [20]. We utilize the idea of seasonal trend decomposition in Autoformer [32] to make the raw data more predictable before the linear layers.

Since the objective is to train a linear layer-based RoIs prediction network, the optimization process can be expressed as follows:

$$\begin{aligned} &\min_{\theta} \mathcal{L}(\hat{L}^{\tau} - L^{\tau}) = \min \sum\nolimits_{i} [(\hat{x}_{i,l}^{\tau} - x_{i,l}^{\tau})^2 + (\hat{x}_{i,r}^{\tau} - x_{i,r}^{\tau})^2 + \\ &(\hat{y}_{i,t}^{\tau} - y_{i,t}^{\tau})^2 + (\hat{y}_{i,b}^{\tau} - y_{i,b}^{\tau})^2 + (\hat{a}_{i}^{\tau} - a_{i}^{\tau})^2] \\ &\quad s.t. \quad \hat{L}^{\tau} = f(\{L^{\tau-\tau^{'}}, L^{\tau-\tau^{'}+1}, ..., L^{\tau-1}\}, \theta), \end{aligned} \tag{7}$$

where the L^{τ} denotes RoIs information in frame τ, and $\hat{L}^{\tau}$ is the predicted RoIs. L^{τ} is a set of vectors, one of which is denoted by $l_i^{\tau} = \{x_{i,l}^{\tau}, x_{i,r}^{\tau}, y_{i,t}^{\tau}, y_{i,b}^{\tau}\}$. The values in the vector l_i^{τ} represent the horizontal coordinates of the left and right, and the vertical coordinates of the top and bottom of the detection box, respectively. a_i^{τ} is the area of the detection box calculated by l_i^{τ}. Further, $\tau^{'}$ is the number of previous frames used in the prediction network f, and θ is the set of network parameters to be optimized.

Our network consists of two steps, i.e., series decomposition and two linear layers. First, series decomposition

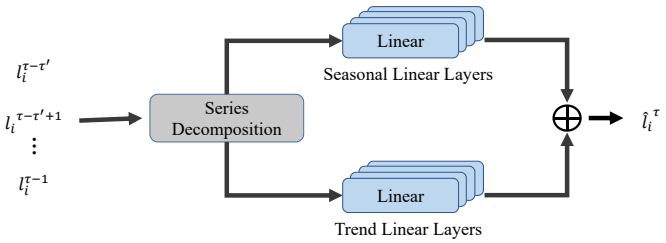


Fig. 7. Network architecture for the RoIs prediction module.

decomposes the raw data input into trend data and seasonal data. Then, two parts of the data are fed into the respective linear layers, and we sum up the two features to get the final prediction. It is worth mentioning that we input the RoIs coordinates into four separate linear layers. The network structure is shown in Fig 7.

Series decomposition could enhance the performance of a vanilla linear when there is a clear trend in the raw data. We use a moving average to smooth out periodic fluctuations and highlight the trends. For length-$\tau^{'}$ input series $\mathbf{D} = \{l_i^{\tau-\tau^{'}}, l_i^{\tau-\tau^{'}+1}, ..., l_i^{\tau-1}\} \in \mathbb{R}^{\tau^{'} \times 4}$, the process of series decomposition is:

$$\mathbf{D}_{tr} = AvgPool(Padding(\mathbf{D})). \tag{8}$$

$$\mathbf{D}_{sea} = \mathbf{D} - \mathbf{D}_{tr}, \tag{9}$$

where $\mathbf{D}_{sea}, \mathbf{D}_{tr} \in \mathbb{R}^{\tau^{'} \times 4}$ denotes the seasonal data and the trend data, respectively. The $AvgPool$ for moving average with the $padding$ operation keeps the series length unchanged. We input the $\mathbf{D}_{sea}$ and $\mathbf{D}_{tr}$ into separate linear layers, and sum up the features obtained from seasonal and trend linear layers to get the final prediction.

EVO calculates the deviation between the predicted value and object detection result:

$$\begin{aligned} Dev(\hat{l}_i^{\tau} - l_i^{\tau}) = (\hat{x}_{i,l}^{\tau} - x_{i,l}^{\tau}) + (\hat{x}_{i,r}^{\tau} - x_{i,r}^{\tau}) + \\ (\hat{y}_{i,t}^{\tau} - y_{i,t}^{\tau}) + (\hat{y}_{i,b}^{\tau} - y_{i,b}^{\tau}) + (\hat{a}_i^{\tau} - a_i^{\tau}). \end{aligned} \tag{10}$$

On the edge server, we keep the object detection results uploaded by each vehicle, aggregate these results, and make predictions on the data of all connected vehicles at the current frame. After comparing the Dev with a threshold, we consider objects with deviations larger than the threshold as prediction failures. EVO performs a separate coarse registration for these failed predictions. In addition, EVO performs coarse registration at regular intervals to reset the correspondences.

3.3 Perception Fusion

EVO aggregates the point clouds of each vehicle on the edge server. The edge cloud can significantly reduce the transmission latency compared with the cloud, and can provide more computing resources compared with mobile devices. By exchanging data through the edge server, limitations of direct data interaction between vehicles can be avoided. For example, in V2V systems, vehicles must send data multiple times or forward it through other vehicles, which results in additional latency and bandwidth consumption. Furthermore, aggregating information at the edge cloud can ensure the privacy and security of each vehicle. The edge server can also handle inconsistencies in the time when the data are offloaded to the server based on the movement information.

According to the type of shared sensor data, multi-vehicle perception fusion is divided into three primary levels: raw-level, feature-level, and object-level [6]. The raw fusion will share the raw point clouds, which challenge the transmission bandwidth and latency. Feature-level fusion will fuse intermediate features from 3D object detections with small amounts of data, while object-level fusion will only fuse the results of detections. Multiple vehicles can upload their 3D object detection results to the edge server, where the results are aggregated and distributed. Feature-level data or raw point clouds can be offloaded for more accurate detection results when network load allows. As a complement to the autonomous or assisted driving system, EVO provides the relative poses for various perception fusion methods and is suitable for any form of perception fusion.

4 Performance Evaluation

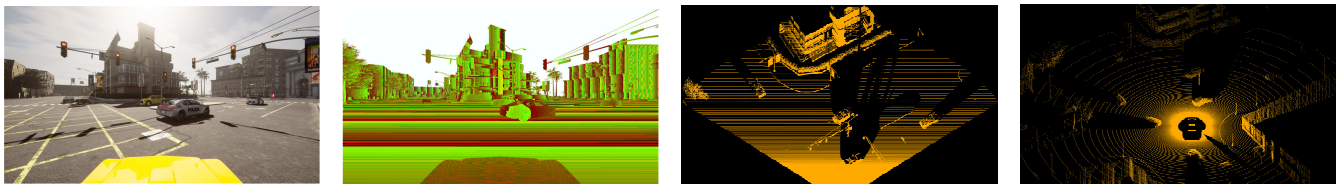
This section details the related implementation of EVO and conducts experiments in different road scenarios. We evaluate EVO in terms of end-to-end delay, accuracy, and application-level performance, and compare it with two widely deployed SLAM methods. In addition, we examine the RoIs prediction module. Our results show that EVO is able to achieve centimeter-level pose estimation accuracy while ensuring real-time performance and effectively improving the perception range of vehicles.

4.1 Evaluation Setup

System implementation. EVO consists of over 6K lines of code written in Python and C++. To balance precision and runtime, we choose YOLOV5 [22] as the 2D detector, which uses the Object dataset from the KITTI [33] for training and pruning. Open3D [27] and PCL [34] are used to process the point clouds, and PointPillars [35], a widely used 3D detection algorithm, is chosen for the detection of 3D point clouds. We introduce the designed 3D feature extraction method FPFH [36]. The manually designed features are able to accommodate the point clouds of different voxel sizes.

The components of EVO are deployed in vehicles and edge servers. We allocate different computational resources to them. Each vehicle utilizes a laptop with AMD R7 4800H CPU, GeForce RTX 2060 GPU, and 16 GB DDR4 3200Mhz RAM. The edge server has more computational resources than the vehicle side. However, we limit the upper limit of the edge server computational resources to distinguish the edge server from the cloud server. The edge server has 2 * Intel Xeon Sliver 4310 CPUs, GeForce RTX 3080TI GPU, and 6 * 32 GB DDR4 3200Mhz RAM. Edge server has more concurrent computing power to process data sent from several vehicles simultaneously.

Dataset description. Since existing multi-vehicle collaboration datasets lack the prior knowledge required by EVO, we utilize the CARLA simulator [37] to build a multi-vehicle collaborative dataset to evaluate EVO. CARLA is an open



(a) a RGB image collected by vehicle in CARLA.

(b) a depth image collected by vehicle in CARLA.

(c) point clouds collected by vehicle in CARLA.

(d) point clouds obtained by back-projection from RGB-D image.

Fig. 8. Dataset consisting of RGB-D images and LiDAR point clouds.

urban driving simulator that creates free-use digital assets, including city layouts, buildings, and vehicles. CARLA provides the flexibility to configure the sensor suite of agents, whereby we can specify the placement of sensors such as cameras, depth cameras, and LiDAR for the vehicle. CARLA also provides a range of measurements related to vehicle state and traffic rule compliance. State measurements include vehicle position and direction relative to a world coordinate system, velocity, acceleration vectors, and the cumulative impact of collisions.

We generate collaborative multi-vehicle datasets that contains more than forty scenarios extracted from town maps, including intersections, meetings, and traffic jams. We divide these scenarios into hard and easy scenarios based on whether the average speeds of the vehicles are above 20mph. At least four vehicles in motion are present in each scenario, and each scenario consists of continuous scenes. The perceptions from the sensors of vehicles in these scenes are captured. Further, we collect depth data using the two methods provided by CARLA, i.e., the RGB-D camera and LiDAR, respectively, as shown in Fig. 8. The depth image in Fig. 8(b) is obtained by the RGB-D camera, and the point clouds in Fig. 8(c) are obtained by LiDAR. The depth image can be back-projected as point clouds, as shown in Fig. 8(d). The datasets contain more than 1900 data tuples, each of which consists of depth data, RGB images, ground truths of sensor locations, and rough GNSS data. The LiDAR data are cropped to point clouds with a FOV of 110 degrees. In addition, We collect point-cloud models for nearly thirty vehicles and prior transformation matrices.

For each scene, the data to be processed and offloaded at the vehicle side includes RGB images, depth information (depth image or point clouds), and rough GNSS information. The deviation of the GNSS positioning is between 5 and 10m in each scene. The edge side stores the features of the point-cloud models of all the vehicles in that scene, including coarse-grained features and fine-grained features extracted in advance. In addition, we record each vehicle its true location at each moment as the ground truth for evaluating accuracy.

4.2 Evaluation Methodology

Evaluation metrices. For EVO, we use three metrics to measure the performance of the system, i.e., run time, accuracy, and application-level performance. We evaluate test set loss and prediction success rates for the RoIs prediction module.

First, we focus on the end-to-end latency, including the computational latency of each component and the transmission latency between the vehicles and the edge server. We measure the run time and effectiveness of modules in the data preprocessing component and the relative pose estimation component.

Second, we evaluate the accuracy of EVO by two main metrics, the relative translation error (RTE) and relative rotation error (RRE) [28]. Using the true locations of the sensors, we calculate the ground truth transformation matrix TM_g, representing the transformation of poses between vehicles. We compare the ground truth with the result of the fine registration TM_f to obtain the RTE and REE. RTE is calculated as: $RTE = \|t_g - t_f\|$, where t_g and t_f are the translation vectors of TM_g and TM_f, respectively. RRE is defined as: $RRE = |arccos((tr(R_g^T R_f) - 1)/2)|$, where R_g and R_f are the rotation matrices of TM_g and TM_f. As in [16], We define the results where the RTE is less than a predefined threshold as 'success'.

Then, we evaluate the application-level performance of the vehicle after deploying EVO. EVO focuses on extending the perception range of vehicles, enabling the vehicle to obtain the perception of blind spots and distant objects. We set two evaluation metrics, i.e., the maximum perception distance and the perception area. We project and voxelize the point clouds before and after fusion to the ground plane. For maximum perception distance, we set the distance between the farthest voxel and the origin as the maximum perception distance. For the perception area, we obtain it by counting the number of these voxels.

Comparing EVO with existing work. We consider *EVO-Naive*, a variant of EVO, to evaluate the hierarchical registration module. EVO-Naive does not include the hierarchical registration module that directly aligns multiple point-cloud models with real-time point clouds in pose estimation. Furthermore, we classify the current state-of-the-art methods into two categories, namely visual SLAM [11], [38], [39] and LiDAR SLAM [5]. We deploy these methods for relative pose estimation among vehicles and evaluate the accuracy compared with EVO.

4.3 Evaluation of Real-time Performance

4.3.1 End-to-End latency.

We focus on the overall end-to-end latency to meet the real-time performance. We evaluate each EVO component's impact on the overall end-to-end latency separately. Here, we assume that the edge server is always available to execute the relative pose estimation at any time.

To evaluate the end-to-end real-time performance, we measure the overall latency of EVO in the datasets, where

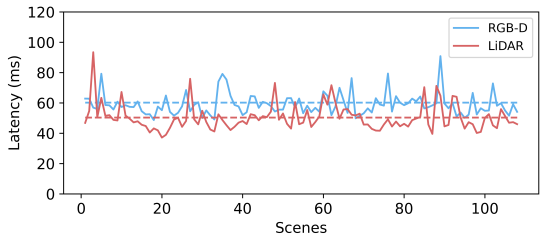


Fig. 9. Measurement of end-to-end latency in continuous scenes.

the coarse-grained voxel size in the relative pose estimation module is set to 0.35m, and the fine-grained voxel size is set to 0.1m. We measure the performance of multiple vehicles with continuous scenes in RGB-D camera and LiDAR datasets, respectively. The result is shown in Fig. 9. The denser depth information from the RGB-D camera results in greater computational overhead and end-to-end latency. The mean values of the end-to-end latency of the driving sequence evaluation based on the two depth sensors are about 60.2ms and 50.3ms, respectively, which are less than 100ms. EVO is able to complete the pose estimation in real-time.

4.3.2 *Run time of the data preprocessing module.*

We validate the effectiveness of the data preprocessing and relative pose estimation module to the real-time performance of EVO separately. First, we observe the number of points retained after each step of the data preprocessing component and count the average run time. EVO processes raw RGB-D images (1920*1080) or LiDAR point clouds to reduce the amount of data transferred from the moving vehicle to the edge server and to eliminate irrelevant points. Table 1 shows the effect of each data preprocessing step on the number of points and computational latency. Background filtering is able to reduce the number of points by 22.99% and 25.52%, respectively, while running at just 2.65ms and 0.99ms. Due to the larger number of points and the fact that depth images need to be back-projected to form 3D point clouds, the computation of the background filtering module takes longer on the RGB-D images than LiDAR point clouds. Voxelization can effectively convert dense points into relatively sparse ones to reduce the data offloaded to the edge server. After data preprocessing, the offloaded RGB-D images decrease from the original 1.07MB to 30.04KB, and the LiDAR point clouds decrease from the original 671.29KB to 16.98KB. In addition, we simulate the vehicle-side to edge-side network transmission on a mobile vehicle by connecting a laptop to the edge server via wifi wirelessly and measure the data transfer time traces under an 80MHz 802.11ac wireless network. The results show that the average data transmission latency per frame is 3.31ms and 1.93ms. The transmission latency is significantly reduced compared to transmitting raw data.

4.3.3 *Run time of the hierarchical registration module.*

We evaluate the contribution of the hierarchical registration module to the real-time performance of EVO. RoIs prediction may produce three different states, i.e., success, partial success, and reset. A successful prediction means that the objects of the current frame are successfully tracked by the historical frames so that EVO can skip the coarse registration. Predicting partial success means that EVO needs to perform the coarse registration of partial point clouds. EVO performs a reset, i.e., the coarse registration, at fixed intervals in order to discover new objects and correspondences. In the experiment, we set this interval to every 5 frames, so the reset rate is above 20%. When the number of pairwise point clouds between vehicles obtained at the previous scene is less than 2, we reset the coarse registration. EVO also executes a reset when all of the predictions fail. In our dataset, we count each state's proportion and run time obtained by RoIs prediction module, as shown in Table 2. The average computational latency in the four groups is less than the reset latency, indicating that the incorporation of the RoIs prediction module can effectively reduce the computational latency of the coarse registration module.

We validate the effectiveness of the hierarchical registration module by comparing EVO with EVO-Naive. The results are shown in Fig. 10. EVO-Naive simply aligns real-time point clouds with point-cloud models of nearby vehicles. The pre-registration includes coarse registration, RoIs prediction, and registrations reduction. EVO-naive does not perform pre-registration, so the run time of feature extraction and correspondence searching is significantly larger than that of EVO. In the experiment, the average total run time of the relative pose estimation module is 35.58ms in EVO and 74.11ms in EVO-Naive.

4.4 Evaluation of Accuracy

Besides the run time, it is crucial to demonstrate whether the pose estimation module of EVO can meet the accuracy requirements of multi-vehicle cooperative perception.

4.4.1 *Accuracy of the hierarchical registration module.*

We choose the fine-grained voxel size of 0.1m and evaluate the fine registration accuracy of the RGB-D and LiDAR datasets separately. The results are shown in Fig. 11 and

TABLE 1
The number of retained points and run time after each step in data preprocessing module, where the fine-grained voxel size is 0.1m.

		Number of Points/Voxels	Proportion(%)	Run Time (ms)
RGB-D	Detection	85806	100	14.1
	Background Filtering	66078	77.01	2.65
	Voxelization	2433	2.84	4.77
LiDAR	Detection	6143	100	14.08
	Background Filtering	4575	74.48	0.99
	Voxelization	1377	22.42	1.07

TABLE 2
We evaluate the rate and running time of coarse registration in RoIs predicting success, partial success, and reset.

Dataset	Group	Success(%/ms)	Partial Success(%/ms)	Reset(%/ms)	Average(ms)
RGB-D	Easy	62.65/5.33	14.46/16.37	22.89/15.67	9.29
	Hard	57.90/5.27	18.42/15.70	23.68/18.12	10.23
LiDAR	Easy	64.52/5.31	8.60/13.75	26.88/14.03	8.38
	Hard	51.61/5.23	19.35/13.70	29.03/16.18	10.05

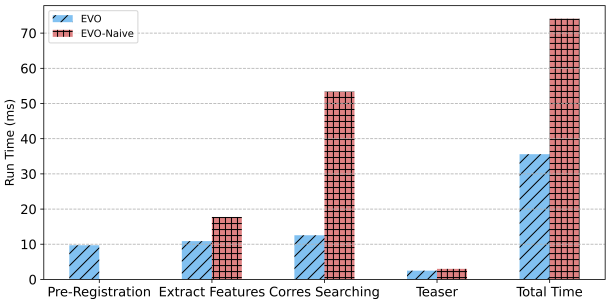


Fig. 10. The average run time of each step of relative pose estimation in EVO and EVO-Naive.

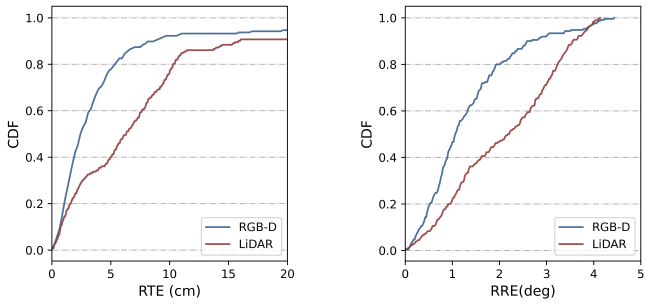


Fig. 12. CDF of RTE(left)/RRE(right) with different sensors.

Fig. 12. For the RGB-D dataset, there is more point cloud information in each data frame, which enables relatively accurate pose estimation. The average RTE in RGB-D dataset is 11.97cm, while in the LiDAR dataset, it is 16.42cm. The average values of REE are 1.53 degrees and 2.72 degrees with two different sensors, respectively. We count the success rate of the relative pose estimation module, which is above 96%. Fig. 12 shows the cumulative distribution functions (CDFs) of RTE and RRE. On the RGB-D dataset, more than 90% of RTE results are below 20cm and more than 80% of results are below 10cm. Most of the results have an RRE that is less than 4 degrees. Due to the sparser point clouds of LiDAR, the accuracy of EVO on the LiDAR dataset is lower than that on the RGB-D camera dataset. Subsequently, we can decrease the fine-grained voxel size of LiDAR data to obtain more denser point clouds and achieve higher accuracy. In summary, EVO enables the accurate transformation matrices between vehicles as the basis for perception fusion.

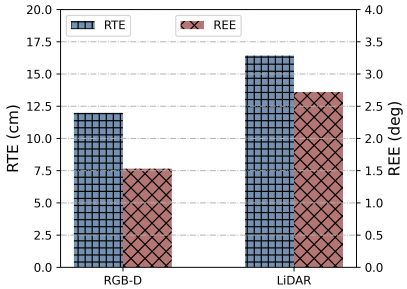


Fig. 11. RRE/RTE with different sensors.

4.4.2 Comparison with SLAM-based methods.

We compare the accuracy of pose estimation with EVO and SLAM-based methods for four intersection scenarios in our dataset. We employ the works based on LiDAR SLAM and visual SLAM to construct the 3D points map of these scenarios, respectively. Then, we set up these scenarios with multiple vehicles with line-of-sight occlusions between them to simulate a realistic driving circumstance. We evaluate the localization accuracy and the run time of each of these methods, where the run time of the SLAM-based methods is not counted for map construction and downloading. We set the line and plane resolution for LiDAR SLAM as low as possible, and make visual SLAM extract as many feature points as possible while guaranteeing real-time performance. The results are shown in Table 3. In contrast, the relative pose estimation component in EVO performs better than these SLAM methods in terms of accuracy and runtime.

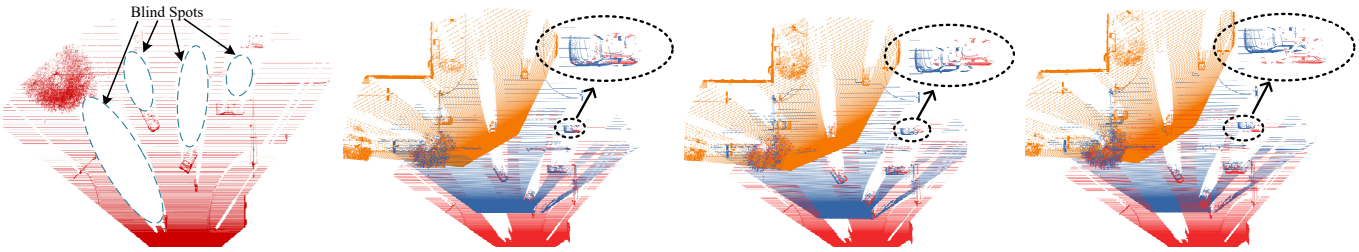
In addition, we utilize the localization results of the three methods as a basis for fusing the perceptions of three vehicles in the same scene to visualize the accuracy of EVO. Fig. 13(a) shows the perception of an ego-vehicle, which has several blind spots. The others in Fig.13 demonstrate the effect of the fusion after three methods, where EVO makes observations of the same vehicle from different vehicles more similar due to higher accuracy.

4.4.3 Analysis of registrations reduction.

We evaluate the effect of registrations reduction on the accuracy and success rate of pose estimation. In registrations reduction, the maximum hop count is configured as 3 to avoid the accumulation of errors. We evaluate RTE and success rates according to hop count in our dataset. The results of RTE at different hop counts are shown in

TABLE 3
Comparison of EVO and SLAM-based methods in terms of localization accuracy and runtime.

Method	Sensor	RTE(m)	Run Time (ms)
LiDAR SLAM [5]	LiDAR	0.45	82.05
Visual SLAM [39]	RGB-D	0.64	86.08
EVO	LiDAR	0.16	51.34
	RGB-D	0.10	64.70



(a) Perception range of ego-vehicle. (b) Fusion after EVO. (c) Fusion after LiDAR SLAM. (d) Fusion after visual SLAM.

Fig. 13. The effect of fusion after different methods of pose estimation.

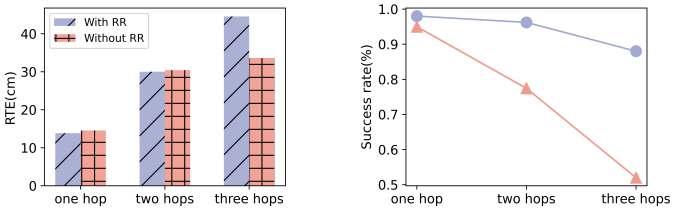


Fig. 14. RTE(cm)/Success rate(%) in different hop count with/without registrations reduction.

Fig. 14. As the hop count increases, the error in the relative pose estimation accumulates, reaching 0.44m for the RTE when the hop count is 3 with the registrations reduction module. In contrast, EVO enables fine registration between more pairwise point clouds without registrations reduction, thus reducing the cumulative error in pose estimation. The higher the hop count, the lower the success rate of the pose estimation. With registrations reduction, the success rates of one hop and two hops are 98% and 96%, respectively, while the success rate for three hops is only 88%. Since the registrations reduction module filters out many pairwise point clouds with large distances and small correspondence ratios, the success rate is higher than that without registrations reduction. In addition, the calculations of fine registration are reduced to 57.6% of the original amount with registration reduction. Although the registrations reduction module adds a few cumulative errors, it significantly increases the success rate of fine registration and reduces the amount of computation.

4.5 Application-level Performance

EVO focuses on extending the perception range of vehicles. We evaluate the application-level performance of EVO by comparing the overall perception area of the vehicle before and after the deployment of EVO, as well as the maximum perception distance. First, we project the point clouds to the ground plane and divide that into voxels. If the number of points in a voxel is above a threshold, we consider the region to be observable, and vice versa, we consider the region to be a blind spot. The voxel size is 0.5m, so each voxel represents that $0.25m^2$ area to be perceived. Then, we count the number of voxels in the perceptions before and after fusion. We choose a vehicle for each scenario as the primary vehicle to validate the effect.

We deploy EVO to integrate RGB-D images and LiDAR point clouds. The results are shown in Fig.15 and Fig.16. The maximum perception distance of each ego-vehicle is 32.3 - 39.6m before fusion, while the deployment of EVO increases the maximum perception distance to 38.8 - 67.0m. The average maximum perception distances are 35.5m and 48.0m, respectively. Therefore, EVO is able to effectively increase the perception distance of ego-vehicle by up to $1.31\times$ in all scenarios. These scenarios include meetings, intersections, and followings, where the improvement of maximum perception distance is smaller in meetings. In the set of scenes that do not include meetings, the average maximum perception distance is boosted to 53.6m, which reaches $1.51\times$ that of an ego-vehicle. In addition, EVO could provide a more comprehensive area of point clouds. Before fusion, the average perception area perceived by the ego-vehicle is $243.9m^2$, while after fusion, it is $594.5m^2$. EVO achieves the perception area by 243%. In short, EVO can effectively extend the vehicles' perception range.

4.6 Evaluation of RoIs Prediction

We use the KITTI dataset [33] to train the RoIs prediction module. KITTI is an open-source autonomous driving dataset that includes real datasets from multiple scenarios. Using 75% of the data in this dataset as training data, we train the Dlinear-based prediction network and employ the Adam optimizer [40] with a learning rate of 1e-3 to minimize the loss function in Eq. (7). We compare the RoIs prediction network with an attention LSTM network, which is proposed in ELF [41].

The training loss and test loss are shown in Fig. 17 and Fig. 18. The training loss curve shows that the training loss of our model is lower than that of ELF's proposed attention LSTM model, 0.120 VS 0.171. Further, the RoIs prediction produces an average test loss of 0.1389 on the test sets, while the attention LSTM produces an average loss of 0.1784 on the test sets. The results show that our Dlinear-based prediction model outperforms the attention LSTM model.

Further, we employ both prediction methods in the EVO and evaluate their accuracy in our dataset. The evaluation is based on Eq. (10), where we set the loss threshold as 0.1. We count the percentage of successful and partially successful predictions obtained by both methods. Our proposed Dlinear-based prediction method has a higher success rate than attention LSTM, 83.85% vs 77.36%. In partial successful prediction, 57.85% objects are predicted successfully in Dlinear compared to 48.42% in attention LSTM.

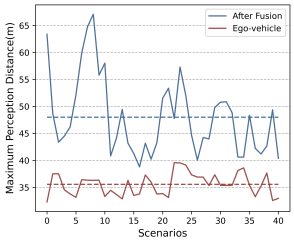


Fig. 15. The maximum perception distance in each scenario.

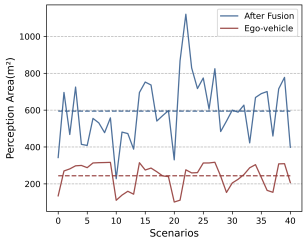


Fig. 16. The average perception area in each scenario.

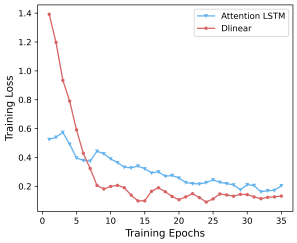


Fig. 17. Training loss of attention LSTM and Dlinear.

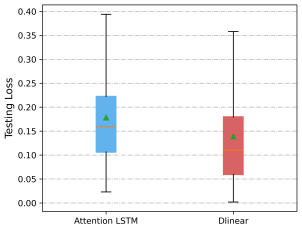


Fig. 18. Testing loss of attention LSTM and Dlinear.

5 Related Works

5.1 Cooperative Perception Based on Connected Vehicles

Connected autonomous vehicles have been considered as an innovative and sustainable future mobility solution [1]. AVR exploits data collected by stereo cameras and feature-based maps to perform pose estimation [9] according to ORBSLAM2 [13]. AutoCast focuses on infrastructure-free methods to maximize information sharing through wireless channels [10]. Its control plane and data plane assume that each vehicle can utilize pre-constructed 3D maps via LiDAR SLAM for pose estimation. When there are many dynamic objects, the accuracy of pose estimation decreases. CarMap introduces crowdsourcing to solve the map update problem [39]. CarMap only contains map features, which effectively reduces the size of the map. Other methods focus on synergistic perception and do not detail the pose estimation process. [5].

Cooperative perception systems utilizing roadside infrastructures are potential solutions to high-level autonomous driving [42]. VI-EYE assembles LiDAR and computing resources of roadside infrastructures, extracts salient points from the real-time point clouds acquired by vehicles and infrastructures, and performs registration to share the original point clouds [16]. VIPS independently performs 3D object detection on infrastructures and vehicles, constructs two multi-affinity graphs based on target information, and uses graph-matching algorithms to identify common visible targets of vehicles and infrastructures [15]. Due to its cost, it will take time for infrastructure-assisted autonomous driving to spread.

In addition, a lot of works focus on fusing perceptions from multiple vehicles in the presence of network transmission constraints [6]–[8], [43], [44]. OPV2V proposes a new attentive intermediate fusion pipeline to aggregate intermediate layer information for object detection [7]. V2VNet uses a spatial perceptual graph neural network to aggregate information received from all nearby self-driving vehicles, allowing us to intelligently combine data from different points in time and viewpoints in the scene [44]. Most of this type of works use intermediate fusion, ensuring detection accuracy and reducing the stress of the network. However, these works assume that the position of the vehicles is known and ignore the process of pose estimation.

5.2 Edge Computing for Mobile Devices and Vehicles

With edge computing, mobile devices can migrate computationally intensive tasks to edge servers to reduce the occupation of their computing resources. Due to the richer computing resources of edge servers compared with mobile devices, tasks can be offloaded to edge servers to reduce computation latency. Liu et al. achieve real-time high-precision object detection for commercial AR/MR systems by adopting low-latency offloading technology with the assistance of edge server [45]. Elf speeds up the execution of computationally intensive tasks by splitting and offloading computational tasks to multiple edge cloud platforms [41]. EdgeDuet selectively offloads objects to the edge cloud according to the resolution and optimizes the offloaded detection pipelines in tiles to improve detection accuracy and reduce latency [46].

In the process of CAV (connected autonomous vehicles) perceptions sharing, the edge cloud can be used as a platform for data fusion and forwarding, which avoids extra delay and bandwidth consumption. Unlike V2V sharing, the vehicles only need to offload their data once to an edge server [5]. Livemap processes data from individual vehicles to detect, track, and match objects on the road and uploads the results to the edge platform for real-time global information [38]. Edgesharing has a real-time 3D feature map in the edge server that provides accurate location and object-sharing services to vehicles passing through the region [11].

6 Discussion and Future Works

Supplement to "Relative". "Relative" means that the vehicle is aware of the pose of other vehicles relative to itself but not of its own pose in the world coordinate system. For perception fusion between vehicles, the relative pose is sufficient. However, in some applications of autopilot systems, it may be necessary to know the absolute position. With the relative position, we can also get the precise absolute position. In a local region, the vehicle with the minimum perception constraints can be selected from multiple vehicles for absolute positioning via the SLAM. The absolute position of other vehicles can be obtained by relative transformation. In addition, precise absolute positions can be obtained by optimizing the relative pose and GNSS information of multiple vehicles through the Kalman filter.

Shared Range of Perception. In our experimental dataset, the FOV of depth information in the RGB-D and LiDAR datasets is 110 degrees. We argue that the perception extension in the front of the connected vehicles will have a more critical impact on decision-making than in the rear. EVO can easily extend the FOV to 360 degrees, which requires multiple cameras or a 360 panoramic camera. In the future, we will consider the decision-making to transfer

only essential perceptions between vehicles based on their short-term driving behaviour.

7 Conclusion

In this paper, we present an edge-assisted multi-vehicle cooperative perception system, i.e., EVO, without requiring any pre-constructed 3D map and roadside infrastructures. EVO is composed of the data preprocessing component, the relative pose estimation component, and the perception fusion component. We design the hierarchical registration module, the RoIs prediction module, and the registrations reduction module to ensure real-time performance and accuracy in relative pose estimation. The results show that our system can increase the maximum perception distance from 32-39m to 38-67m, and achieve a $2.43\times$ improvement in perception area. In addition, EVO is able to estimate the relative poses of multiple vehicles in real-time and obtain more accurate results than state-of-the-art works.

References

[1] P. Seuwou, E. Banissi, and G. Ubakanma, "The future of mobility with connected and autonomous vehicles in smart cities," *Digital twin technologies and smart cities*, pp. 37–52, 2020.

[2] Y. Wang, Q. Mao, H. Zhu, Y. Zhang, J. Ji, and Y. Zhang, "Multi-modal 3d object detection in autonomous driving: a survey," *arXiv preprint arXiv:2106.12735*, 2021.

[3] A. Chtourou, P. Merdrignac, and O. Shagdar, "Collective perception service for connected vehicles and roadside infrastructure," in *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*. IEEE, 2021, pp. 1–5.

[4] G. Cui, W. Zhang, Y. Xiao, L. Yao, and Z. Fang, "Cooperative perception technology of autonomous driving in the internet of vehicles environment: A review," *Sensors*, vol. 22, no. 15, p. 5535, 2022.

[5] X. Zhang, A. Zhang, J. Sun, X. Zhu, Y. E. Guo, F. Qian, and Z. M. Mao, "Emp: Edge-assisted multi-vehicle perception," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, 2021, pp. 545–558.

[6] Y. Jia, R. Mao, Y. Sun, S. Zhou, and Z. Niu, "Mass: Mobility-aware sensor scheduling of cooperative perception for connected automated driving," *arXiv preprint arXiv:2302.13029*, 2023.

[7] R. Xu, H. Xiang, X. Xia, X. Han, J. Li, and J. Ma, "Opv2v: An open benchmark dataset and fusion pipeline for perception with vehicle-to-vehicle communication," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 2583–2589.

[8] Y. Hu, S. Fang, Z. Lei, Y. Zhong, and S. Chen, "Where2comm: Communication-efficient collaborative perception via spatial confidence maps," *arXiv preprint arXiv:2209.12836*, 2022.

[9] H. Qiu, F. Ahmad, F. Bai, M. Gruteser, and R. Govindan, "Avr: Augmented vehicular reality," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, 2018, pp. 81–95.

[10] H. Qiu, P. Huang, N. Asavisanu, X. Liu, K. Psounis, and R. Govindan, "Autocast: Scalable infrastructure-less cooperative perception for distributed collaborative driving," *arXiv preprint arXiv:2112.14947*, 2021.

[11] L. Liu and M. Gruteser, "Edgesharing: Edge assisted real-time localization and object sharing in urban streets," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[12] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time." in *Robotics: Science and Systems*, vol. 2, no. 9. Berkeley, CA, 2014, pp. 1–9.

[13] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE transactions on robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[14] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and D. Rus, "Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping," in *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2020, pp. 5135–5142.

[15] S. Shi, J. Cui, Z. Jiang, Z. Yan, G. Xing, J. Niu, and Z. Ouyang, "Vips: real-time perception fusion for infrastructure-assisted autonomous driving," in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, 2022, pp. 133–146.

[16] Y. He, L. Ma, Z. Jiang, Y. Tang, and G. Xing, "Vi-eye: semantic-based 3d point cloud registration for infrastructure-assisted autonomous driving," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, 2021, pp. 573–586.

[17] S. Seebacher, B. Datler, J. Erhart, G. Greiner, M. Harrer, P. Hrassnig, A. Präsent, C. Schwarzl, and M. Ullrich, "Infrastructure data fusion for validation and future enhancements of autonomous vehicles' perception on austrian motorways," in *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*. IEEE, 2019, pp. 1–7.

[18] M. Gabb, H. Digel, T. Müller, and R.-W. Henn, "Infrastructure-supported perception and track-level fusion using edge computing," in *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019, pp. 1739–1745.

[19] E. Arnold, M. Dianati, R. de Temple, and S. Fallah, "Cooperative perception for 3d object detection in driving scenarios using infrastructure sensors," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 3, pp. 1852–1864, 2020.

[20] A. Zeng, M. Chen, L. Zhang, and Q. Xu, "Are transformers effective for time series forecasting?" in *Proceedings of the AAAI conference on artificial intelligence*, vol. 37, no. 9, 2023, pp. 11 121–11 128.

[21] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *Proceedings of the IEEE*, 2023.

[22] G. Jocher, A. Stoken, J. Borovec, NanoCode012, ChristopherSTAN, L. Changyu, Laughing, tkianai, A. Hogan, lorenzomammana, yxNONG, AlexWang1900, L. Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, F. Ingham, Frederik, Guilhen, Hatovix, J. Poznanski, J. Fang, L. Yu, changyu98, M. Wang, N. Gupta, O. Akhtar, PetrDvoracek, and P. Rai, "ultralytics/yolov5," 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4154370

[23] B. Li, B. Wu, J. Su, and G. Wang, "Eagleeye: Fast sub-net evaluation for efficient neural network pruning," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer, 2020, pp. 639–654.

[24] H. Dong, Y. Xie, X. Zhang, W. Wang, X. Zhang, and J. He, "Gpsmirror: Expanding accurate gps positioning to shadowed and indoor regions with backscatter," *arXiv preprint arXiv:2304.07572*, 2023.

[25] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[26] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[27] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[28] H. Yang, J. Shi, and L. Carlone, "Teaser: Fast and certifiable point cloud registration," *IEEE Transactions on Robotics*, vol. 37, no. 2, pp. 314–333, 2020.

[29] Y. Xu, X. Zhou, S. Chen, and F. Li, "Deep learning for multiple object tracking: a survey," *IET Computer Vision*, vol. 13, no. 4, pp. 355–368, 2019.

[30] Y. Park, L. M. Dang, S. Lee, D. Han, and H. Moon, "Multiple object tracking in deep learning approaches: A survey," *Electronics*, vol. 10, no. 19, p. 2406, 2021.

[31] G. Ciaparrone, F. L. Sánchez, S. Tabik, L. Troiano, R. Tagliaferri, and F. Herrera, "Deep learning in video multi-object tracking: A survey," *Neurocomputing*, vol. 381, pp. 61–88, 2020.

[32] H. Wu, J. Xu, J. Wang, and M. Long, "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting," *Advances in Neural Information Processing Systems*, vol. 34, pp. 22 419–22 430, 2021.

[33] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[34] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China: IEEE, May 9-13 2011.

[35] 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/CVF conference on computer vision and pattern recognition*, 2019, pp. 12 697–12 705.
[36] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (fpfh) for 3d registration," in *2009 IEEE international conference on robotics and automation*. IEEE, 2009, pp. 3212–3217.
[37] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.
[38] Q. Liu, T. Han, J. L. Xie, and B. Kim, "Real-time dynamic map with crowdsourcing vehicles in edge computing," *IEEE Transactions on Intelligent Vehicles*, 2022.
[39] F. Ahmad, H. Qiu, R. Eells, F. Bai, and R. Govindan, "Carmap: Fast 3d feature map updates for automobiles." in *NSDI*, 2020, pp. 1063–1081.
[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[41] W. Zhang, Z. He, L. Liu, Z. Jia, Y. Liu, M. Gruteser, D. Raychaudhuri, and Y. Zhang, "Elf: accelerate high-resolution mobile deep vision with content-aware parallel offloading," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, 2021, pp. 201–214.
[42] C. Creß and A. C. Knoll, "Intelligent transportation systems with the use of external infrastructure: A literature survey," *arXiv preprint arXiv:2112.05615*, 2021.
[43] R. Xu, H. Xiang, Z. Tu, X. Xia, M.-H. Yang, and J. Ma, "V2x-vit: Vehicle-to-everything cooperative perception with vision transformer," in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXIX*. Springer, 2022, pp. 107–124.
[44] T.-H. Wang, S. Manivasagam, M. Liang, B. Yang, W. Zeng, and R. Urtasun, "V2vnet: Vehicle-to-vehicle communication for joint perception and prediction," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer, 2020, pp. 605–621.
[45] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *The 25th annual international conference on mobile computing and networking*, 2019, pp. 1–16.
[46] Z. Yang, X. Wang, J. Wu, Y. Zhao, Q. Ma, X. Miao, L. Zhang, and Z. Zhou, "Edgeduet: Tiling small object detection for edge assisted autonomous mobile vision," *IEEE/ACM Transactions on Networking*, 2022.

Yu Zhao is currently a master's student in University of Science and Technology of China. He received his B.S. degree from Chongqing University in 2021. His research interests include edge computing, autonomous driving and reinforcement learning.

He Sun received his B.S. degree from the School of Computer Science and Technology and B.A. degree from the School of Foreign Languages, Qingdao University, Qingdao, China in 2020. He is currently pursuing the Ph.D. degree on computer science with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, China. His research interests include reinforcement learning, online-learning theory, mobile computing system, and privacy preservation.

Mingjun Xiao (Member, IEEE) received the PhD degree from the University of Science and Technology of China in 2004. He is currently a professor with the School of Computer Science and Technology, University of Science and Technology of China. He has authored or coauthored more than 100 papers in referred journals and conferences, including TSC, TMC, TC, TPDS, TON, TKDE, INFOCOM, and ICDE, etc. His research interests include mobile crowdsensing, edge computing, federated learning, auction theory, data security and privacy. He was a TPCmember of INFOCOM'23, INFOCOM'22, IJCAI'22, INFOCOM'21, IJCAI'21, and so on. He is on the reviewer board of several top journals, such as TSC, TMC, TON, TPDS, TVT, and TC.

Jie Wu is Laura H. Carnell Professor at Temple University and the Director of the Center for Networked Computing (CNC). He served as Chair of the Department of Computer and Information Sciences from the summer of 2009 to the summer of 2016 and Associate Vice Provost for International Affairs from the fall of 2015 to the summer of 2017. Prior to joining Temple University, he was a program director at the National Science Foundation and was a distinguished professor at Florida Atlantic University, where he received his Ph.D. in 1989. His current research interests include mobile computing and wireless networks, routing protocols, network trust and security, distributed algorithms, applied machine learning, and cloud computing. Dr. Wu regularly published in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Service Computing and Journal of Computer Science and Technology. Dr. Wu is/was general chair/co-chair for IEEE DCOSS'09, IEEE ICDCS'13, ICPP'16, IEEE CNS'16, WiOpt'21, ICDCN'22, IEEE IPDPS'23, and ACM MobiHoc'23 as well as program chair/cochair for IEEE MASS'04, IEEE INFOCOM'11, CCF CNCC'13, and ICCCN'20. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a Fellow of the AAAS and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award. He is a Member of the Academia Europaea (MAE).

Junjie Shao is currently a master's student at the University of Science and Technology of China. He received his B.E. degree from the University of Science and Technology of China in 2021. His research interests include edge computing, game theory, and intelligent transportation.

Jinbo Cai is currently a master's student at the University of Science and Technology of China. He received his B.E. degree from Wuhan University in 2021. His research interests edge computering, mobile crowdsensing and intelligent transportation.