# TrajMatch: Toward Automatic Spatio-Temporal Calibration for Roadside LiDARs Through Trajectory Matching

Haojie Ren, Sha Zhang, Sugang Li, Yao Li, Xinchen Li, Jianmin Ji, Yu Zhang, and Yanyong Zhang, *Fellow, IEEE*

***Abstract*— Recently, deploying sensors such as LiDARs on the roadside to monitor the passing traffic and assist autonomous vehicle perception has become popular. However, unlike autonomous vehicle systems, roadside sensor systems involve sensors from different subsystems, resulting in a lack of synchronization in both time and space between the sensors. Calibration is a critical technology that enables the central server to fuse data generated by different location infrastructures, which vastly improves sensing range and detection robustness. Regrettably, existing calibration algorithms frequently assume that LiDARs have significant overlap or that temporal calibration has already been achieved. However, since these assumptions do not always hold in real-world scenarios, the calibration results obtained from existing algorithms are frequently unsatisfactory. In this paper, we propose TrajMatch - the first system that can automatically calibrate roadside LiDARs in both time and space. The main idea is to automatically calibrate the sensors based on the result of the detection/tracking task, rather than relying on extracting special features. Furthermore, we propose a novel mechanism for evaluating calibration parameters that align with our algorithm, and we demonstrate its effectiveness through experiments. This mechanism can also guide parameter iterations for multiple calibrations, further enhancing the accuracy and efficiency of our calibration method. Finally, to evaluate the performance of TrajMatch , we collected two datasets, one simulated dataset LiDARnet-sim 1.0 and one real-world dataset. The experimental results show that TrajMatch can achieve a spatial calibration error of less than 10*cm* and a temporal calibration error of less than 1.5*ms*.**

***Index Terms*— Roadside traffic monitoring, spatiotemporal calibration, trajectory matching.**

## I. Introduction

IT has become a recent trend to have smart traffic systems involve cameras/LiDARs at important locations especially at busy intersections to closely monitor the traffic for improved traffic flow, driving safety, and energy efficiency [1]. Specifically, LiDARs are increasingly deployed for such purposes due to their ability to provide accurate ranging information that is particularly important for traffic monitoring. To address the issue of single LiDAR obstruction(such as in Figure 5) and enhance the accuracy [2] and sensing range of traffic scenes [3], it has become a common practice to deploy multiple LiDARs in a collaborative manner. This approach also helps to improve the system's robustness and redundancy. We show an intersection scene in Figure 1, where intelligent roadside infrastructures equipped with computing and sensing units capture real-time environment data and perform necessary calculations such as detection and tracking, finally, they will transfer the data to a smart traffic server(STS), where their data will be fused.

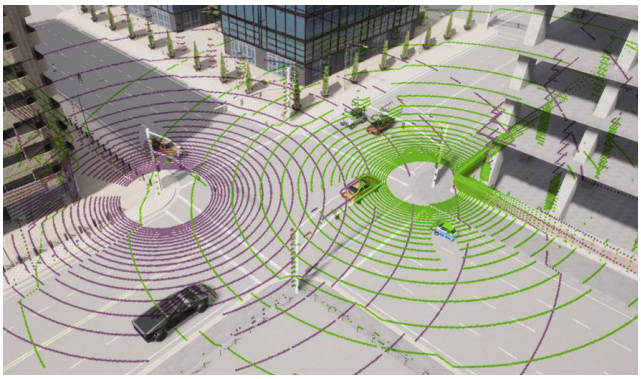


Fig. 1. An example deployment of the roadside traffic monitoring system. The point clouds from the two LiDARs diagonally deployed in a 4-way intersection.

This work addresses the problem of spatio-temporal calibration for sensors installed on the infrastructure, which allows for the effective integration of data from multiple LiDARs at various locations, thereby providing a more comprehensive traffic overview.

We note that there are few studies on calibrating roadside multiple LiDARs. [4] Although there exist some methods to calibrate LiDARs, they don't work for roadside because of their own limitations. For the space dimension, most studies have focused on the calibration of the two sensors in space, and assumed that the time between sensors has been synchronized [4], [5], [6]. There are mainly two types of methods of space calibration, target-based, and targetless-based. The

Manuscript received 16 November 2022; revised 22 April 2023 and 12 June 2023; accepted 15 June 2023. Date of publication 27 July 2023; date of current version 1 November 2023. This work was supported by the Chinese Academy of Sciences Frontier Science Key Research Project under Grant ZDBS-LY-JSC001. The Associate Editor for this article was F. Xia. *(Corresponding author: Yanyong Zhang.)*

Haojie Ren, Sha Zhang, Yao Li, Xinchen Li, Jianmin Ji, Yu Zhang, and Yanyong Zhang are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China (e-mail: rhj@mail.ustc.edu.cn; zhsh1@mail.ustc.edu.cn; zkdly@mail.ustc.edu.cn; jianmin@ustc.edu.cn; yuzhang@ustc.edu.cn; yanyongz@ustc.edu.cn).

Sugang Li is with Google Cloud, Sunnyvale, CA 94089 USA (e-mail: sugangli@google.com).

Digital Object Identifier 10.1109/TITS.2023.3295757

target-based methods require identifiable objects (such as checkerboards, polygonal boards, and apriltags), and estimate the relative pose between sensors by aligning the target positions observed on each sensor [5], [7], [8]. These methods require pricey and lengthy manual operations, and cannot be applied to large-scale deployment. As for the targetless methods, they usually extract the features from the environment, and then match the features based on their similarity to construct the constraints required for calibration. The methods of feature extraction face two common challenges in roadside scenarios: 1. Road-side point clouds are relatively sparse, making it difficult to extract geometric features; 2. In traffic scenes, there are many repeated structures, which makes feature matching difficult [9].

For the time dimension, other studies mainly discussed the problem of asynchronous timestamps caused by different frequencies between different sensors [10], [11]. Reference [12] considers the problem of time offset between the camera and the radar.

With the above challenges in mind, this paper proposes TrajMatch - the first system that can automatically calibrate LiDARs on the roadside both temporally and spatially. The primary concept is to use trajectory information of moving objects, provided by the point cloud-based object detection/tracking module, to identify correspondences between trajectories captured by different LiDARs.

The main contributions of this paper are as follows:

1) We propose TrajMatch , the first system that can automatically achieve accurate spatio-temporal calibration for roadside LiDARs. The algorithm uses the results of the object detection/tracking module to perform LiDAR calibration, without extracting special features for the calibration task separately.
2) We propose a continuous calibration method that can continue to refine the calibration as cars pass by. In particular, our system is able to self-assess the quality of each calibration session and integrate the well-performed sessions to improve the calibration accuracy continuously.
3) Our system can achieve centimeter-level spatial calibration accuracy as well as millisecond-level temporal calibration accuracy without relying on any external information about the deployment configuration and the environment.

## II. Related Work

The spatio-temporal calibration of roadside LiDARs faces several important technical challenges due to the properties of the system deployment. Below we explain the challenges in detail.

### A. Spatial Calibration

Most studies focus on calibrating two sensors in space and assume that the two sensors are synchronized in time. However, even for spatial synchronization only, there are still problems.

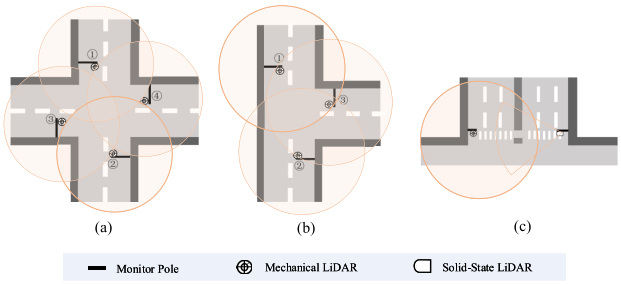



Fig. 2. Typical deployment scenarios for roadside traffic monitoring: (a) a 4-way intersection, (b) a 3-way intersection, and (c) a sidewalk.

The first challenge we face is that there are usually significant differences in the positioning between LiDARs. Figure 1 shows a scenario of roadside deployment of multiple LiDARs. In this case, the width of the 4-way intersection is $60m$. In Figure 2 we enumerate three deployment scenarios: (a) a 4-way intersection; (b) a 3-way intersection; (c) a sidewalk. In these scenarios, there are significant transformations among the LiDARs. The ICP [13] is the most commonly used method for space calibration and has many variants, but these algorithms are heavily dependent on close initial guess for the unknown transformation. NDT [14] is a method that represents point clouds by a set of Gaussian distributions for quick alignment. But similar to ICP, NDT also highly depends on initial guess between two point clouds [15]. These algorithms can work well for scenarios where there is a relatively small transformation between consecutive frames, such as in SLAM, but may not be suitable for roadside scenes.

Another issue is that when a LiDAR is installed on the roadside, its sensing range becomes very large, resulting in extremely sparse point clouds that make it difficult to extract local features such as lines and surfaces. Even an 80-line LiDAR with high line density has a sensing range of only 160m * 160m, and the point cloud is still very sparse. Many existing calibration methods rely on discriminative hand-crafted features in the environment such as curves [16], intersection lines [17], and adaptive covariance [18]. However, it is hard to find these features in scenarios where LiDAR sensors are installed on the side of the road, making these methods not applicable in such cases.

Furthermore, there are some target-based methods, such as ordinary boxes [7], apriltags [5], and reflective conical [19]. But these methods require a significant amount of manual effort, which is not feasible for large-scale deployment in roadside systems.

### B. Temporal Calibration

Regarding the time aspect, other researchers have mainly focused on the problem of temporal synchronization caused by the different sampling frequencies of sensors. Reference [10] considered the synchronization between the camera and the LiDAR. Reference [11] proposed a multi-threaded time synchronization method that stores the data in a buffer and updates it in real time. Reference [20] employed an algorithm combining Kalman filtering and Lagrangian interpolation to

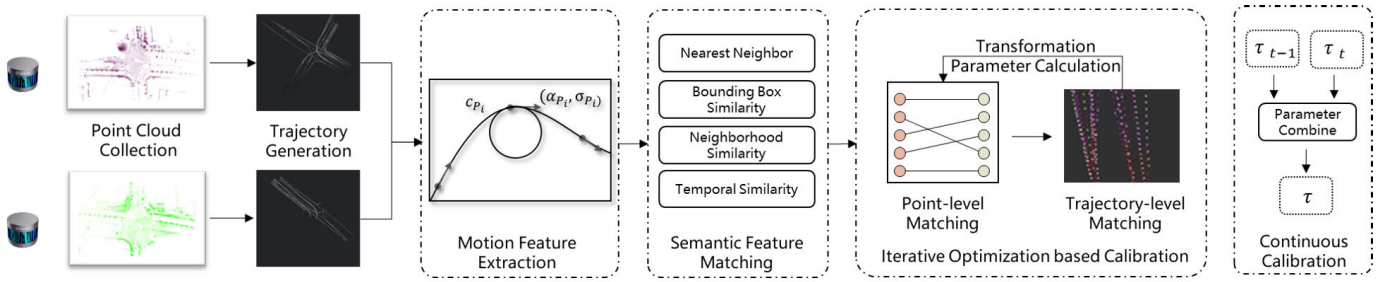



Fig. 3. Framework of our TrajMatch system. In this paper, we exploit a system for performing spatio-temporal calibration for roadside LiDARs. Our system takes LiDARs detection and tracking results as input, and consists of the following main modules: (1) motion feature extraction (**local**), (2) semantic feature matching (**middle**), (3) iterative optimization (**global**), and (4) continuous calibration. Finally, we propose a strategy to adjust the errors over continuous calibration.

achieve temporal synchronization between the data from the two sensors.

In roadside systems, another key issue is that sensors are affiliated with different systems and work separately, operating on different timelines, resulting in data from each sensor at the same timestamp being offset by several seconds [21]. Reference [12] used a motion-fitting method to solve the problem of time synchronization between the camera and the millimeter-wave radar.

## III. System Overview

The spatio-temporal calibration between roadside LiDARs intends to obtain the transformation parameters between LiDARs and align their timelines. Given the detection/tracking results from the point clouds, our task is to find a suitable transformation that facilitates the fusion of the data from different LiDARs coherently. Our system supports two data fusion modes. First, once calibrated, the raw point clouds can be directly merged to form a unified point cloud. Second, the transformation calculated by our algorithm can also support fusing the high-level information, such as the features extracted by deep learning models, or even the object list and their trajectories. The fusion of raw point clouds can maximally retain the information, which may potentially lead to fine-grained enhancements such as point cloud segmentation. However, this fusion method requires expensive hardware support, especially in terms of network communication. Taking the RS-Ruby_Lite as an example, that outputs around $1,440,000$ points per second, and around $30M$ of point cloud data are generated per second. As a result, in this work, we take the second approach and fuse the object-level information, which is a practical and effective approach given today's infrastructure capability.

Towards this goal, we propose TrajMatch whose framework is shown in Figure 3. Our calibration method does not assume any initial state of the system, neither the transformation parameters nor the time deviation between LiDARs. As a stateless system, we align the LiDARs both spatially and temporally at the same time, by going through an efficient iterative optimization process – matching the object's positions, calculating the transformation parameters accordingly, and then aligning the trajectories with the parameters. We strive to make progress in each step, and in an iterative fashion converge to the optimal state.

## IV. Spatio-Temporal Synchronization Method

As shown in Figure 3, TrajMatch consists of the three main steps: (i) point cloud based trajectory generation, (ii) trajectory motion feature extraction, (iii) semantics aware position matching, (iv) iterative optimization based calibration, and a method to further improve the success rate through continues calibrations. Below we discuss these steps one by one, using the calibration of two LiDARs as the running example.

### A. Problem Formulation

First of all, we formally define the LiDAR spatio-temporal calibration problem. We suppose $P = \{p_1, p_2, p_3, \ldots\}$ and $Q = \{q_1, q_2, q_3, \ldots\}$ are two groups of 4D point clouds with time dimension [22]. Points in $P$ and $Q$ have similar forms:

$$p_i = \left\{x_i^p, y_i^p, z_i^p, t_i^p\right\}, \quad q_j = \left\{x_j^q, y_j^q, z_j^q, t_j^q\right\}, \tag{1}$$

where $x$, $y$, and $z$ represent the coordinates of the three dimensions in their respective coordinate systems, and $t$ is the timestamp. Especially, before calibration, $P$ and $Q$ have different coordinate systems. For example, the $x$ axis of $P$ may correspond to the $y$ axis of $Q$. Additionally, there may be a certain deviation in the time dimension $t$ for the point clouds collected at the same time.

In this work, we define the problem in the 4D space, for a point $p = \left[x, y, z, t\right]^T$, the coordinate transformation can be defined as:

$$p' = \begin{bmatrix} R_3 & 0 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \\ d_z \\ d_t \end{bmatrix} = R \times p + T, \tag{2}$$

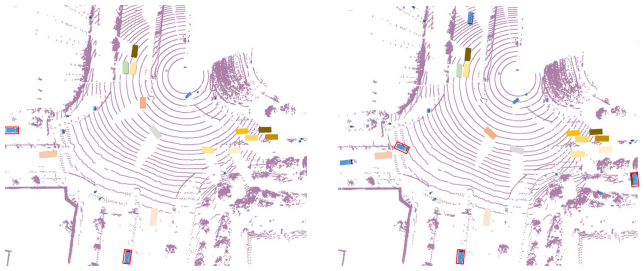
where $d_t$ represents the offset in time, and $\mathbf{R}$ and $\mathbf{T}$ denote the rotation matrix and the translation vector between LiDARs, respectively.

The spatio-temporal calibration problem for two 4D point clouds $P$ and $Q$ can be described as:

$$R, T = \arg\min_{R,T} \frac{1}{|P|} \sum_{i=1}^{|P|} \| p_i - (R \times q_i + T) \|^2, \tag{3}$$

where $p_i$ and $q_i$ are the corresponding points in the $P$ and $Q$.

In this way, we transform the LiDAR calibration problem into an optimization problem in which we try to find the



(a) (b)

Fig. 4. An example detection and tracking result from two adjacent point cloud frames. The detection algorithm is responsible for outputting the bounding box of the object, and the tracking algorithm is responsible for correlating the objects across frames. We use the same color to represent the same object, and we annotate mistakenly detected objects with red boxes.

rotation parameter $R$ and the translation vector $T$, with the objective of minimizing the matching error between the data of the two LiDARs in both time and space. We call the collection of $R$ and $T$ as the system's transformation parameters.

### B. Point Cloud Based Trajectory Generation

Different from the calibration algorithms that use raw point clouds as input, we use the point cloud detection/tracking results over a period of time $T$. We note that the choice of $T$ should be aware of the time offset between the two LiDARs.

As soon as LiDAR generates a point cloud frame, 3D object detection is performed. Many point cloud based detection algorithms have been proposed recently, such as point-based algorithms in [23] and voxel-based algorithms in [24]. These detection algorithms identify each object, classify it into one of the traffic participant categories such as cars, trucks, pedestrians, bicycles, etc., and estimate its 3D bounding box. Figure 4 gives an example point cloud frame and the objects detected from the frame using the PointPillars algorithm [25].

After performing object detection on each frame, we then perform the multi-object tracking (MOT) task to associate each object's positions across frames to obtain the movement trajectory. In this work, we engage the AB3DMOT algorithm [26], [27] to calculate the trajectories. Figure 4 shows an example association results from two adjacent point cloud frames, in which we use the same color to denote the same car in these two frames.

Here, the detection and tracking tasks constitute the preprocessing step for our TrajMatch system.

### C. Trajectory Motion Feature Extraction

Firstly, We follow the discussion in [28] to give the mathematical definition of the trajectory.

A trajectory ($TR$) is a chronologically ordered sequence of 3D position coordinates of an object within a certain period of time, representing the motion information of an object, which is denoted by $TR = \{P_0, \ldots, P_i, \ldots, P_n\}$ $(1 \leq i \leq n)$. Here, $P_i$ is the position of the object at time $t_i$, where each position consists of a spatial coordinate set and a time stamp such as $P_i = (x_i, y_i, z_i, t_i)$. To be more precise, position $P_i$ corresponds to the object's center coordinate $((x_i, y_i, z_i))$ detected from point cloud frame $f_i$ that is captured at time $t_i$. Furthermore, we call a set of trajectories a LiDAR detects a trajectory database ($TD$), with $TD = \{TR_1, TR_2, TR_3, \ldots, TR_m\}$, where $m$ is the number of objects that are detected.

Next, we extract suitable features for trajectories, which should accurately describe the object's motion pattern on the trajectory. Considering that a trajectory consists of a list of positions, we choose to extract motion features for each position to capture a more fine-grained motion pattern. When choosing the features, we need to take into consideration the following factors. First of all, there exists an unknown coordinate transformation, the features need to be rotation-invariant and translation-invariant. This means the feature extracted by the algorithm should be consistent, regardless of how the coordinate is rotated and translated.

Therefore, we use the velocity and curvature information at each position, both rotation-invariant, to describe the motion at that position. Suppose a trajectory is composed of $n$ positions. Then a trajectory's feature vector thus consists of $n$ velocity features and $n$ curvature features.

*1) Velocity:* The velocity is an important feature for discriminating the mobility pattern. First, we note that the mean speed alone (as in [29]) is insufficient for this purpose. For example, one car may travel at a constant speed throughout the entire journey, while another car may experience changes in speed, such as acceleration, deceleration, and stops, during the journey. As a result, we engage both the mean and variance of the velocity here.

For a trajectory $TR = \{P_0, P_1, P2, \ldots, P_n\}$, we can get a sequence of velocities $\{v_0, v_1, \ldots, v_{n-1}\}$ for the trajectory positions, with $v_i = dis(P_i, P_i + 1)/(t_{i+1} - t_i)$. Then the velocity feature $V(P_i) = \{\alpha_i, \sigma_i\}$ for position $P_i$ can be calculated as:

$$\alpha(P_i) = \bar{v} = \frac{1}{2m} \sum_{j=i-m}^{j=i+m-1} v_j,$$

$$\sigma^2(P_i) = \frac{1}{2m} \sum_{j=i-m}^{j=i+m-1} v_j - \bar{v}^2,$$

where $m$ is a hyper-parameter to balance the detection/tracking noise. For example, if the trajectory obtained from the tracker has significant noises, selecting a small $m$ likely leads to large errors in the feature estimation.

*2) Curvature:* The curvature characterizes the angle of a trajectory at a certain location. Suppose there are three consecutive positions: $P_{i-1}$, $P_i$, $P_{i+1}$. The curvature at position $P_i$ is calculated as:

$$c(P_i) = cos\theta = \frac{\overrightarrow{P_i P_{i-1}} \cdot \overrightarrow{P_i P_{i+1}}}{\left|\overrightarrow{P_i P_{i-1}}\right| \left|\overrightarrow{P_i P_{i+1}}\right|}, \quad \theta \in (0, \pi), \tag{4}$$

where $\theta$ denotes the angle between the two segments $\overrightarrow{P_{i-1} P_i}$ and $\overrightarrow{P_i P_{i+1}}$. We note that the smoother the trajectory is at position $P_i$, the closer $\theta$ is to $\pi$, and the smaller $c(P_i)$ is.

### D. Semantics Aware Position Matching

In this step, we establish the correspondences between two positions from different trajectories. First, we obtain the matching relationship of a series of positions based on the information of the position itself, and then we will consider the distribution of its neighbors to remove erroneous matches.

*1) Motion-based Position Matching:* we utilize the Euclidean distance to measure the distance between two position motion features. The distance between features $A = \{c_A, \alpha_A, \sigma_A\}$ and $B\{c_B, \alpha_B, \sigma_B\}$ is calculated as:

$$d(A, B) = \lambda_c |c_A - c_B| + \lambda_\alpha |\alpha_A - \alpha_B| + \lambda_\sigma |\sigma_A - \sigma_B|, \tag{5}$$

where the values of $\lambda_c, \lambda_\alpha, \lambda_\sigma$ can be adjusted to give different weights to each of the features attributes. For example, between the velocity mean and variance, the variance $\sigma$ is more susceptible to noises. When the detection/tracking noise in the system is higher than a threshold, we can reduce $\lambda_\sigma$ to lower its weight.

Based on the above distance calculation, we next perform motion-based position matching. Suppose we have two LiDARs at an intersection, one with trajectory database $P$ and the other $Q$. Then for each position $p_i \in P$, we find the closest position $q_j$ in $Q$. If the $d(p_i, q_j) < d_{th}$, we add the $(p_i, q_j)$ pair to the matched set.

By going through the position matching algorithm, we obtain a large number of position pairs in which a position may have several matches. However, we find that many of these pairs are actually "false" matches. False matches can happen for the following reasons. Firstly, restricted by traffic rules, the speed and direction of movement of vehicles are mostly similar. Secondly, limited by the vehicle kinematics model, the speed and curvature of the vehicle usually change slowly [30], especially compared to the LiDAR sampling rate. Thirdly, there are errors in point cloud detection/tracking algorithms. All these reasons attribute to the high false match ratio.

As an example, in a typical scenario with 2 LiDARs deployed at a 4-way intersection (see Figure 1), we obtain 875 matched position pairs, among which, only at most 175 pairs are actually valid.

*2) Semantics Aware Position Matching:* We argue that a large number of false matches can be filtered out if we take into consideration the semantic information of the object, which has been generated in the earlier object detection and tracking tasks. In this work, we propose to explore the addition of semantic information to enhance our position matching:

- *Mutually Nearest Neighbor*. This is a simple observation. If positions $P_a \in TD_a$ and $P_b \in TD_b$ are a true match, then $P_a$ is the closest to $P_b$ in $TD_a$, and $P_b$ is the closest to $P_a$ in $TD_b$.
- *Bounding Box Similarity*. If positions $P_a \in TD_a$ and $P_b \in TD_b$ are a true match, then their associated object bounding boxes (generated by the detector) should be similar to each other, including the box length, width, and height.
- *Neighbor Count Similarity*. If positions $P_a \in TD_a$ and $P_b \in TD_b$ are a true match, then the number of neighboring objects within a certain area is the same.
- *Neighborhood Distribution Consistency*. If positions $P_a \in TD_a$ and $P_b \in TD_b$ are a true match, the distribution of the surrounding positions is similar and remains so in the adjacent frames. Based on this, we can construct a time-domain histogram to capture the distribution.

With this semantic information taken into consideration, we can effectively filter out false matches. In the same example discussed above, the ratio of true matches increases from 22% to 61% after filtering.

### E. Iterative Optimization Based Calibration

Next, we considered the overall distribution of the data and employ an iterative approach (similar to ICP [13]) to optimize the position matches and to achieve accurate calibration at the same time. The optimization approach involves three main operations: position matching, transformation parameter calculation, and trajectory matching. Namely, our optimization works as follows:

1) $S1$*: Transformation Parameter Calculation.* As discussed in Sec. II, with a list of matched position pairs as the geometry constraints, we can calculate the transformation parameters by solving Eqn. 3. Then the key to correct transformation parameters is to have a set of accurately matched position pairs.
2) $S2$ : *Trajectory Matching.* Based upon the transformation parameters, we can convert each position's coordinates into a common system. With each position under the common coordinate system, we can now calculate the distance between two trajectories. In particular, we can obtain the matched trajectory pairs from matched position pairs and the trajectory information from the tracker. Meanwhile, we can also calculate the distance between matched trajectories. If this distance has become lower than the preset threshold, then we have achieved satisfactory calibration results and can terminate the optimization process.
3) $S3$ : *Position Matching.* With the matched trajectory pairs and each trajectory's position sequence, we can further refine our matched position pairs by adding missed matches and removing false ones. Then we go back to $S1$, and continuously optimize until the distances between the matched trajectories are small than a threshold or the iteration exceeds the maximum number of times.

### F. Continuous Calibration

After discussing our calibration process, we next discuss how the system assesses the calibration effect and how we combine the effect of multiple calibrations. As such, we can achieve continuous calibration for the roadside LiDARs.

Suppose our calibration results give the transformation parameters $\tau = (R, T)$.

In an actual system, we perform LiDAR calibration at a certain frequency. For example, we can perform calibration

as soon as an object is detected, or every $k$ objects. In this process, we keep the current transformation parameters $\tau_t$ with a certain score $s_t$. In addition, suppose the latest calibration process gives $\tau_{t-1}$ and $s_{t-1}$.

Then we adjust the transformation parameters as to achieve continuous calibration:

$$\tau = \frac{s_{t-1}}{s_{t-1}+s_t} \times \tau_{t-1} + \frac{s_t}{s_{t-1}+s_t} \times \tau_t. \tag{6}$$

## V. Performance Evaluation

In order to thoroughly evaluate TrajMatch , we collect a simulated dataset LiDARnet-sim 1.0 and a real-world dataset. The details of the dataset can be found in Section V-A. We define the evaluation metrics in Section V-B. In Section V-C, we show the evaluation results.

### A. Evaluation Dataset

To validate the proposed method, we build two datasets, one real-world and one simulated.

*1)* LiDARnet-sim 1.0 *:* Since there is no public dataset to support the research on roadside LiDAR calibration, we construct LiDARnet-sim 1.0 using the CARLA [31] and SUMO [32] co-simulation platform. Utilizing a simulated dataset offers several advantages. Firstly, it provides ground truth information such as the translation of LiDARs and object detection/tracking information, which is crucial for the accurate evaluation of the proposed method. Secondly, generating various traffic scenes in a simulated environment is much easier and more efficient, allowing for a more comprehensive evaluation of the system.

Specifically, we simulate the traffic flow based on Sumo and then record the sensor data based on Carla. The dataset was constructed by considering several factors. The first factor is the traffic flow, in which we add a different number of cars to the map to achieve varying traffic densities. The second factor is the LiDAR type, including a 32-line LiDAR with a sensing range of 50m and an 80-line LiDAR with a sensing range of 150m. The third factor involved different traffic scenarios, such as three-way intersections, crossroads, and other complex traffic situations.

*2) Real-World Dataset:* To validate the actual usability of the algorithm in a real-world scenario, we have chosen to record a dataset at a seven-lane intersection. Figure 5 shows the point cloud map of the recorded dataset scene. We mainly recorded datasets for two deployment scenarios. (1) diagonally at the intersection with RS-Ruby Lite (80 beams); (2) on both sides of the sidewalk with RS-LiDAR-32 (32 beams) (Figure 5). The LiDARs recorded the dataset at a frequency of 10 Hz.

Furthermore, in addition to LiDAR deployment and models, we also took into account different traffic scenarios. Apart from the typical intersection scenario, we also considered two specific traffic scenarios: 1) a dense pedestrian crossing scenario; and 2) a scenario where one LiDAR is occluded when a bus passes through.

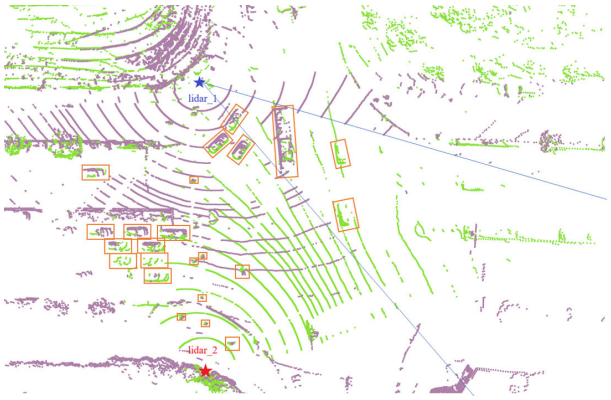



Fig. 5. We have provided an example of registration results in a real scene. Additionally, in the given figure, we can see that when the bus passes, certain areas on the right side of $LiDAR_1$'s point cloud are blocked. As we can see, $LiDAR_2$ plays a vital role in supplementing the missing data and providing a more complete view of the scene.

### B. Metrics and Baseline Algorithms

*1) Metrics:* To compare the performance with other calibration algorithms, we adopt the metrics defined in [33], which are also used in [9] and [34]. Firstly, the relative rotation error (**RRE**) is:

$$angle = F\left(R_T^{-1} \cdot R_E\right),$$
$$RRE = \sum_{i=1}^{3} |angle(i)|, \tag{7}$$

where $R_T$ and $R_E$ are the rotation matrices of the ground-truth transformation and the estimated transformation, respectively. $F(\cdot)$ transforms a rotation matrix to three Euler angles. Secondly, the relative translation error (**RTE**) is defined as:

$$RTE = ||t_T - t_E||, \tag{8}$$

where $t_T$ and $t_E$ are the translation vector of the ground-truth transformation and the estimated transformation, respectively.

In addition, we define a calibration session as a 'success' when the RRE and RTE are below the predefined threshold [35]. In this paper, we choose the threshold as $1m$, which is sufficient for the requirements of autonomous driving navigation [36]. The **success rate** is then defined as the average rate at which a session is successful.

To test the algorithm's ability to time-synchronize, we used timing offset error (**TOE**). The TOE is defined as:

$$TOE = |T_T - T_E|, \tag{9}$$

where $T_T$ and $T_E$ are the ground truth and estimated timing offsets, respectively.

*2) Baseline Algorithms:* We compare TrajMatch with four baseline LiDAR calibration algorithms: (i) ICP [13], in which the calibration results are achieved through iterative optimization, (ii) NDT [14] in which the calibration is determined based on the similarity between the probability density functions (PDFs) of the point clouds; (iii) K-4PCS [37] in which key points are extracted to improve the efficiency of the optimization process; (iv) SAC-IA [38], in which the calibration is based on local geometry features around a

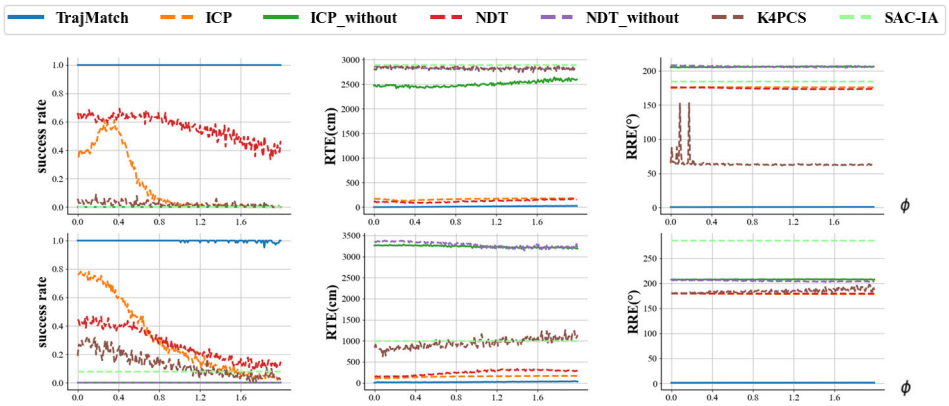



Fig. 6. We show here the effect of the algorithm in two different scenarios. The average value is used here, and our algorithm is filtered using our evaluation tool. In the first row, we put the LiDARs on the diagonal of the intersection, and in the second row, we put them on the three-way intersection.

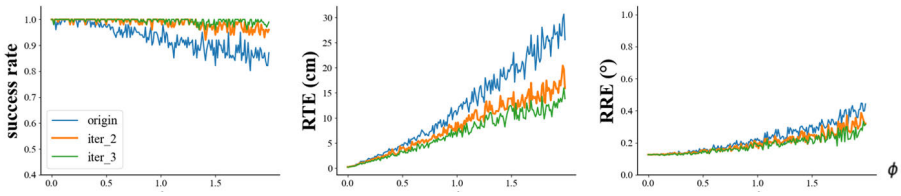



Fig. 7. The spatial calibration results after 1, 2, and 3 passes. For RTE and RRE.

point, called Fast Point Feature Histograms (FPFH). These algorithms are implemented based on the Point Cloud Library (PCL) [39]. We have carefully adjusted their parameter values to ensure they achieve the best performance with our datasets. Among these algorithms, only SAC-IA is based on raw point clouds, and the others are based on detection/tracking results.

### C. Calibration Results

*1) Results With Simulated Dataset:* We first show the results with the simulated dataset.

*a) Spatial calibration vs deployment setting:* In this set of experiments, we compare TrajMatch with the baseline algorithms in terms of the spatial calibration results. To simulate different detection errors, we add the noise that follows the Gaussian distribution in the range $(0, \phi)$ to the ground-truth trajectory. We vary $\phi$ from 0 to 2m. According to the leaderboard of nuScenes [40], this value is significantly larger than the ranging error based on LiDARs detection, which is around 0.2m. By choosing a larger value, we make sure that our evaluation covers the worse case scenarios. In addition, since the performance of ICP and NDT is largely dependent on the choice of the initial values, we test them under two situations, one with a hand-picked initial value and the other one without an initial value.

The first row of Figure 6 shows the success rate, RTE, and RRE at a 4-way intersection. The LiDARs are installed diagonally at the intersection with a distance of 28.8m, as shown in Figure 2(a). The number of LiDAR beams is 32, the detection distance is 50m. On these three metrics, our results have been consistently the best. Even when $\phi$ is larger than 1.2m, TrajMatch can still achieve a success rate above 95%. We also tested using 80-line LiDAR and the results were similar to 32-line, so we didn't put result pictures here. The second row shows the result at a three-way intersection, and we came to a similar conclusion.

TABLE I
THE EFFECTIVENESS OF OUR PARAMETER ASSESSMENT

| | RTE(cm) | RRE(°) | score |
|---|---|---|---|
| 1 | 4.117 | 1.550 | 0.848 |
| 2 | 5.373 | 1.573 | 0.841 |
| 3 | 14.759 | 1.568 | 0.824 |
| 4 | 17.087 | 1.520 | 0.823 |
| 5 | 3569.470 | 183.878 | 0.00 |

TABLE II
SPATIAL CALIBRATION RESULTS WITH DIFFERENT TRAFFIC FLOW. HERE, WE REPORT THE MEDIAN VALUES FOR RRE AND RTE

| Scenarios | Number of cars | RRE(○) | RTE(cm) | success rate (%) |
|---|---|---|---|---|
| four-way intersection | 50 | 2.55 | 3.58 | 99 |
| | 100 | 0.13 | 2.67 | 99 |
| | 200 | 2.53 | 2.08 | 98 |

Furthermore, we demonstrate the effectiveness of the evaluation mechanism, through which we can easily catch these outliers and discount them. Table I summarizes the scores for different cases and we are able to easily differentiate the outlier (with a score of 0) and other normal cases (with a score higher than 0.8).

*b) Spatial calibration vs traffic flow:* In the second set of experiments, we test the effect of the traffic flow. Specifically, we simulate different traffic flows by randomly placing 50, 100, and 200 vehicles in the city. This parameter affects the number of cars that pass by our intersection and tests our

TABLE III
SPATIAL CALIBRATION RESULTS WITH DIFFERENT ROTATION ANGLE. HERE, WE REPORT THE MEDIAN VALUES FOR RRE AND RTE

| Scenarios | Rotation Angle(°) | RRE(°) | RTE(cm) | success rate (%) |
|---|---|---|---|---|
| four-way intersection | 0 | 0.04 | 2.65 | 97 |
| | 30 | 0.14 | 1.92 | 99 |
| | 60 | 0.05 | 3.08 | 98.1 |
| | 90 | 0.04 | 2.61 | 99 |
| | 120 | 0.05 | 3.02 | 99 |

TABLE IV
THE EFFECTIVENESS OF OUR PARAMETER ASSESSMENT

| Algorithm | time(s) | RRE(°) | RTE(m) |
|---|---|---|---|
| ICP | 20.76 | 179.7 | 1.07 |
| NDT | 72.05 | 175.2 | 1.3 |
| K4PCS | 1234.17 | 119.0 | 26.81 |
| SAC-IA | 1508.49 | 42.59 | 13.67 |
| TrajMatch | 81.09 | 1.5 | 0.05 |

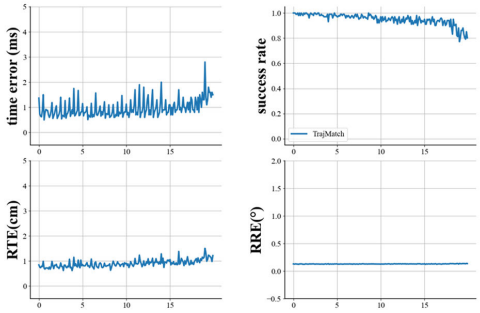



Fig. 8. Spatial-temporal calibration results with different time offset values at a 4-way intersection. The abscissa axis is the time deviation, from 0-20s.

TABLE V
RESULT OF POINTPILLARS

| AP | AMOTA | AMOTP(m) | RRE(°) | RTE(m) | error rate(%)) |
|---|---|---|---|---|---|
| 0.90 | 0.270 | 0.63 | 0.47 | 0.19 | 0.98 |

capability of dealing with multiple cars at the same time. Here, we set $\phi = 0.2m$ and the time offset $\Delta_t = 0.5s$. The results are summarized in Table II. We observe that the traffic flow does not have a noticeable impact on the performance. In all three situations, we have RTE less than $5cm$ and RRE less than 3 degree. It suggests that TrajMatch can handle different traffic flows effectively.

*c) Spatial calibration vs rotation:* In the third set of experiments, we evaluate the robustness of the system to the LiDAR rotation angle. We vary the rotation angle while fixing the rest of the parameters: the time offset is $0.5s$, and the standard deviation of the Gaussian noise $\phi = 0.2m$. The calibration results are summarized in Table III, which sufficiently demonstrate that our system is robust to different LiDAR rotation angles as we have chosen rotation-invariant features.

*d) Continuous calibration:* In the fourth set of experiments, we evaluate the effectiveness of our continuous calibration capability. That is when we perform multiple rounds of calibration processes, can we improve the calibration accuracy? We show the results in Figure 7 for the 4-way intersection case.

The results show that continuous calibration can steadily improve performance. In particular, when we have two rounds of calibration, the accuracy improves significantly compared to having a single round. However, the improvement becomes much less pronounced when we have three rounds or more. In fact, after three passes, the RTE decreases from 30cm to 10cm, and RRE decreases from 0.4° to 0.2°. This renders TrajMatch a very viable approach that is able to refine its performance as more calibration sessions are conducted.

*e) Runtime compare:* In this section, we compare the real-time performance of the algorithms, focusing specifically on spatial calibration, as the other algorithms are only designed for spatial calibration. We compared the time required for different algorithms to calibrate 100 sets of data. As the result of IV, we can see that only the ICP algorithm and NDT algorithm, which assign initial values, are faster than TrajMatch in terms of running speed, but their errors are much larger. TrajMatch incurs only a small amount of additional processing time, resulting in more accurate results. This is because they use local optimization algorithms, which can only find local optimal solutions and cannot guarantee global optimal solutions.

*f) The accurate of detector:* We present a detailed performance evaluation of the PointPillars algorithm for detection and tracking, as shown in the table V. It is worth noting that, the AMOTA of PointPillars is 0.63m, which is much higher than the current best model's AMOTA of 0.403m [40] (the lower, the better).

The PointPillars is by far not a top performing model. On the leaderboard of Kitti 3D detection models, PointPillars ranks only 236th [41]; on the Nuscenes leaderboard, it ranks 242th [40]. There are many advanced models for object detection that have been developed in recent years, such as CenterPoint [42], Voxel Rcnn [24]. These models can provide more precise detection results. Based on our analysis of the existing experimental results, we determine that our algorithm is entirely compatible with these detection models.

*g) Temporal calibration:* Finally, we report the temporal calibration result. In this set of experiments, we have $\phi = 0.2m$. We vary the time deviations of the two LiDARs from 0 to 20s. Figures 8 show the results for a 4-way intersection. As shown in Figure 8(d), When the time offset is less than 18s, we can achieve inter-frame synchronization, and the time synchronization error is less than 2ms. We believe that this is mainly due to the relative positions of vehicles that can form spatial constraints to assist our time calibration when there are multiple trajectories. Again, we believe such a temporal calibration accuracy is rather comparative for smart traffic applications.

*2) Results With Real-World Dataset:* In addition to the simulated environment, we have also recorded the dataset in the real world for testing. Figure 9 shows the combined point

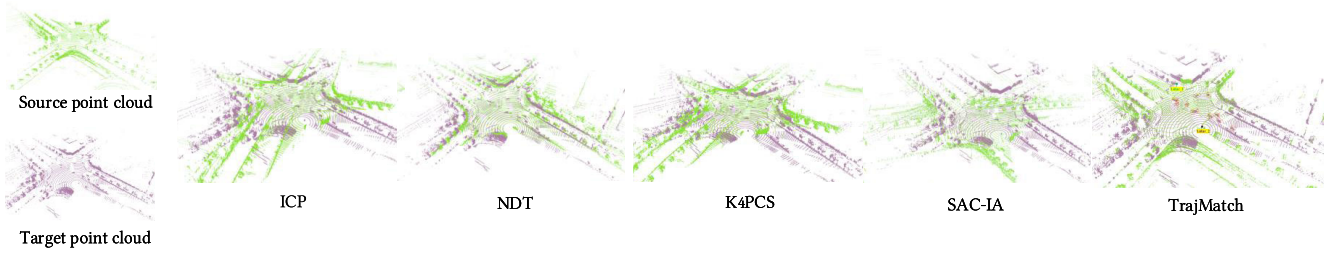



Fig. 9. Visualization results for different calibration algorithms with scenario 1. TrajMatch gives the best point cloud alignment.

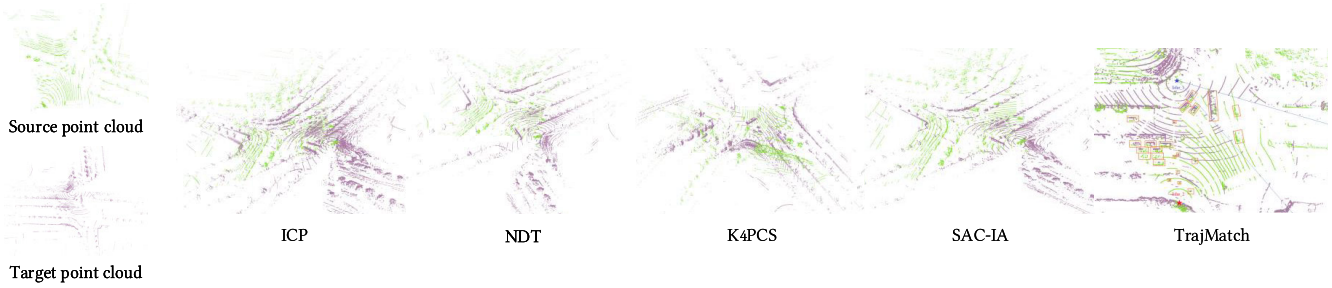



Fig. 10. Visualization results for different calibration algorithms with scenario 2. TrajMatch gives the best point cloud alignment.

TABLE VI
THE CALIBRATION RESULTS FOR REAL-WORLD TRACES IN SCENARIO 1

| | Angle error(°) | | | Transform error | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | $R_x$ | $R_y$ | $R_z$ | $T_x$ | | $T_y$ | | $T_z$ | |
| | | | | num(cm) | rate(%) | num(cm) | rate(%) | num(cm) | rate(%) |
| 1 | -0.499 | 1.787 | -0.148 | 11.7 | 0.2 | 13.5 | 3.9 | 0.0 | 0.0 |
| 2 | -0.5 | 1.785 | -0.187 | 9.9 | 0.2 | 7.1 | 2.0 | 0.0 | 0.0 |
| 3 | -0.499 | 1.787 | -0.238 | 1.2 | 0.0 | 19.1 | 5.5 | 0.0 | 0.0 |
| 4 | -0.5 | 1.78 | 0.195 | 19.3 | 0.3 | 9.5 | 2.7 | 0.0 | 0.0 |

clouds for TrajMatch and 4 other algorithms. On this dataset, all 4 baseline algorithms fail to calibrate the LiDARs and align the point clouds. TrajMatch aligns the two point clouds nicely. In this scenario, the point clouds of the same object from different LiDARs are well-matched. In addition, we also show the quantitative calibration results in Table VI. We observe that the translation error of the algorithm is less than $15cm$.

## VI. CONCLUSION

In this paper, we present TrajMatch , a system that automates the spatio-temporal calibration for roadside LiDARs. Our calibration method does not assume any initial state of the system. As a stateless system, we align the LiDARs spatially with centimeter-level accuracy and temporally with millisecond accuracy, at the same time. The calibration is achieved through an efficient iterative optimization process. In addition, we also propose a continuous calibration mechanism that can accurately assess the quality of each calibration session and update the overall calibration when the assessment is above a certain threshold.

We believe that traffic monitoring with roadside sensors will become an important component for future cities, and that our work takes the first step towards having a robust and accurate roadside perception system. Moving forward, we will continue to make effort to realize the vision. Specifically, we will investigate the calibration process for different deployment settings and different sensors. We will also carefully exploit the fused data from these sensors for improved traffic through and driving safety.

## REFERENCES

[1] M. P. Mollah, B. Debnath, M. Sankaradas, S. Chakradhar, and A. Mueen, "Efficient compression method for roadside LiDAR data," in *Proc. 31st ACM Int. Conf. Inf. Knowl. Manage.* New York, NY, USA: Association for Computing Machinery, Oct. 2022, pp. 3371–3380, doi: 10.1145/3511808.3557144.

[2] X. Cai et al., "Analyzing infrastructure LiDAR placement with realistic LiDAR simulation library," 2022, *arXiv:2211.15975*.

[3] E. Arnold, M. Dianati, R. de Temple, and S. Fallah, "Cooperative perception for 3D object detection in driving scenarios using infrastructure sensors," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 3, pp. 1852–1864, Mar. 2022.

[4] J. Kim, C. Kim, Y. Han, and H. J. Kim, "Automated extrinsic calibration for 3D LiDARs with range offset correction using an arbitrary planar board," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2021, pp. 5082–5088.

[5] Y. Xie, R. Shao, P. Guli, B. Li, and L. Wang, "Infrastructure based calibration of a multi-camera and multi-LiDAR system using apriltags," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2018, pp. 605–610.

[6] J. Quenzel, N. Papenberg, and S. Behnke, "Robust extrinsic calibration of multiple stationary laser range finders," in *Proc. IEEE Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2016, pp. 1332–1339.

[7] Z. Pusztai and L. Hajder, "Accurate calibration of LiDAR-camera systems using ordinary boxes," in *Proc. CVPR*, 2017, pp. 1–9.

[8] B. Xue et al., "Automatic calibration of dual-LiDARs using two poles stickered with retro-reflective tape," in *Proc. IEEE Int. Conf. Imag. Syst. Techn. (IST)*, Dec. 2019, pp. 1–6.

[9] 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 *Proc. MobiCom*, 2021, pp. 1–14.

[10] C. Wang, S. Liu, X. Wang, and X. Lan, "Time synchronization and space registration of roadside LiDAR and camera," *Electronics*, vol. 12, no. 3, p. 537, Jan. 2023.

[11] Y. Fu et al., "A camera–radar fusion method based on edge computing," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Oct. 2020, pp. 9–14.

[12] Y. Du, B. Qin, C. Zhao, Y. Zhu, J. Cao, and Y. Ji, "A novel spatio-temporal synchronization method of roadside asynchronous MMW radar-camera for sensor fusion," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 11, pp. 22278–22289, Nov. 2022.

[13] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 2, pp. 239–256, Feb. 1992.

[14] P. Biber and W. Straßer, "The normal distributions transform: A new approach to laser scan matching," in *Proc. IROS*, 2003, pp. 2743–2748.

[15] Z. Dong et al., "Registration of large-scale terrestrial laser scanner point clouds: A review and benchmark," *ISPRS J. Photogramm. Remote Sens.*, vol. 163, pp. 327–342, May 2020.

[16] B. Yang and Y. Zang, "Automated registration of dense terrestrial laser-scanning point clouds using curves," *ISPRS J. Photogramm. Remote Sens.*, vol. 95, pp. 109–121, Sep. 2014.

[17] I. Stamos and M. Leordeanu, "Automated feature-based range registration of urban scenes of large scale," in *Proc. CVPR*, 2003, pp. 1–9.

[18] D. Zai et al., "Pairwise registration of TLS point clouds using covariance descriptors and a non-cooperative game," *ISPRS J. Photogramm. Remote Sens.*, vol. 134, pp. 15–29, Dec. 2017.

[19] T. Kim and T. Park, "Calibration method between dual 3D LiDAR sensors for autonomous vehicles," in *Proc. SICE*, 2017, pp. 1075–1081.

[20] J. Ma, Z. Tian, Y. Li, and M. Cen, "Vehicle tracking method in polar coordinate system based on radar and monocular camera," in *Proc. Chin. Control Decis. Conf. (CCDC)*, Aug. 2020, pp. 93–98.

[21] J. Rehder, R. Siegwart, and P. Furgale, "A general approach to spatiotemporal calibration in multisensor systems," *IEEE Trans. Robot.*, vol. 32, no. 2, pp. 383–398, Apr. 2016.

[22] H. Shi, G. Lin, H. Wang, T. Hung, and Z. Wang, "SpSequenceNet: Semantic segmentation network on 4D point clouds," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 4573–4582.

[23] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 652–660.

[24] J. Deng, S. Shi, P. Li, W. Zhou, Y. Zhang, and H. Li, "Voxel R-CNN: Towards high performance voxel-based 3D object detection," in *Proc. AAAI Conf. Artif. Intell.*, vol. 35, no. 2, 2021, pp. 1201–1209.

[25] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "PointPillars: Fast encoders for object detection from point clouds," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 12697–12705.

[26] X. Weng, J. Wang, D. Held, and K. Kitani, "AB3DMOT: A baseline for 3D multi-object tracking and new evaluation metrics," 2020, *arXiv:2008.08063*.

[27] Y. Li, J. Deng, Y. Zhang, J. Ji, H. Li, and Y. Zhang, "EZFusion: A close look at the integration of LiDAR, millimeter-wave radar, and camera for accurate 3D object detection and tracking," *IEEE Robot. Autom. Lett.*, vol. 7, no. 4, pp. 11182–11189, Oct. 2022.

[28] G. Yuan, P. Sun, J. Zhao, D. Li, and C. Wang, "A review of moving object trajectory clustering algorithms," *Artif. Intell. Rev.*, vol. 47, no. 1, pp. 123–144, Jan. 2017.

[29] R. S. D. Sousa, A. Boukerche, and A. A. F. Loureiro, "Vehicle trajectory similarity: Models, methods, and applications," *ACM Comput. Surveys*, vol. 53, no. 5, pp. 1–32, Sep. 2021.

[30] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2015, pp. 1094–1099.

[31] A. Dosovitskiy, G. Ros, and F. Codevilla, "CARLA: An open urban driving simulator," in *Proc. Conf. Robot Learn.*, 2017, pp. 1–9.

[32] D. Krajzewicz, J. Erdmann, and M. Behrisch, "Recent development and applications of SUMO—Simulation of urban mobility," *Int. J. Adv. Syst. Meas.*, vol. 5, nos. 3–4, pp. 48–164, 2012.

[33] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2012, pp. 3354–3361.

[34] C. Choy, W. Dong, and V. Koltun, "Deep global registration," in *Proc. CVPR*, 2020, pp. 2514–2523.

[35] G. Elbaz, T. Avraham, and A. Fischer, "3D point cloud registration for localization using a deep neural network auto-encoder," in *Proc. CVPR*, 2017, pp. 1–9.

[36] J. Liu, H. Wu, C. Guo, H. Zhang, W. Zuo, and C. Yang, "Progress and consideration of high precision road navigation map," *Chin. J. Eng. Sci.*, vol. 20, no. 2, p. 99, 2018.

[37] P. W. Theiler, J. D. Wegner, and K. Schindler, "Keypoint-based 4-points congruent sets–automated marker-less registration of laser scans," *ISPRS J. Photogramm. Remote Sens.*, vol. 96, pp. 149–163, Oct. 2014.

[38] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (FPFH) for 3D registration," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2009, pp. 1–12.

[39] R. B. Rusu and S. Cousins, "3D is here: Point cloud library (PCL)," in *Proc. ICRA*, 2011, pp. 1–4.

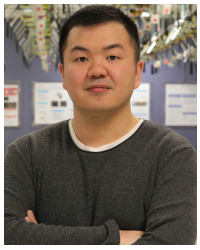
[40] *Leaderboard of Detection on Nuscenes*. Accessed: Jul. 23, 2023. [Online]. Available: https://www.nuscenes.org/object-detection?externalData=all&mapData=all&modalities=Any

[41] *Leaderboard of Detection on Kitti*. (2012). [Online]. Available: https://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d

[42] T. Yin, X. Zhou, and P. Krahenbuhl, "Center-based 3D object detection and tracking," in *Proc. CVPR*, 2021, pp. 1–10.


**Haojie Ren** received the bachelor's degree from the Hefei University of Technology. He is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, University of Science and Technology of China. His current research interests include the automatic calibration of sensors and collaborative perception.

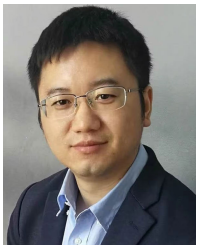
**Sha Zhang** received the bachelor's degree in computer science and technology from the University of Science and Technology of China, Hefei, Anhui, China, in 2020, where she is currently pursuing the Ph.D. degree. Her research interests include multimodality pretraining, 3D pretraining, and multi-object tracking (MOT).



**Sugang Li** received the M.S. and Ph.D. degrees from WINLAB, Rutgers University. He is currently a Software Engineer with Google Kubernetes Engine Networking. His research interests include the IoT protocol design over future internet architecture, edge/mobile computing, and machine learning.

**Yao Li** received the B.S. degree in electronic information engineering from Jilin University, Changchun, China, in 2019. He is currently pursuing the Ph.D. degree with the Computer Science and Technology, University of Science and Technology of China (USTC). His research interests include computer vision and intelligent transportation perception systems.

**Xinchen Li** received the B.S. degree in computer science and technology from the Taiyuan University of Technology, Taiyuan, Shanxi, China, in 2014. Currently, he is pursuing the Ph.D. degree with the University of Science and Technology of China, Hefei, Anhui, China. His research interests include multi-sensor calibration and sensor fusion techniques and their application in autonomous vehicles.



**Jianmin Ji** is currently an Associate Professor with the University of Science and Technology of China. His research interests include interested in AI and robotics, especially in deep reinforcement learning, robot navigation, knowledge representation and reasoning, and answer set programming.

**Yu Zhang** is currently a Professor with the University of Science and Technology of China. Her research interests include building automated, intelligent, reliable, and efficient programming systems for emerging fields, such as artificial intelligence (e.g., autonomous driving), cloud native, and quantum computing.

**Yanyong Zhang** (Fellow, IEEE) received the B.S. degree from the University of Science and Technology of China (USTC) in 1997 and the Ph.D. degree from Penn State University in 2002.

From 2002 and 2018, she was on the faculty of the Electrical and Computer Engineering Department at Rutgers University. Since July 2018, she has been joined the school of Computer Science and Technology at USTC. She has 21 years of research experience in the areas of sensor networks, ubiquitous computing, and high-performance computing, and has published more than 140 technical papers in these fields. She was also a member of the Wireless Information Networks Laboratory (Winlab). She received the NSF CAREER award in 2006. She has served/currently serves as the Associate Editor for several journals, including IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON SERVICE COMPUTING, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, and *Elsevier Smart Health*. She has served on various conference TPCs including DSN, Sensys, Infocom, etc. She is the TPC Co-Chair for IPSN'22.