Received 21 May 2024, accepted 23 June 2024, date of publication 2 July 2024, date of current version 12 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3422098

RESEARCH ARTICLE

Noise-Tolerant Trajectory Distance Computation in the Presence of Inherent Noise for Video Surveillance Applications

YONGJIN KWON[1,2], **JINYOUNG MOON**[1], **AND YEONSEUNG CHUNG**[2]
[1]Superintelligence Creative Research Laboratory, Electronics and Telecommunications Research Institute (ETRI), Daejeon 34129, Republic of Korea
[2]Department of Mathematical Sciences, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 34141, Republic of Korea

Corresponding author: Yongjin Kwon (scocso@etri.re.kr)

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government [Ministry of Science and ICT (MSIT)] (No. 2014-3-00123, Development of High Performance Visual BigData Discovery Platform for Large-Scale Realtime Data Analysis and No. 2020-0-00004, Development of Previsional Intelligence based on Long-term Visual Memory Network).

ABSTRACT As the importance of trajectory analysis arises in video surveillance, it becomes crucial to define the dissimilarity measure between two trajectories. Although the Hausdorff distance can be considered as a viable candidate for the measure, it is challenging to deal with noise present in trajectories since the Hausdorff distance is susceptible to noise so that even a single noise point may significantly distort the distance computation. In this paper, we propose a novel approach to alleviate the influence of inherent noise by setting noise-like points apart from ordinary points with a novel spatial tree structure during trajectory distance computation, without additional noise detection processes. In particular, we present R^{on}-tree, an extension of the existing spatial tree structure, that seamlessly finds *permanent* noise-like points, which are considered to have a low possibility of being ordinary points, and then keeps them in a separate auxiliary R-tree, without any separate process of disclosing noise-like points. We exploit R^{on}-tree to compute the noise-tolerant trajectory distance by modifying an existing algorithm for the Hausdorff distance. We also build an algorithm for noise-tolerant trajectory search to ensure accurate and high-quality search results even with noisy trajectories. The empirical results show that in all cases, our proposed approach yields the distance closest to the true one than any other competitor. The effectiveness of our approach is further examined by applying our noise-tolerant trajectory search to a real video surveillance dataset.

INDEX TERMS Trajectory distance, noise-tolerant Hausdorff distance, trajectory search, video surveillance.

I. INTRODUCTION

Trajectories, generated by tracking or sensing moving objects, are regarded as key features that provide the concise characteristics and semantics in their movements [1]. An enormous number of studies have investigated methods of analyzing and utilizing trajectories, such as trajectory clustering [2], [3], [4], [5], trajectory prediction [6], [7], [8], trajectory classification [9], [10], [11], trajectory pattern mining [12], [13], [14], and trajectory representation learning [15], [16], [17], [18]. Especially as the demand for vision-based surveillance technology continues to escalate [19], trajectory analysis can serve as a valuable tool for video surveillance. For example, by scrutinizing the trajectories of individuals appearing in videos, it is possible to recognize human behaviors, detect anomalous activities, and even forecast future events [20], [21], [22]. This information aids in the efficient automatic video analysis, a capability particularly beneficial in scenarios such as smart cities [23]. In these contexts, where numerous cameras generate substantial amounts of videos, relying solely on manual review of all footage is impractical [24].

In trajectory analysis, it is important to measure how similar or dissimilar two trajectories are from each

The associate editor coordinating the review of this manuscript and approving it for publication was Tao Zhou.

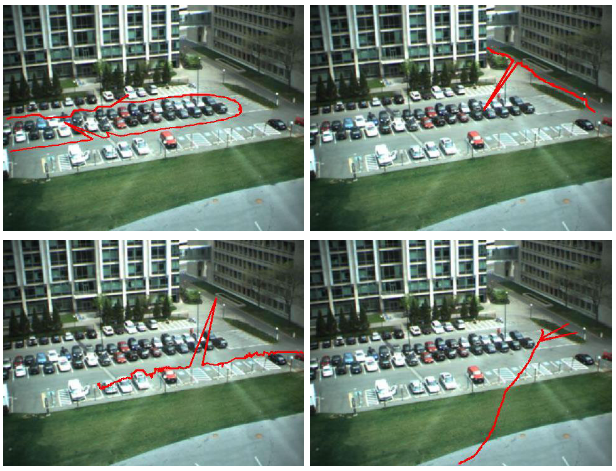


FIGURE 1. Examples of trajectories with a few noise points in the MIT trajectory dataset [29].

other [25], [26]. One candidate for this measure is the *Hausdorff distance* [27]. The Hausdorff distance between two nonempty point sets A and B in $\mathbb{R}^d$ is defined by

$$H(A, B) = \max\{h(A, B), h(B, A)\},$$

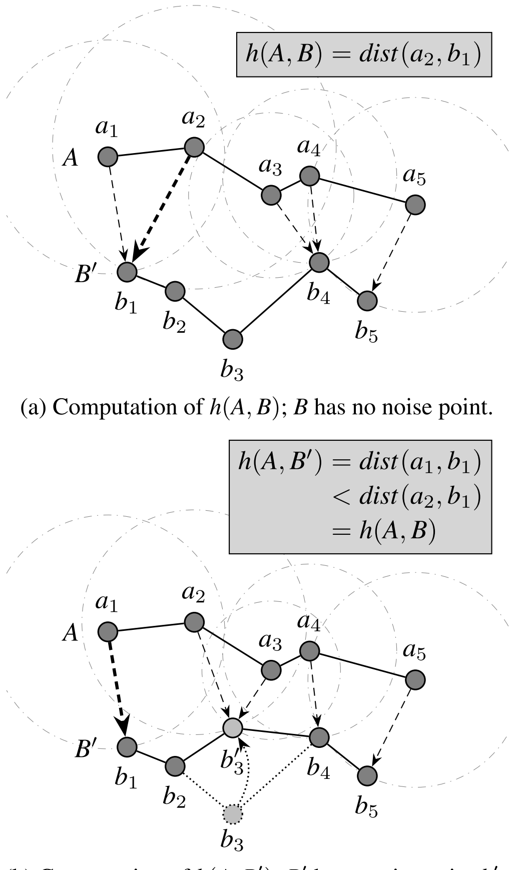
where $h(A, B)$, called the *directed Hausdorff distance*, is defined by

$$h(A, B) = \max_{a\in A} \left\{ \min_{b\in B} dist(a, b) \right\},$$

where $dist(\cdot, \cdot)$ is the Euclidean distance in $\mathbb{R}^d$. Treating a trajectory as a point set, the Hausdorff distance can be easily adopted to compute the distance between two trajectories.

A predominant disadvantage of the Hausdorff distance is its sensitivity to noise, which is usually unavoidable in real-world applications. Even a single noise point in either set can substantially affect the distance [28]. As shown in Fig. 1, which presents some trajectories from the MIT trajectory dataset [29], one of the well-known datasets in video surveillance, trajectories may contain one or several noise points, due to undesirable factors such as tracking errors. Although these noise points are a minuscule fraction of the trajectories, the trajectory distance based on the Hausdorff distance is significantly affected by such noise points. One might suggest that the emerging techniques for object detection and tracking could be employed to handle the inherent noise, but the noise is often inevitable in real-world applications even with such advanced algorithms in place.

To address the unwanted impact of noise, one may consider detecting noise points within trajectories [30] to reduce their influence by omitting them from the distance computation. This approach, however, usually requires ancillary implementations and computational resources for noise detection, which are not even a crucial part of the distance computation itself. Alternatively, one can use a modified distance designed for noise tolerance. There are several variants to mitigate the effect of noise. The partial Hausdorff distance [27] considers


$h(A, B) = dist(a_2, b_1)$
A
a_1
a_2
a_3
a_4
a_5
B'
b_1
b_2
b_3
b_4
b_5
$h(A, B') = dist(a_1, b_1) < dist(a_2, b_1) = h(A, B)$
b_3'

(a) Computation of $h(A, B)$; B has no noise point.

(b) Computation of $h(A, B')$; B' has a noise point b_3'.

FIGURE 2. An example of an underestimation of the Hausdorff distance caused by a noise point.

the kth largest value of $\min_{b\in B} dist(a, b)$ among $a \in A$, rather than taking the maximum over A, when computing $h(A, B)$. Similarly, the fractional Hausdorff distance [31] takes the $q\%$ quantile of $\min_{b\in B} dist(a, b)$ for $a \in A$. For example, it is recommended to use the 95% quantile of the distances in $h(A, B)$ to evaluate the performance of brain tumor segmentation methods [32]. More variants of the Hausdorff distance can be found in [33]. Although these variants require only slight modifications to the distance computation without additional noise detection processes, they may fail to diminish the influence of noise because they typically operate under the assumption that noise points lead to an overestimation of the distance. This is not always the case, as a noise point may, in fact, cause an underestimation of the distance, as shown in Fig. 2.

In this paper, we propose a novel approach to compute the noise-tolerant trajectory distance based on the Hausdorff distance in the presence of noise. Unlike existing methods, our approach eliminates the additional costs associated with noise detection while still preventing noise-like points inherent in noisy trajectories from being factored into the distance computation. In particular, instead of concerning the extra efforts for extracting noise-like points, we attempt to segregate noise-like points from ordinary points by adopting a novel spatial tree structure during trajectory distance computation. We initially pay attention to R^o-tree [34], a variant of R-tree used for trajectory distance computation in [35], that isolates noise-like objects in internal nodes at a higher level during the construction. Although R^o-tree can distinguish a few noise-like points from the others, R^o-tree has insufficient space to store noise points that usually occur proportionally to the length of the trajectory. Thus, we introduce R^{on}-tree, a variant of R^o-tree, that finds *permanent* noise-like points during the point insertion, which are considered to have a very low possibility of becoming ordinary points, and then stores them in a separate auxiliary R-tree. To compute the noise-tolerant distance, we modify an incremental algorithm for computing the Hausdorff distance [35], to visit ordinary points only by substituting R^{on}-tree for traditional R-tree, so that the distance would be close to the true distance (i.e., the distance between ordinary trajectories). In addition, we leverage the noise-tolerant trajectory distance to trajectory search, with the aim of finding the closest trajectories to a given query trajectory, to provide accurate and high-quality search results, even on noisy trajectories. The experimental results demonstrate that our method produces a distance that is closer to the true distance than any other approach. We also verify the effectiveness of our noise-tolerant trajectory distance by applying our noise-tolerant trajectory search to a real video surveillance dataset.

The main contributions of this paper can be summarized as follows.

- We propose a novel structure R^{on}-tree that seamlessly detaches permanent noise-like points in a trajectory and then keep them in a separate auxiliary tree. It resolves the problem of the previous R^o-tree and accomplishes the isolation of noise-like points.
- We exploit R^{on}-tree to develop an algorithm for computing the noise-tolerant trajectory distance. This algorithm is a simple adapation of an existing algorithm for the Hausdorff distance. We also extend the use of this algorithm to noise-tolerant trajectory search.
- We empirically demonstrate that assuming the presence of noise, our noise-tolerant distance is closer to the true one than any other method. We also show that our algorithm for noise-tolerant trajectory search returns more accurate and high-quality search results than other competitors.
- We utilize our noise-tolerant trajectory search to a real video surveillance dataset. We verify the effectiveness of our noise-tolerant trajectory distance by showing that the noise-tolerant trajectory search algorithm can not only identify relevant trajectories with some noise points, but it can also provide more semantically closer results to query trajectories with irregular mid-points.

The rest of this paper is arranged as follows. Section II presents related studies on computations of the Hausdorff distance and noise handling in a trajectory. Section III revisits the structure of R^o-tree and then extends it to our proposed R^{on}-tree. Section IV explains a modified algorithm for computing the noise-tolerant trajectory distance based on the Hausdorff distance using R^{on}-tree, and then describes how to employ the noise-tolerant distance for noise-tolerant trajectory search. Section V presents the experimental results of the noise-tolerant trajectory distance. Finally, Section VI concludes the paper and discusses future work.

II. RELATED WORK

A. HAUSDORFF DISTANCE AND ITS COMPUTATIONS

The Hausdorff distance is a dissimilarity measure that computes the extent to which two sets differ from each other. One of the notable advantages of the Hausdorff distance is that it is not restricted to a particular type of set. For example, the Hausdorff distance can quantify the similarity between two point sets, even though they have different sizes, without establishing a one-to-one mapping between them [36]. Thus it is frequently used in a number of applications, such as image processing and retrieval [37], [38], [39], [40], three-dimensional (3D) mesh matching [41], [42], and pattern recognition [43], [44], as well as trajectory analysis [25], [45].

The simplest way to compute the directed Hausdorff distance between two point sets A and B is to iterate a nested loop, where the inner loop scans all points b in B for a given $a \in A$ to compute $\min_{b \in B} dist(a, b)$, and the outer loop scans all points a in A to compute $h(A, B)$. Since this simple method requires high computational complexity, several studies have been suggested to build an efficient and practical algorithm for computing the Hausdorff distance. Nutanong et al. [35] exploited R-trees to avoid scanning all points in A and B. After building R-trees of A and B, their algorithms traverse the R-trees in an appropriate order, based on the lower and upper bounds of the Hausdorff distance, to quickly visit each pair of $a \in A$ and $b \in B$, whose distance appears to be close to $h(A, B)$. Taha and Hanbury [46] proposed an early breaking optimization that terminates the inner loop before scanning all points in B if it is determined that the inner loop cannot update the current candidate of $h(A, B)$. Chen et al. [47] enhanced the early breaking optimization, called the local start search algorithm, to terminate the inner loop in an earlier iteration, with the aid of a space-filling curve. They argued that the spatial locality of points is helpful for finding unnecessary points that do not contribute to $h(A, B)$. Zhang et al. [48] presented an octree-based approach, which is a slight improvement in the local start search algorithm, to accelerate the early breaking optimization for dense point sets. Ryu and Kamata [49], unlike other algorithms based on the early breaking optimization for stopping the inner loop, suggested a strategy of ruling out to reduce the number of iterations of the outer loop. To compute $h(A, B)$, their method sampled a small subset B_s of B and then computed a temporary candidate of $h(A, B)$ and a list of temporary minimum distances of A by computing $h(B_s, A)$, the directed Hausdorff distance in the opposite direction, which was used to exclude unnecessary points in A.

B. DETECTING NOISE POINTS IN A TRAJECTORY

Trajectories generated from various applications, including video surveillance, may contain noise because of the inevitable errors arising from sensing devices or tracking algorithms. A number of different approaches have been proposed to detect noise points in a trajectory. One approach is trajectory smoothing, which exploits mean/median filters [50]. For each point a_i in a trajectory A, the mean filter estimates the true point by $\widehat{a_i} = \sum_{j=1}^{n} a_{i-n+j}/n$, where n is the sliding window size. The median filter, which is more tolerant to noise, chooses the true point $\widehat{a_i}$ by the median of $a_{i-n+1}, a_{i-n+2}, \cdots, a_{i-1}, a_i$. Another way of identifying noise points is to apply a heuristic rule [51]. For example, if the speed at the point a_i, which is computed based on the time interval and distance from a_{i-1}, exceeds a predefined threshold, then we concern a_i as a noise point. Custers et al. [52] leveraged the physical properties of moving objects to extract noise points in a trajectory. In their study, a specific physical model is assumed, and then a maximum consistent subsequence of the trajectory is computed using that physical model. Then the points that are not included in the subsequence are considered as noise points. Chen et al. [26] suggested a way of substituting noise points through Lagrange interpolation. Their method compares neighboring points to determine noise points that exceed a predefined distance threshold, assuming that the movement of each object is limited. These approaches, however, require additional implementations and computational resources for noise-detecting algorithms and processes, which are not even an essential part of the distance computation.

III. R^{ON}-TREE

In this section, we first describe the structure and insertion/deletion process of R^o-tree [34], and we then investigate how R^o-tree fails to capture noise points. Traditional spatial tree structures, such as R-tree and its variants, have no ability to distinguish noise-like points from ordinary points. Although R^o-tree attempts to separate noise-like points, it often fails due to the inadequate space in this tree for noise-like points.

To address this limitation, we introduce R^{on}-tree, a novel extension of R^o-tree, where *permanent* noise-like points, which are not likely to be ordinary points, are identified during the insertion process and isolated into a separate auxiliary tree. This separation resolves the failure of R^o-tree in effectively handling noise-like points. These noise-like points, which may distort the distance calculation, will be excluded from the distance computation in our proposed method. This integrated approach eliminates the need for additional noise detection or smoothing algorithms, streamlining the process of noise-tolerant distance computation.

A. REVISITING R^O-TREE

R^o-tree [34], a variant of R-tree, is a balanced tree structure for spatial data, which isolates noise-like objects in internal nodes at a higher level. While R-tree relies on heuristics to maintain the minimum bounding rectangle (MBR) of each node as small as possible, R^o-tree identifies noise-like objects that cause significant changes in the MBR of each internal node during the insertion and deletion processes without additional efforts. This assessment of the extent of the MBR changes is performed using the *outlier identification algorithm* [53]. By setting aside noise-like objects, the nodes in R^o-tree maintain smaller MBRs than those in R-tree, so that R^o-tree achieves a better spatial query performance.

1) OUTLIER IDENTIFICATION ALGORITHM

We briefly describe the *outlier identification algorithm* [53], which enables R^o-tree to disclose noise-like objects during the insertion and deletion process. Given a set of objects S and a number p, the purpose of the outlier identification algorithm is to find an optimal subset P of S consisting of p objects such that the *gain* (see below) of shrinking the MBR of S to the MBR of $S - P$ is maximized. In other words, it aims to find p sacrifices in S to shrink the MBR of S as much as possible by deleting them. In practice, it returns a suboptimal subset T of S consisting of at most p objects such that the gain of shrinking the MBR of S to the MBR of $S - T$ is larger than a significant fraction of the maximal gain in a greedy fashion. The algorithm is leveraged to select sacrifices to be reinserted or pushed up in R^o-tree, when a node overflows.

The *gain* of shrinking a rectangle represents how much the *quality* of the rectangle is improved. Specifically, the gain of shrinking a rectangle R_1 to R_2 is defined by

$$G(R_1, R_2) = 1 - \frac{Q(R_1)}{Q(R_2)} = \frac{Q(R_1) - Q(R_2)}{Q(R_2)},$$

where the quality of a rectangle R with width w and height h is defined by

$$Q(R) = \frac{1}{w \times h} \left(\frac{\min\{w, h\}}{\max\{w, h\}} \right)^{\alpha}.$$

for parameter $\alpha \in [0, 1]$. Conversely, the *loss* of expanding a rectangle R_2 to R_1 is defined by the gain of shrinking R_1 to R_2, i.e.,

$$L(R_2, R_1) = G(R_1, R_2).$$

Since inserting an object into a target node incurs an expansion of the MBR of the node, the insertion operation at any leaf or internal node requires the calculation of the loss measure in R^o-tree.

2) STRUCTURE OF R^O-TREE

Similar to traditional R-tree, all leaf nodes in R^o-tree only contain spatial objects and appear at the same level. Internal nodes, unlike R-tree, may contain child nodes and some spatial objects, and appear at a higher level. The node at the highest level, called the *root* node, should have at least two child nodes, unless the node itself is a leaf node. In addition, each node has two parameters, namely m and M. Each leaf node should have at least m and at most M spatial

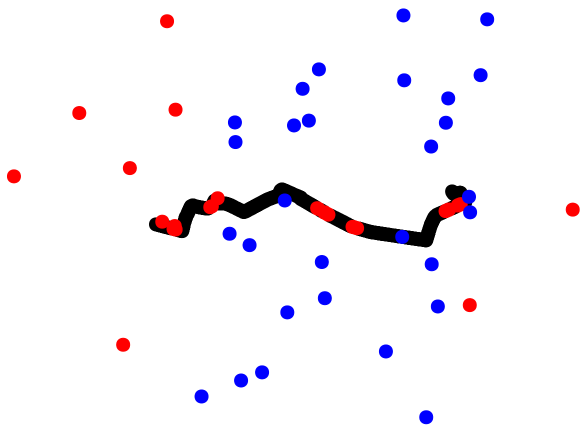


FIGURE 3. An exemplar of R^O-tree for a trajectory consisting of 800 points, with 5% noise points. Black points are true ordinary points, red points are considered as noise-like points by R^O-tree, and blue points are considered as ordinary points by R^O-tree but they are, in fact, noise points.

objects, and each internal node should have at least m child nodes and the sum of the number of its child nodes and its spatial objects should not exceed M. Typically, m is set to $0.4\,M$.

The insertion process of R^o-tree is described as follows. If a spatial object O is inserted into an internal node N, then we first try to find a child node C whose MBR contains O. If C exists, then we recursively insert O into C. Otherwise, O is regarded as a *temporary* noise-like object and is maintained in N. When N overflows, i.e., N temporarily has more than M entries of child nodes and noise-like objects, we first try to remove some entries that are chosen by the outlier identification algorithm, and then reinsert them back into R^o-tree itself. If the reinsertion is not available, then we next determine whether N has a sufficient number of noise-like objects, and then select one of them to *push down* into a child node. If the pushdown is also not available, then we should split N into two internal nodes to be inserted into the parent of N. We first split the child nodes in N into two groups in a similar way as in R-tree, and then assign each noise-like object into one of the groups in a way that the group expands by accommodating the object with minimum loss. On the other hand, if a spatial object O is inserted into a leaf node N, then O is simply kept in N. When N overflows, i.e., N temporarily has more than M spatial objects, we first attempt to perform the reinsertion as described above, and if the reinsertion is not available, then we split N into two leaf nodes to be inserted into the parent of N by dividing the spatial objects in a similar way in R-tree. Moreover, after the splitting of N, we reveal noise-like objects for each of them by the outlier identification algorithm to *push up* into the parent node.

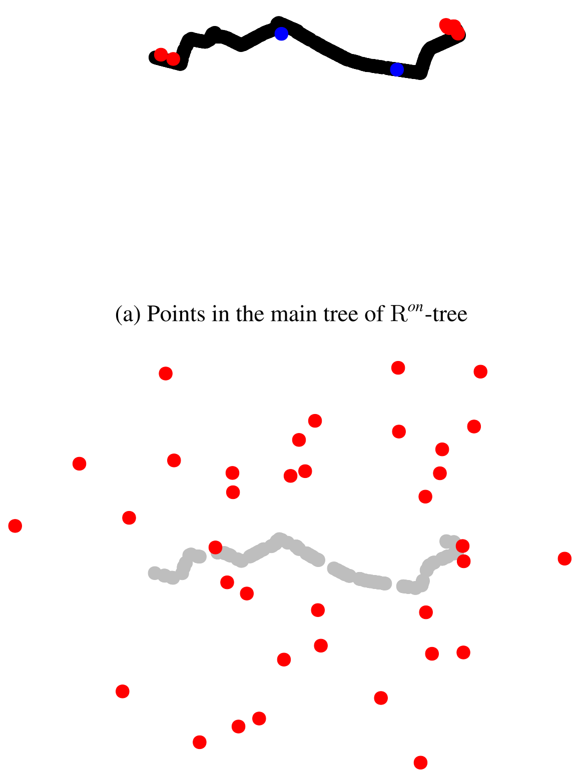
In the deletion process of R^o-tree, a notable difference compared with traditional R-tree is that the deletion may occur at an internal node, as well as a leaf node. If a spatial object O is isolated at an internal node N as a noise-like object and is to be deleted, then we simply eliminate it and adjust the MBR of N and all the ancestors of N. In this case, N cannot underflow since the number of child nodes remains unchanged. If a spatial object O is located at a leaf node N and is to be deleted, then we first eliminate it and handle the underflow of N and all the ancestors of N, if necessary. When a leaf node N underflows, i.e., N temporarily has less than m spatial objects, we first check whether the parent of N has a noise-like object O'. If O' exists, then we *push down* O' into N to make N have m spatial objects. Otherwise, N is removed from the parent of N, and the remaining spatial objects in N are reinserted back into R^o-tree. The deletion of N may cause an underflow of the parent of N, which is an internal node. When an internal node N underflows, i.e., N temporarily has less than m child nodes, we first *push down* noise-like objects in N into child nodes to anticipate splitting some child node to resolve the underflow problem. If N has no more noise-like objects, then N is removed from the parent node, and the remaining child nodes in N are reinserted back into R^o-tree. During the deletion process, if node N does not underflow, then we adjust the MBR of N and all the ancestors of N.

3) FAILURE OF R^O-TREE

While the construction of R^o-tree may distinguish a few noise objects from the others, R^o-tree may fail to perceive some noise points when there are more noise points in a given trajectory, which usually appear proportionally to the length of the trajectory. For example, consider an internal node N that has 12 leaf nodes as children, with parameters $m = 6$ and $M = 16$. Then N can maintain approximately 120 points at its children, which are in some part of the trajectory. If the trajectory has a small number of noise points so that N contains 1-3 noise points, then they can be isolated at N without any problems since N can keep four noise points. If the trajectory has 5% noise points, however, then there would be 5-7 noise points in N, but N has insufficient room to isolate them. Although the ancestors of N can also retain some noise-like points, the situation becomes even worse when there are other internal nodes suffering from the lack of capacity for noise-like points since the number of internal nodes at higher levels is much smaller than that of internal nodes at lower levels. In such a situation, the insertion of R^o-tree is likely to *push down* noise points into leaf nodes, which means that the noise points would be considered as ordinary points. Fig. 3 shows a failure of R^o-tree to extract true noise points from ordinary points.

B. STRUCTURE OF R^{ON}-TREE

R^{on}-tree is an extension of R^o-tree to alleviate the inadequate space for noise-like objects. The key intuition is that we attach a separate auxiliary R^*-tree [54] that keeps *permanent* noise-like objects, which are not likely to be ordinary objects, chosen from *temporary* noise-like objects. Let R be an R^{on}-tree. The main structure of R is almost identical to that of R^o-tree. With the root node $R.root()$, all leaf nodes contain



(a) Points in the main tree of R^{on}-tree

(b) Points in the auxiliary tree of R^{on}-tree

FIGURE 4. An exemplar of R^{on}-tree for the same trajectory as Fig. 3. (a) Black points are true ordinary points, red points are considered as temporary noise-like points by the main tree of R^{on}-tree, and blue points are considered as ordinary points by the main tree of R^{on}-tree but they are in fact noise points. (b) Both points are considered as permanent noise-like points, but gray points are in fact ordinary points.

ordinary objects and internal nodes may have temporary noise-like objects as well as child nodes. In addition, R possesses an auxiliary tree, accessed by $R.noise\_tree()$, that has the same structure as R*-tree. All leaf nodes of $R.noise\_tree()$ include permanent noise-like objects, and internal nodes contain only child nodes.

Permanent noise-like objects are selected at the pushup during the insertion process. Recall that in R^{o}-tree, after a leaf node N is split into two leaf nodes, we discriminate noise-like objects from the other ordinary objects in both nodes. The noise-like objects are then pushed up into the parent node PN. The problem arises when PN does not have adequate space to save them. In R^{o}-tree, it can be resolved by the overflow treatment; instead, we concern it as a chance to find permanent noise-like objects. We first compute the *optimal loss* associated with each noise-like object in PN, which is the smallest loss associated with expanding the MBR of some child node by adding the object. Note that it suffices to check the noise-like objects that are pushed up from N since we already maintain a list of noise-like objects in PN that are sorted by the optimal loss in ascending order [34]. We choose noise-like objects that exceed the space limit and that have the largest optimal losses compared to the others, and then we push them up into the parent of PN. This process may continue until we arrive at the root node. Since the root node does not have a parent, the noise-like objects that cannot be kept in the root node are treated as *permanent* noise-like objects, and they are finally moved to $R.noise\_tree()$. Through this process, temporary noise-like objects that are most significantly different from ordinary points are classified as permanent noise-like objects. Note that R^{on}-tree involves only a minimal additional construction cost compared to R^{o}-tree. Specifically, R^{on}-tree slightly modifies the overflow treatment process of R^{o}-tree, to recognize and segregate permanent noise-like points, which requires a small auxiliary tree construction cost.

Fig. 4 exemplifies how R^{on}-tree distinguishes noise-like points from ordinary points. The main tree of R^{on}-tree, as shown in Fig. 4 (a), excludes points that are farther from the trajectory, and the remaining noise points are isolated as temporary noise-like points (red points). The main tree may contain some false positives (blue points), but they would not be considered as being harmful in that they are almost adjacent to ordinary points. On the other hand, the auxiliary tree of R^{on}-tree in Fig. 4 (b) shows that apparent noise points are likely to be considered as permanent noise-like points (red points) during the construction of R^{on}-tree. The auxiliary tree, however, also contains a number of ordinary points (gray points), which is the unwilling behavior. This is due to the loss measure used in the insertion process [53]. The loss measure increases as the smaller rectangle expands. While noise points may expand a larger rectangle because of their nature, a new ordinary point may cause a larger loss since ordinary points are usually confined within a smaller rectangle. Note that such sacrifices are located within the trajectory in $\mathbb{R}^d$, so they affect the trajectory distance computation to a much lesser extent.

The deletion of a spatial object O from R is easily implemented by calling the deletions of O from both R and $R.noise\_tree()$. Note that R^{on}-tree requires more deletion costs than R^{o}-tree since we should determine which tree contains O by traversing both its main tree and auxiliary tree. In this paper, however, the deletion costs are not critical in that we are dealing with trajectories that are already generated, so that we are not likely to update individual points in them.

IV. NOISE-TOLERANT TRAJECTORY DISTANCE COMPUTATION

This section describes an algorithm, called NT-Inc-HausDist, for noise-tolerant trajectory distance computation based on the Hausdorff distance. We choose the incremental algorithm (Inc-HausDist) [35] as a starting point for achieving noise tolerance. Unlike other state-of-the-art algorithms, including early breaking optimization [46], diffusion search [48], and ruling out [49] algorithms, Inc-HausDist is an R-tree-based algorithm so that R^{on}-tree can be easily applied to the algorithm with small modifications. NT-Inc-HausDist exploits R^{on}-tree to effectively filter out noise-like points to ensure that the trajectory distance calculated remains close to the true distance even between noisy trajectories. In addition, we leverage R^{on}-tree to noise-tolerant trajectory search to guarantee superior search results even on noisy trajectories.

Note that we can similarly build two other algorithms (NT-BF-HausDist and NT-DF-HausDist) based on the two branch-and-bound algorithms in [35]. The details of these algorithms are given in Appendix A.

A. NOISE-TOLERANT TRAJECTORY DISTANCE COMPUTATION

As stated, the directed Hausdorff distance $h(A, B)$ for given trajectories A and B can be computed by iterating a nested loop, where the inner loop visits all points B and the outer loop all points in A. To avoid scanning all points in A and B, the incremental algorithm, Inc-HausDist [35], constructs R-trees for A and B (R_A and R_B, respectively) and then traverses both trees simultaneously using a nested structure of priority queues. Starting from the root nodes of R_A and R_B, we determine which tree is further traversed by comparing the heights of the exploring nodes N_A and N_B. While traversing either tree, we compute the lower/upper bounds of the distance from A to B, and then use them as keys for the inner and outer priority queues. We continue the traversals until we arrive at a point in R_A and a point in R_B simultaneously, at which we return the current upper bound as the desired distance from A to B. A significant advantage of Inc-HausDist is that it exploits a tighter upper bound by considering as an upper bound of $h(A, B)$ the minimum of the upper bounds of the distance from N_A to each node N_B in a subset of nodes in R_B, which is determined through the traversal of R_B. Theorem 1 shows the correctness of the tighter upper bound of $h(A, B)$.

Theorem 1: If $B_1, \cdots, B_n$ are disjoint subsets of B and $B = \bigcup_{i=1}^{n} B_i$, then

$$h(A, B) \leq \min_{1 \leq i \leq n} \overline{h}(A, B_i),$$

where $\overline{h}(A, B)$ denotes any upper bound of $h(A, B)$.

Proof: For any $i = 1, \cdots, n$, $B_i \subset B$ so that

$$\min_{b \in B} dist(a, b) \leq \min_{b \in B_i} dist(a, b).$$

Hence,

$$\begin{aligned} h(A, B) &= \max_{a \in A} \left\{ \min_{b \in B} dist(a, b) \right\} \\ &\leq \max_{a \in A} \left\{ \min_{b \in B_i} dist(a, b) \right\} \\ &= h(A, B_i) \\ &\leq \overline{h}(A, B_i). \end{aligned}$$

Since it holds for all $i = 1, \cdots, n$, we have

$$h(A, B) \leq \min_{1 \leq i \leq n} \overline{h}(A, B_i). \qquad \square$$

Algorithm 1 NT-TravLeft(N_A, SPQ, MPQ)

```
    Input: Node $N_A$ from $R_A$, SecPQ $SPQ$, MainPQ $MPQ$
    Output: Updated $MPQ$
 1: if $N_A$ is a leaf node then
 2:     for each point $P_A$ in $N_A$ do
 3:         $NewSPQ \leftarrow$ Create a SecPQ
 4:         $MinUB \leftarrow \infty$
 5:         for each entry $(N_B, \cdot)$ in $SPQ$ do
 6:             $LB \leftarrow$ MinDist($P_A$, $N_B.shrunkenMBR()$)
 7:             $NewSPQ.push(N_B, LB)$
 8:             $UB \leftarrow$ MaxDist($P_A$, $N_B.shrunkenMBR()$)
 9:             $MinUB \leftarrow \min(MinUB, UB)$
10:         end for
11:         $MPQ.push(P_A, MinUB, NewSPQ)$
12:     end for
13: else                                        ▷ $N_A$ is an internal node
14:     for each child node $C_A$ in $N_A$ do
15:         $NewSPQ \leftarrow$ Create a SecPQ
16:         $MinUB \leftarrow \infty$
17:         for each entry $(N_B, \cdot)$ in $SPQ$ do
18:             $LB \leftarrow$ MinDist($C_A.shrunkenMBR()$,
                        $N_B.shrunkenMBR()$)
19:             $NewSPQ.push(N_B, LB)$
20:             $UB \leftarrow$ MaxDist($C_A.shrunkenMBR()$,
                        $N_B.shrunkenMBR()$)
21:             $MinUB \leftarrow \min(MinUB, UB)$
22:         end for
23:         $MPQ.push(C_A, MinUB, NewSPQ)$
24:     end for
25: end if
26: return $MPQ$
```

1) NOISE-TOLERANT DIRECTED TRAJECTORY DISTANCE COMPUTATION

To accomplish noise tolerance, we adopt Inc-HausDist to compute the noise-tolerant directed trajectory distance, called NT-Inc-HausDist. It has three differences from Inc-HausDist. First, we do not visit both temporary and permanent noise-like points in A and B. We do not traverse the auxiliary tree, and even we do not touch temporary noise-like points that are isolated in the main tree. Thus, only ordinary points are visited so that we can achieve a noise-tolerant distance that would be close to the true one. Next, we use *shrunken* MBRs to compute the lower/upper bound of the noise-tolerant distance. The lower/upper bound of the distance between a node N_A in R_A and node N_B in R_B is computed by comparing the MBRs of N_A and N_B. We can reduce the MBR that covers temporal noise-like points, as well as ordinary points, to the shrunken MBR that covers only ordinary points, since temporary noise-like points are not visited during the computation of the noise-tolerant distance.

Algorithm 2 NT-TravRight(N_A, UB, SPQ, MPQ)

```
    Input: Node $N_A$ from $R_A$, Upper bound $UB$, SecPQ $SPQ$, MainPQ $MPQ$
    Output: Updated $MPQ$
 1: $(N_B, \cdot) \leftarrow SPQ.pop()$
 2: $MinUB \leftarrow UB$
 3: if $N_B$ is a leaf node then
 4:     for each point $P_B$ in $N_B$ do
 5:         $LB \leftarrow$ MinDist($N_A.shrunkenMBR()$, $P_B$)
 6:         $SPQ.push(P_B, LB)$
 7:         $UB \leftarrow$ MaxDist($N_A.shrunkenMBR()$, $P_B$)
 8:         $MinUB \leftarrow \min(MinUB, UB)$
 9:     end for
10: else                                        ▷ $N_B$ is an internal node
11:     for each child node $C_B$ in $N_B$ do
12:         $LB \leftarrow$ MinDist($N_A.shrunkenMBR()$,
                                 $C_B.shrunkenMBR()$)
13:         $SPQ.push(C_B, LB)$
14:         $UB \leftarrow$ MaxDist($N_A.shrunkenMBR()$,
                                 $C_B.shrunkenMBR()$)
15:         $MinUB \leftarrow \min(MinUB, UB)$
16:     end for
17: end if
18: $MPQ.push(N_A, MinUB, SPQ)$
19: return $MPQ$
```

Algorithm 3 NT-Inc-HausDist(A, B)

```
    Input: Trajectory $A$, Trajectory $B$
    Output: Noise-tolerant directed trajectory distance from $A$ to $B$
 1: return NT-Inc-HausDist*($A$, $B$, 0)

 2: function NT-Inc-HausDist*($A$, $B$, $LB$)
        Input: Trajectory $A$, Trajectory $B$, Lower bound $LB$
        Output: Maximum of $LB$ and noise-tolerant directed trajectory distance from $A$ to $B$
 3:     $R_A \leftarrow$ Create an R$^{on}$-tree for $A$
 4:     $R_B \leftarrow$ Create an R$^{on}$-tree for $B$
 5:     $MPQ \leftarrow$ Create a MainPQ
 6:     $InitSPQ \leftarrow$ Create a SecPQ
 7:     $InitSPQ.push(R_B.root(), 0)$
 8:     $MPQ.push(R_A.root(), \infty, InitSPQ)$
 9:     while $MPQ$ is not empty do
10:         $(N_A, UB, SPQ) \leftarrow MPQ.pop()$
11:         if $UB \leq LB$ then
12:             return $LB$
13:         end if
14:         $(N_B, \cdot) \leftarrow SPQ.top()$
15:         if $N_A$ and $N_B$ are both points then
16:             return $UB$
17:         else if $N_A$ is farther from the leaf than $N_B$ then
18:             NT-TravLeft($N_A$, $SPQ$, $MPQ$)
19:         else
20:             NT-TravRight($N_A$, $UB$, $SPQ$, $MPQ$)
21:         end if
22:     end while
23: end function
```

The shrunken MBRs produce a tighter upper bound, which improves the performance of NT-Inc-HausDist because the computation of the upper bound is a crucial part of the algorithms. Finally, to compute the lower and upper bounds of the noise-tolerant distance between N_A and N_B, we use MinDist and MaxDist [55], rather than HausDistLB and HausDistUB [35], because HausDistLB and HausDistUB work under the assumption that each border of the MBR contains at least one ordinary point, which would not be true in R^{on}-tree (and R^{o}-tree). There would be noise-like points in N_A, N_B or the nodes at lower levels that lie at some border of the MBRs, and even of the shrunken MBRs, while no ordinary point lies at the border. Hence, we choose looser but accurate functions, MinDist and MaxDist, to compute the lower and upper bounds. Note that using the less tighter lower/upper bounds is likely to cause a decline in computational performance.

In the incremental method, we exploit a nested structure of priority queues, as described in [35], which has a main priority queue (*MainPQ*), and each entry of MainPQ contains a secondary priority queue (*SecPQ*). MainPQ maintains an entry (node N_A, number UB, SecPQ SPQ), where N_A is a node from R_A, UB is an upper bound of the noise-tolerant distance from N_A to B, and SPQ is the associated SecPQ that determines the next node in R_B to be explored. For each entry (N_A, UB, SPQ) of MainPQ, the associated SecPQ takes entries (node N_B, number LB), where N_B is a node from R_B and LB is an lower bound of the noise-tolerant distance from N_A to N_B. Note that MainPQ arranges its entries in descending order, while SecPQ arranges its entries in ascending order. This nested structure is well associated with the directed Hausdorff distance since it is a type of max-min distance.

NT-Inc-HausDist exploits two functions, NT-TravLeft (Algorithm 1) and NT-TravRight (Algorithm 2), to traverse R_A and R_B, respectively. NT-TravLeft(N_A, SPQ, MPQ) for a node N_A in R_A explores each point P_A (if N_A is a leaf node) or child node C_A (if N_A is an internal node) in N_A and updates MPQ as follows. For each entry (N_B, $\cdot$) in SPQ, a new SecPQ $NewSPQ$ takes an entry (N_B, LB), where LB is a lower bound of the distance from P_A or C_A to N_B, and $MinUB$ is set to the minimum of the upper bounds of the distance from P_A or C_A to N_B. MPQ are then updated by adding an entry (P_A or C_A, $MinUB$, $NewSPQ$). On the other hand, NT-TravRight(N_A, UB, SPQ, MPQ) for a node N_A in R_A explores the points P_B (if N_B is a leaf node) or child nodes C_B (if N_B is an internal node) in N_B, where N_B is the first component of the entry dequeued from SPQ, and updates MPQ as follows. We first initialize $MinUB$ with UB. For each point P_B or child node C_B in N_B, an entry (N_B, LB) is inserted into SPQ, where LB is a lower bound of the distance from N_A to P_B or C_B, and $MinUB$ is given by the minimum of the upper bounds of the distance from N_A to P_B or C_B. Finally, we update *MPQ* by adding an entry (N_A, *MinUB*, *SPQ*).

As described in Algorithm 3, NT-Inc-HausDist returns the result of the *starred* algorithm (NT-Inc-HausDist*) with third argument 0 (Line 1). NT-Inc-HausDist* takes two trajectories A, B and a nonnegative number *LB* as arguments. Note that the third argument *LB* plays the role of terminating the algorithm if we explore a node with an upper bound *UB* that is smaller than *LB*. NT-Inc-HausDist* first initializes *InitSPQ* with entry ($R_B.root()$, 0) and *MPQ* with entry ($R_A.root()$, ∞, *InitSPQ*) (Lines 5 to 8). In the while loop (Lines 9 to 22), the entry (N_A, *UB*, *SPQ*) is dequeued from *MPQ*. If $UB \leq LB$, then the algorithm returns *LB* in that the distance is even smaller than *UB*. We also look up the head entry ($N_B, \cdot$) of *SPQ*. If N_A and N_B are both points, then *UB* is the desired distance since N_B is the closest point for N_A by the ascending ordering of *SPQ*, and N_A yields the highest distance from any other points in A by the descending ordering of *MPQ*. Otherwise, we further traverse either R_A or R_B, by comparing the heights of N_A and N_B.

2) NOISE-TOLERANT UNDIRECTED TRAJECTORY DISTANCE COMPUTATION

To compute the noise-tolerant *undirected* trajectory distance, we use a similar approach to that provided in [35]. Let

$$\begin{aligned} L_1 &= \text{MinDist}(R_A.root().shrunkenMBR(), \\ &\qquad\qquad R_B.root().shrunkenMBR()), \\ L_2 &= \text{MinDist}(R_B.root().shrunkenMBR(), \\ &\qquad\qquad R_A.root().shrunkenMBR()). \end{aligned}$$

Note that L_1 and L_2 are lower bounds of the noise-tolerant distance from A to B and from B to A, respectively. By the definition of $H(A, B)$, both $h(A, B)$ and $h(B, A)$ themselves can be treated as lower bounds of $H(A, B)$. Based on this observation, if $L_1 > L_2$, then we first compute d_1 as the noise-tolerant distance from A to B using NT-Inc-HausDist. Taking $LB = d_1$ as an argument, we compute d_2 as the noise-tolerant distance from B to A using the *starred* algorithm (NT-Inc-HausDist*), to guarantee that $d_2 \geq d_1$. Then d_2 will be the desired distance between A and B. If $L_1 \leq L_2$, then we simply change the order of A and B and then proceed to the calculation.

B. NOISE-TOLERANT TRAJECTORY SEARCH

For a given query trajectory Q, trajectory search finds the closest trajectory to Q from a collection of trajectories $\mathcal{D}$, where the closeness is measured by the Hausdorff distance. TrajSearch [35] is an incremental algorithm for trajectory search when $\mathcal{D}$ consists of ordinary trajectories. We introduce an algorithm for noise-tolerant trajectory search (called NT-TrajSearch), assuming that trajectories in $\mathcal{D}$ are noisy, which is also a modification of TrajSearch. For simplicity, we only consider the noise-tolerant *undirected* trajectory distance as the closeness measure, but the algorithm can be easily extended to directed ones.

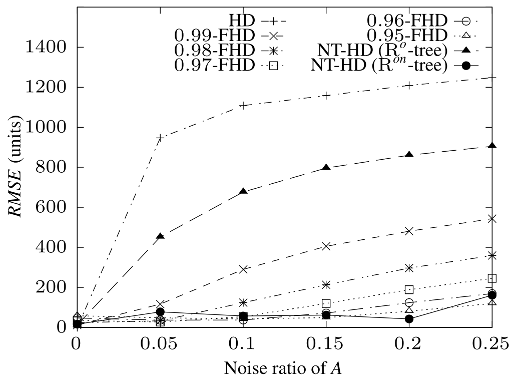
Algorithm 4 NT-TrajSearch(P, R_B)

```
    Input: Query trajectory $Q$, Trajectory set $\mathcal{D}$
    Output: Trajectory with the smallest noise-tolerant undirected trajectory distance from $Q$
1:  $M_Q$ ← Create an MBR of $Q$
2:  $\mathcal{R}^{on}_{\mathcal{D}}$ ← Create a collection of R$^{on}$-trees for $\mathcal{D}$
3:  $R_{\mathcal{D}}$ ← Create an R-tree for the main trees of $\mathcal{R}^{on}_{\mathcal{D}}$
4:  $PQ$ ← Create a priority queue in "ascending order"
5:  $PQ.push(R_{\mathcal{D}}.root(), 0, False)$
6:  while $PQ$ is not empty do
7:      $(N_{\mathcal{D}}, d_N, IsFinal) \leftarrow PQ.pop()$
8:      if $N_{\mathcal{D}}$ is a trajectory then
9:          if $IsFinal$ then
10:             return $d_N$
11:         else
12:             $d$ ← noise-tolerant trajectory undirected distance between $N_{\mathcal{D}}$ and $Q$
13:             $PQ.push(N_{\mathcal{D}}, d, True)$
14:         end if
15:     else                 ▷ $N_{\mathcal{D}}$ is either a leaf or internal node
16:         for each trajectory or child node $C_{\mathcal{D}}$ in $N_{\mathcal{D}}$ do
17:             $d \leftarrow \max(\text{MinDist}(M_Q, C_{\mathcal{D}}.MBR()), \text{MinDist}(C_{\mathcal{D}}.MBR(), M_Q))$
18:             $PQ.push(C_{\mathcal{D}}, d, False)$
19:         end for
20:     end if
21: end while
```

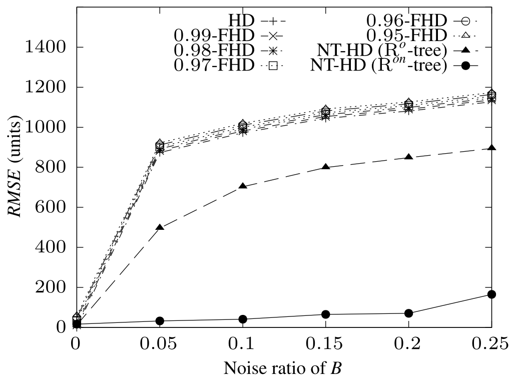
NT-TrajSearch is described by Algorithm 4. NT-TrajSearch takes a query trajectory Q and a trajectory set $\mathcal{D}$ as arguments. We first build R^{on}-trees for all trajectories in $\mathcal{D}$, and then create an R-tree $\mathcal{R}_{\mathcal{D}}$ for the main trees of those R^{on}-trees. Initially, a priority queue *PQ*, which contains entries (node $N_{\mathcal{D}}$, number d_N, boolean *IsFinal*) in ascending order, takes an entry ($R_{\mathcal{D}}.root()$, 0, *False*). For each iteration of the while loop (Lines 6 to 20), the entry ($N_{\mathcal{D}}$, d_N, *IsFinal*) is dequeued from *PQ*. If $N_{\mathcal{D}}$ is actually a trajectory, then we either return d_N if *isFinal* is true, or we compute the noise-tolerant undirected trajectory distance d between $N_{\mathcal{D}}$ and Q using one of the noise-tolerant algorithms (NT-DF-HausDist, NT-BF-HausDist, or NT-Inc-HausDist), and then enqueue the entry ($N_{\mathcal{D}}$, d, *True*) into *PQ* if *isFinal* is false. Otherwise, we estimate the lower bound of the noise-tolerant undirected trajectory distance between Q and each trajectory or child node $C_{\mathcal{D}}$ in $N_{\mathcal{D}}$ using their MBRs M_Q and $C_{\mathcal{D}}.MBR()$ to find a trajectory with a smaller lower bound in a best-first manner. Owing to the nature of R^{on}-trees, it is safe to use as the lower bound the maximum of MinDist(M_Q, $C_{\mathcal{D}}.MBR()$) and MinDist($C_{\mathcal{D}}.MBR()$, M_Q).

V. EXPERIMENTS

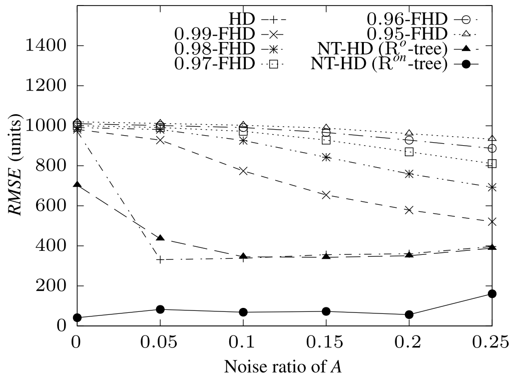
This section presents the experimental results of the proposed algorithms. We first examine how the estimated trajectory distance based on R^{on}-tree is noise-tolerant in the presence of noise. We also present the results of the noise-tolerant



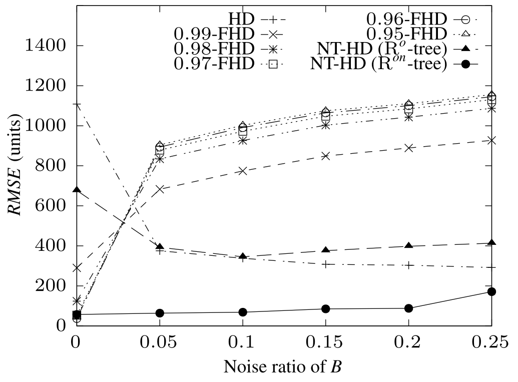
(a) The noise ratio of A varies, while B is ordinary.



(b) The noise ratio of B varies, while A is ordinary.



(c) The noise ratio of A varies, while B is noisy.



(d) The noise ratio of B varies, while A is noisy.

FIGURE 5. Comparison of the *RMSE*s of the noise-tolerant *directed* trajectory distances, as well as the exact *directed* distance. (a) The noise ratio of *A* varies and *B* is ordinary. (b) The noise ratio of *B* varies and *A* is ordinary. (c) The noise ratio of *A* varies and the noise ratio of *B* is 10%. (d) The noise ratio of *B* varies and the noise ratio of *A* is 10%.

trajectory search compared to the true results. Then we show, as an example application, the qualitative results of the noise-tolerant trajectory search in video surveillance.

In our experimental studies, we used the following datasets.

- We used the Oldenburg road network [56] to quantitively examine the proposed algorithms. Specifically, each ordinary trajectory generated is the shortest path in the network, whose end points are randomly picked and whose length is 2,000±1% units. Each trajectory was represented as a point set, where points on the trajectory were sampled at five different resolutions: 400, 800, 1,200, 1,600, and 2,000. To create noisy trajectories, we follow the transformation method in [57] so that we select 5%, 10%, 15%, 20%, and 25% of the points on each ordinary trajectory, but we add Gaussian noise, rather than a fixed margin, to simulate the natural uncertainty [58], with zero mean and standard deviation of 500 units, respectively. The trajectory dataset was arranged according to the resolution and noise ratio (including ordinary trajectories).
- We exploited the MIT trajectory dataset [29] to verify the effectiveness of the noise-tolerant trajectory distance in video surveillance by applying our noise-tolerant trajectory search algorithm (NT-TrajSearch). This dataset is constituted of 40,453 object trajectories recorded over five days from a single stationary camera in a parking lot.

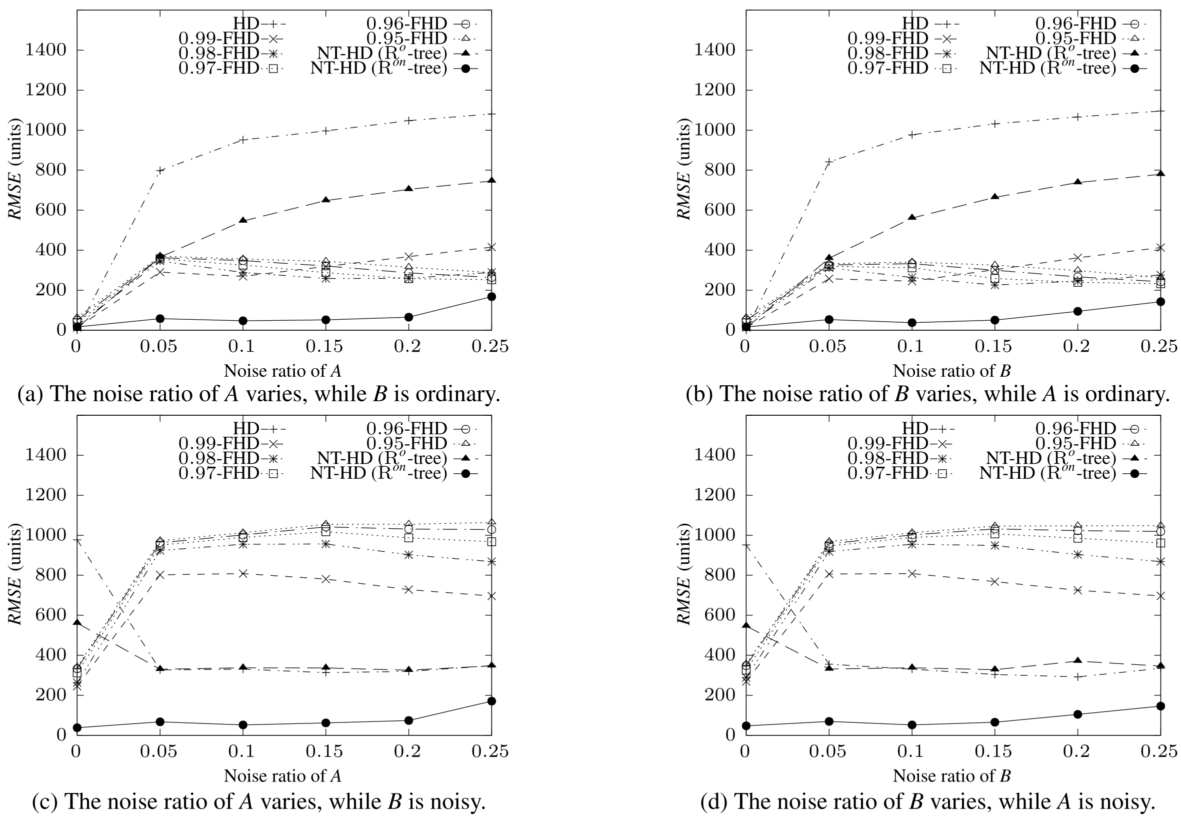
A. NOISE-TOLERANT TRAJECTORY DISTANCE COMPUTATION

In this experiment, we explored the closeness of the noise-tolerant distances between trajectories A and B to the true distances, based on the following four cases.

(a) The noise ratio of A is varied, while B is ordinary (i.e., non-noisy).
(b) The noise ratio of B is varied, while A is ordinary.
(c) The noise ratio of A is varied, while the noise ratio of B is 10%.
(d) The noise ratio of B is varied, while the noise ratio of A is 10%.

These four cases cover all possible scenarios that could occur during the noise-tolerant trajectory directed and undirected distance computation. Conditions (a) and (b) assess the results of varying noise ratios in one trajectory while keeping the other trajectory non-noisy. Conditions (c) and (d) evaluate the effect of varying noise ratios in one trajectory while the other trajectory has a fixed noise ratio.

For comparison, we examined the Hausdorff distance (HD), 100α%-quantile fractional Hausdorff distance (α-FHD)



(a) The noise ratio of A varies, while B is ordinary. (b) The noise ratio of B varies, while A is ordinary. (c) The noise ratio of A varies, while B is noisy. (d) The noise ratio of B varies, while A is noisy.

FIGURE 6. Comparison of the *RMSE*s of the noise-tolerant *undirected* distances, as well as the exact *undirected* distance. (a) *B* is ordinary and the noise ratio of *A* varies. (b) *A* is ordinary and the noise ratio of *B* varies. (c) The noise ratio of *B* is 10% and the noise ratio of *A* varies. (d) The noise ratio of *A* is 10% and the noise ratio of *B* varies.

with $\alpha = 0.99, 0.98, 0.97, 0.96$, and 0.95, and the proposed noise-tolerant trajectory distance (NT-HD (R^{on}-tree)). α-FHD is a well-known variant of Hausdorff distance that is frequently employed in the presence of noise [32], [59]. In particular, α-FHD computes trajectory distance without requiring separate noise detection or smoothing, making it suitable for comparison with our approach. In addition, we also examined the noise-tolerant distance based on R^{o}-tree (NT-HD (R^{o}-tree)), which can be easily implemented by adapting our proposed algorithms. Note that all distance measures are compared to the *true* distance, which is considered as the distance between two trajectories obtained in the ideal case where noise detection completely filters out all noise points.

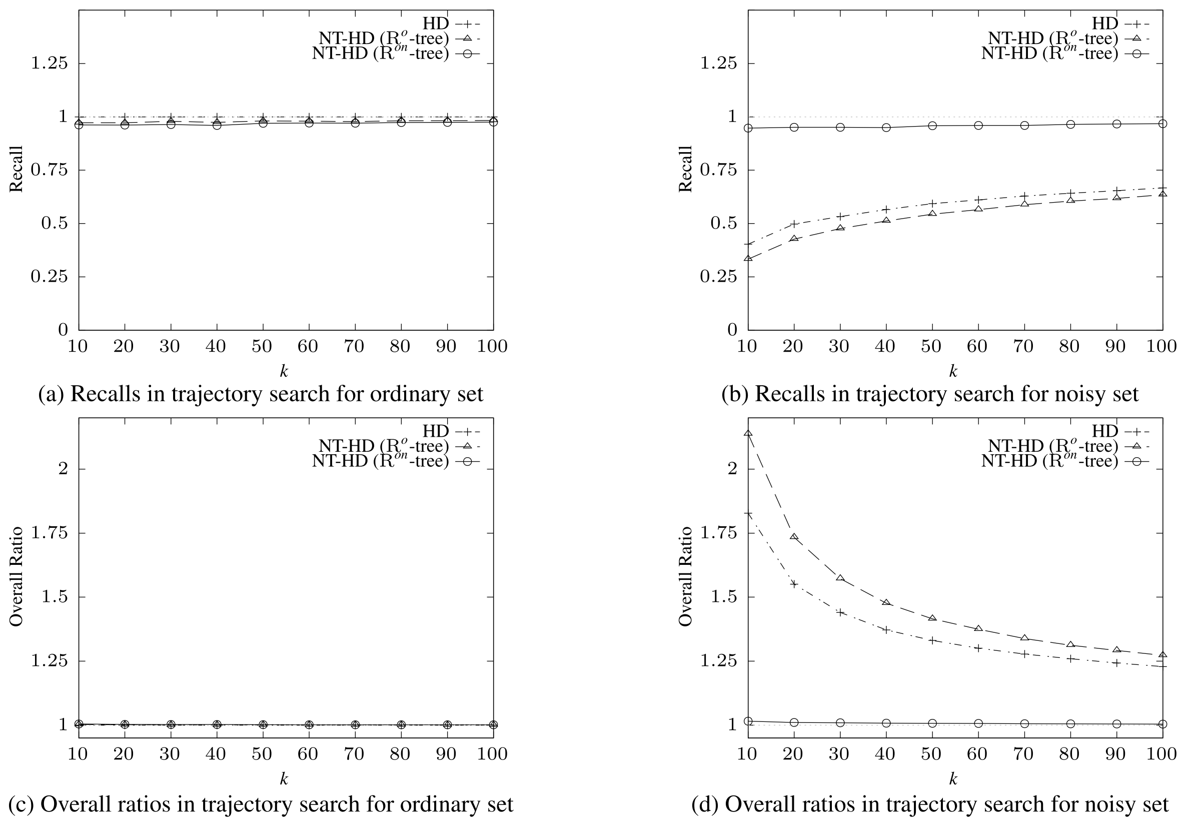
To measure how close the noise-tolerant distance is to the true distance, we used the *root mean squared error* (RMSE), which is defined by

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\widehat{d_i} - d_i)^2}{n}},$$

where n is the number of pairs of trajectories, $\widehat{d_i}$ is the noise-tolerant distance of the ith pair, and d_i is the *true* distance of the ith pair of ordinary trajectories. Note that in this experiment, we set the resolutions of both A and B to 2,000, and $n = 200$.

Fig. 5 shows the *RMSE*s of the *directed* distances. First, we analyze the case where both A and B are ordinary, to ensure the practical feasibility of noise-tolerant trajectory distance computation in the absence of noise. As shown in Fig. 5 (a), where the noise ratio of A is 0 (and in Fig. 5 (b), where the noise ratio of B is 0), it can be observed that the *RMSE*s of all methods approach 0. Specifically, the *RMSE* of NT-HD (R^{on}-tree) (≤ 20) is much closer to 0 than those of α-FHDs ($\alpha < 0.99$) and is comparable to those of 0.99-FHD and NT-HD (R^{o}-tree). This indicates that our proposed method is also applicable to situations when no noise is present in the trajectories.

Next, we examine the cases where either A, B, or both are noisy. As expected, HD has large *RMSE*s for all cases where noise is present. In addition, α-FHDs, although they have been considered as a noise-tolerant alternative, also have large *RMSE*s for all cases except the case (a) since under (a), noise points in A cause the overestimation of $h(A, B)$, which is the underlying assumption of α-FHDs. Moreover, contrary to the expectation that α-FHD would become more noise-tolerant as α decreases, in the cases (c) and (d) where both A and B are noisy, α-FHDs are even larger *RMSE*s than HD, and as α decreases from 0.99 to 0.95, the *RMSE* of α-FHD becomes larger. This is because the noise points in A and in B would appear to cancel each other out to some extent, but α-FHDs would only remove the effects of the noise points



(a) Recalls in trajectory search for ordinary set

(b) Recalls in trajectory search for noisy set

(c) Overall ratios in trajectory search for ordinary set

(d) Overall ratios in trajectory search for noisy set

FIGURE 7. Accuracy measures of the results of noise-tolerant trajectory search, as well as the exact trajectory search (HD). In the left column, the *k* closest trajectories are retrieved from a collection of 2,000 ordinary trajectories. In the right column, we search the *k* closest trajectories from a set of 2,000 noisy trajectories with a noise ratio of 10%.

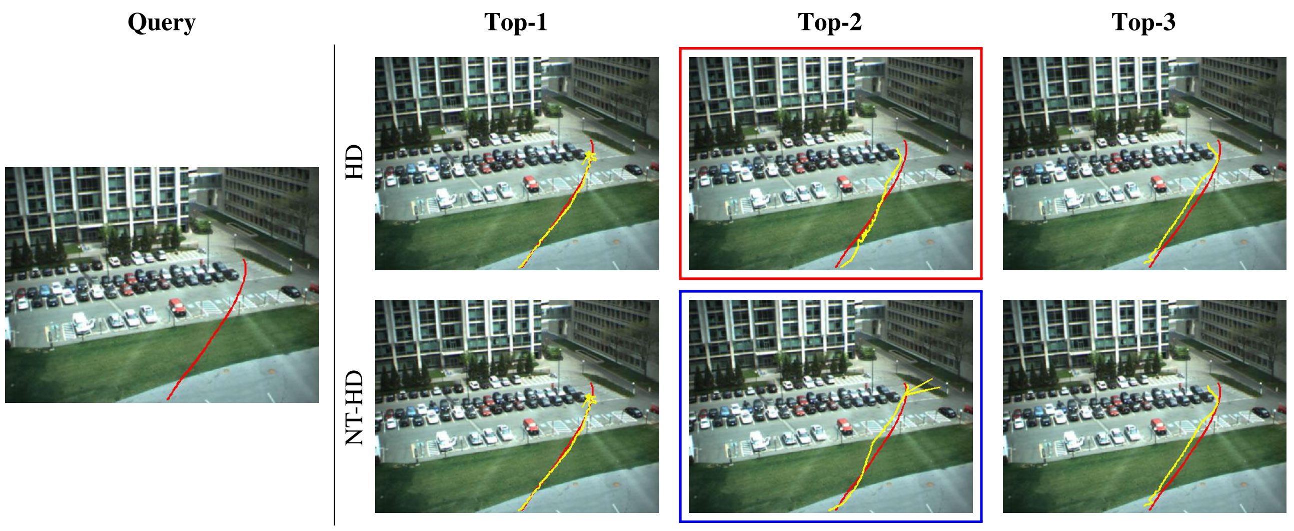
in A, which leads to larger $RMSE$s. NT-HD (R^o-tree) has somewhat smaller $RMSE$s than HD for the cases (a) and (b), which implies that R^o-tree can capture a certain number of noise points. However, R^o-tree has the limitation of its ability to extract all noise points, so NT-HD (R^o-tree) still has large $RMSE$s for all noise ratios ≥ 0.05, and NT-HD (R^o-tree) has similar $RMSE$s to HD for the cases (c) and (d). In contrast, R^{on}-tree can extract most of the noise points with the aid of the separate auxiliary tree, and thus the proposed NT-HD (R^{on}-tree) outperforms the other distances in all cases.

Fig. 6 compares the $RMSE$s of the *undirected* distances. Since the distances themselves are symmetric, the $RMSE$s of the cases (a) and (b), and of the cases (c) and (d) show similar patterns. We can observe that the behaviors of undirected HD, NT-HD (R^o-tree), and NT-HD (R^{on}-tree) are similar to those of the directed ones for all cases. In contrast, α-FHDs, a widely adopted noise-tolerant alternative, show moderate $RMSE$s between Fig. 5 (a) and (b) for cases (a) and (b), respectively, and Fig. 5 (c) and (d) for cases (c) and (d), respectively. In particular, this phenomenon is more pronounced in cases (a) and (b) with ordinary trajectories. Considering that the undirected distance is defined as the maximum of directed distances ($h(A, B)$ and $h(B, A)$), α-FHD may reduce the effect of noise points in situations like Fig. 5 (a), where noise points cause the overestimation of directed distances. Conversely, in situations like Fig. 5 (b), where noise points cause the underestimation of directed distances, α-FHD may fail to adequately mitigate the noise effect, leading to either of these scenarios being reflected in the results of noise-tolerant undirected distance. Consequently, the $RMSE$ of α-FHDs shown in Fig. 6 (a) and (b) can be higher than that in Fig. 5 (a) but lower than that in Fig. 5 (b). Overall, NT-HD (R^{on}-tree) outperforms the other distances in all cases. This is because R^{on}-tree can adequately exclude most of the noise points from the process of distance calculation, as we anticipated. Thus, the noise-tolerant results reaffirm the effectiveness of our proposed method in the presence of noise.

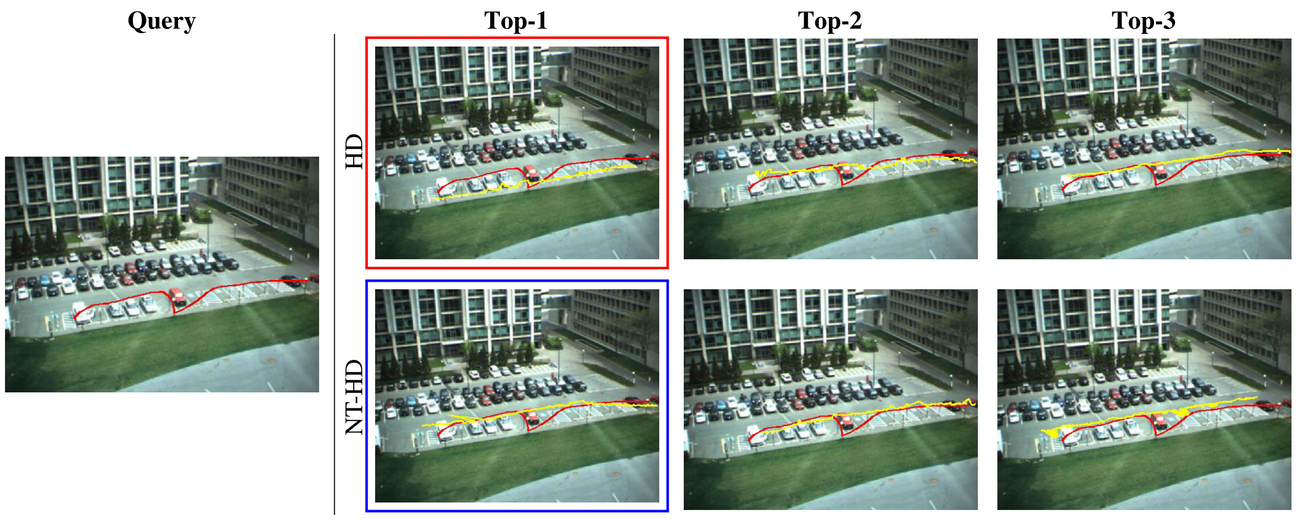
Additional experimental results on the performance measures of the proposed noise-tolerant algorithms are provided in Appendix B.

B. NOISE-TOLERANT TRAJECTORY SEARCH

This experiment demonstrates the accuracy of the results of the trajectory search methods compared with the true results. We compared the proposed noise-tolerant trajectory search NT-TrajSearch (NT-HD (R^{on}-tree)) to the exact (i.e., noise-sensitive) trajectory search TrajSearch (HD). We also implemented the noise-tolerant trajectory search based on R^o-tree (NT-HD (R^o-tree)) by revising NT-TrajSearch. Note that since these algorithms explore trajectories in an incremental manner, it can be easily modified to find the k closest



(a) Comparison of the results of trajectory search algorithms given a simple query trajectory



(b) Comparison of the results of trajectory search algorithms given a query trajectory that detours under the red car

FIGURE 8. Examples of the results of the exact trajectory search (HD) and the noise-tolerant trajectory search (NT-HD) algorithms. For each example, the red line indicates the query trajectory, and the yellow lines indicate the top-1, top-2, and top-3 results of the trajectory search algorithms.

trajectories to the query trajectory Q. We used the following measures to compute the accuracy.

(a) *Recall*: the fraction of the number of the k returned trajectories that are actually members of the true k closest trajectories, i.e.,

$$Recall = \frac{|\{T_1, \cdots, T_k\} \cap \{T_1^*, \cdots, T_k^*\}|}{k},$$

where T_i is the ith returned trajectory and T_i^* is the true ith closest trajectory. Note that $0 \leq Recall \leq 1$, and $Recall = 1$ when the result is correct.

(b) *Overall Ratio* [60]: the average of the rank-i approximation ratios for $i = 1, \cdots, k$, i.e., for a query trajectory Q,

$$Overall\ Ratio = \frac{1}{k}\sum_{i=1}^{k}\frac{H(T_i, Q)}{H(T_i^*, Q)},$$

where T_i is the ith returned trajectory and T_i^* is the true ith closest trajectory. Note that $Overall\ Ratio \geq 1$, and $Overall\ Ratio = 1$ when the result is correct.

We prepared two trajectory sets; one consisted of 2,000 ordinary trajectories (ordinary set) and the other consisted of 2,000 noisy trajectories with a noise ratio of 10% (noisy set). In both trajectory sets, the resolution of each trajectory was set to 2,000. Each record was obtained by averaging the results of 200 different query trajectories, which were ordinary and had a resolution of 2,000, with varying k.

Fig. 7 shows the accuracy measures of trajectory search results. For the ordinary set, all methods return high-quality search results in terms of *Recall* and *Overall Ratio*. For the noisy set, however, we observe that HD and NT-HD (R^o-tree) show poor search results. In particular, the *Recall*s of HD and NT-HD (R^o-tree) were even smaller than 0.5 when $k \leq 20$, i.e., over the half of search results were different from the true ones. The *Overall Ratio*s of HD and NT-HD (R^{o}-tree) were also far from 1, which implies that the quality of the search results is quite poor. On the other hand, NT-HD (R^{on}-tree) still shows accurate and high-quality search results, i.e., both *Recall* and *Overall Ratio* are very close to 1 for any k. In summary, assuming the existence of noise, NT-TrajSearch (NT-HD (R^{on}-tree)) outperforms all other algorithms for trajectory search in accuracy measures.

C. NOISE-TOLERANT TRAJECTORY SEARCH IN VIDEO SURVEILLANCE

In this experiment, we apply our noise-tolerant trajectory search, as a representative example, to the MIT trajectory dataset [29], one of the well-known real surveillance datasets that includes a diverse set of trajectories with varying levels of noise, as shown in Fig. 1. Through this experiment, we validate the effectiveness of our noise-tolerant trajectory distance under real-world conditions. Particularly, we demonstrate that our method can not only identify relevant trajectories despite the inclusion of some noise points, but also provide more semantically closer results to query trajectories.

Fig. 8 shows the results of the exact trajectory search (HD) and our noise-tolerant trajectory search (NT-HD) algorithms given two exemplar query trajectories. For each example, the red line indicates the query trajectory, and the yellow lines indicate the top-1, top-2, and top-3 results of the trajectory search algorithms.

Fig. 8 (a) shows that the top-2 result (blue box) of our noise-tolerant trajectory search algorithm includes some noise points. If these visually identifiable noise points are excluded, we would notice that the resulting trajectory is similar to the query trajectory. This ability to perceive relevant trajectories that may inherently include noise points is the distinct advantage of our noise-tolerant trajectory search algorithm over the exact trajectory search algorithm. Notably, in certain applications such as hidden follower detection [61], where the requirement is to identify relevant trajectories even in the presence of noise, our noise-tolerant trajectory search algorithm could potentially be a valuable tool for discerning truly similar trajectories.

Fig. 8 (b) demonstrates the results of each trajectory search algorithm when the query trajectory does not follow a straight line along the road, but instead detours under the red car in the parking lot. The top-1 result (red box) of the exact trajectory search algorithm presents a trajectory that passes under the red car, despite the difference with the query trajectory. This is because the mid-point of the query trajectory under the red car acts as a noise point, and it causes the underestimation of the trajectory distance. In contrast, the top-1 result (blue box) of our noise-tolerant trajectory search algorithm shows a trajectory that is semantically closer to the query trajectory. This example suggests that when a query trajectory with some irregular mid-points is given, our noise-tolerant trajectory search algorithm can provide more meaningful and robust results than the exact trajectory search algorithm.

VI. CONCLUSION

In this paper, we presented a novel approach for computing the noise-tolerant distance between trajectories, assuming the presence of noise that sporadically appears in video surveillance applications. In contrast to previous studies, we attempted to obviate noise points inherent in trajectories during the distance computation without the need for additional efforts. Specifically, we developed a novel structure, R^{on}-tree, a variant of R^{o}-tree, that detaches permanent noise-like points from ordinary points during the point insertion process and keeps them in a separate auxiliary tree. We presented NT-Inc-HausDist, a modification of the incremental algorithm for the Hausdorff distance, that exploits R^{on}-tree for preventing noise-like points from being involved in the computation process to achieve noise-tolerant trajectory distance computation. In addition, we employed the noise-tolerant distance for trajectory search, called NT-TrajSearch. The experimental results demonstrated that on noisy trajectories, the distance produced by our proposed approach was closer to the true distance than that produced by any other approach. Moreover, we applied our noise-tolerant trajectory search to a real video surveillance dataset and showed the effectiveness of our noise-tolerant trajectory distance.

Although this paper specifically focuses on video surveillance applications, our noise-tolerant trajectory distance can be applied to trajectories in other fields as well. For example, it is worth considering the application of our method to GPS-based trajectories [26], which often provide incorrect locations, especially if the signals from satellites are blocked by walls or buildings [62]. Moreover, we expect that our approach could be applied to trajectories generated from various IoT devices. As part of future work, we aim to enhance the structure of R^{on}-tree to ensure the tighter lower/upper bounds of the noise-tolerant trajectory distance, increasing the performance of the proposed algorithms. In addition, we aim to enhance the quality of R^{on}-tree by decreasing the number of false negatives, i.e., ordinary points that are regarded as noise-like points, as discussed in Section III-B. We also plan to develop noise-tolerant computations for other distances, such as the Fréchet distance [63], which is also known to be noise-sensitive [57], [64].

APPENDIX A BRANCH-AND-BOUND ALGORITHMS

To compute the directed Hausdorff distance $h(A, B)$ for given trajectories A and B, the two branch-and-bound algorithms, DF-HausDist and BF-HausDist [35], first traverse an R-tree for A (called R_A) in a depth-first and best-first manner, respectively. Starting at the root node of R_A, we compute an upper bound of the distance from each child node or point of the exploring node to the root node of an R^{on}-tree for B (called R_B), and we then sort the nodes or points by the upper bound in descending order. We explore the one that has the highest upper bound and repeat the same process until we reach a point. If a point has the highest upper bound, then we traverse R_B in a best-first manner to compute the distance

Algorithm 5 NT-DistToNN(P, R_B)

```
    Input: Point $P_A$, R$^{on}$-tree $R_B$ for a trajectory $B$
    Output: Noise-tolerant distance from $P_A$ to $B$
 1: $PQ \leftarrow$ Create a priority queue in "ascending order"
 2: $PQ.push(R_B.root(), 0)$
 3: while $PQ$ is not empty do
 4:     $(N_B, d_N) \leftarrow PQ.pop()$
 5:     if $N_B$ is a leaf node then
 6:         for each point $P_B$ in $N_B$ do
 7:             $d \leftarrow dist(P_A, P_B)$
 8:             $PQ.push(P_B, d)$
 9:         end for
10:     else if $N_B$ is an internal node then
11:         for each child node $C_B$ in $N_B$ do
12:             $d \leftarrow$ MinDist($P_A$, $C_B.shrunkenMBR()$)
13:             $PQ.push(C_B, d)$
14:         end for
15:     else                                        ▷ $N_B$ is a point
16:         return $d_N$
17:     end if
18: end while
```

from each point in N_A to B. If the distance is larger than any other upper bound, then it is returned as the distance from A to B. Theorem 2 ensures that DF-HausDist and BF-HausDist return the exact value of $h(A, B)$.

Theorem 2: If $A_1, \cdots, A_n$ are disjoint subsets of A and $A = \bigcup_{i=1}^{n} A_i$, then

$$h(A, B) = \max_{1 \le i \le n} h(A_i, B).$$

Proof: From the definition of $h(A, B)$,

$$\begin{aligned} h(A, B) &= \max_{a \in A} \left\{ \min_{b \in B} dist(a, b) \right\} \\ &= \max_{a \in \bigcup_{i=1}^{n} A_i} \left\{ \min_{b \in B} dist(a, b) \right\} \\ &= \max_{1 \le i \le n} \left[\max_{a_i \in A_i} \left\{ \min_{b \in B} dist(a_i, b) \right\} \right] \\ &= \max_{1 \le i \le n} h(A_i, B). \qquad \square \end{aligned}$$

The branch-and-bound algorithms for noise-tolerant directed trajectory distance computation, NT-DF-HausDist and NT-BF-HausDist, have almost identical structures to DF-HausDist and BF-HausDist, respectively. Similar to NT-Inc-HausDist, they also have the same differences from DF-HausDist and BF-HausDist: visiting only ordinary points, exploiting shrunken MBRs, and computing the looser lower/upper bounds.

We first begin with the function NT-DistToNN(P_A, R_B) for a point $P_A \in A$, which computes the noise-tolerant distance from P_A to B. As shown in Algorithm 5, NT-DistToNN traverses R_B in a best-first manner. We first create a priority queue PQ, which takes entries (node N_B, number d_N) and arranges them in ascending order with respect to d_N. Initially,

Algorithm 6 NT-DF-HausDist(A, B)

```
    Input: Trajectory $A$, Trajectory $B$
    Output: Noise-tolerant trajectory distance from $A$ to $B$
 1: return NT-DF-HausDist*($A$, $B$, 0)

 2: function NT-DF-HausDist*($A$, $B$, $LB$)
        Input: Trajectory $A$, Trajectory $B$, Lower bound $LB$
        Output: Maximum of $LB$ and noise-tolerant trajectory
        distance from $A$ to $B$
 3:     $(R_A, R_B) \leftarrow$ Create R$^{on}$-trees for $A$ and $B$
 4:     $MaxLB \leftarrow$ MinDist($R_A.root().shrunkenMBR()$,
                                $R_B.root().shrunkenMBR()$)
 5:     $MaxLB \leftarrow \max(MaxLB, LB)$
 6:     return NT-DF-HausDist$^\dagger$($R_A.root()$, $R_B$, $MaxLB$)
 7: end function

 8: function NT-DF-HausDist$^\dagger$($N_A$, $R_B$, $MaxLB$)
        Input: Node $N_A$ from $R_A$, R$^{on}$-tree $R_B$, Maximum lower
        bound $MaxLB$ obtained so far
        Output: Noise-tolerant trajectory distance from $N_A$ to $B$
 9:     $d_H \leftarrow 0$
10:     if $N_A$ is a point then
11:         return max($MaxLB$, NT-DistToNN($N_A$, $R_B$))
12:     else
13:         $LB \leftarrow$ MinDist($N_A.shrunkenMBR()$,
                                  $R_B.root().shrunkenMBR()$)
14:         $MaxLB \leftarrow \max(MaxLB, LB)$
15:         Create an empty list $L$
16:         if $N_A$ is a leaf node then
17:             for each point $P_A$ in $N_A$ do
18:                 $UB \leftarrow$ MaxDist($P_A$,
                                        $R_B.root().shrunkenMBR()$)
19:                 $L \leftarrow (P_A, UB)$
20:             end for
21:         else                               ▷ $N_A$ is an internal node
22:             for each child node $C_A$ in $N_A$ do
23:                 $UB \leftarrow$ MaxDist($C_A.shrunkenMBR()$,
                                        $R_B.root().shrunkenMBR()$)
24:                 $L \leftarrow (C_A, UB)$
25:             end for
26:         end if
27:         Sort $L$ in descending order
28:         for each element $(C_A, UB)$ in $L$ do
29:             if $UB \ge MaxLB$ then
30:                 $d \leftarrow$NT-DF-HausDist$^\dagger$($C_A$,$R_B$,$MaxLB$)
31:                 $d_H \leftarrow \max(d_H, d)$
32:                 $MaxLB \leftarrow \max(MaxLB, d_H)$
33:             else
34:                 return $d_H$
35:             end if
36:         end for
37:         return $d_H$
38:     end if
39: end function
```

we insert ($R_B.root()$, 0) into PQ and proceed to the while loop (Lines 3 to 18). At the beginning of each iteration, the entry (N_B, d_N) is dequeued from PQ. If N_B is a leaf node, then we compute the noise-tolerant distance d from P_A to each point P_B in N_B and insert the entry (P_B, d) into PQ. If N_B

Algorithm 7 NT-BF-HausDist(A, B)

```
    Input: Trajectory $A$, Trajectory $B$
    Output: Noise-tolerant trajectory distance from $A$ to $B$
 1: return NT-BF-HausDist*($A$, $B$, 0)

 2: function NT-BF-HausDist*($A$, $B$, $LB$)
        Input: Trajectory $A$, Trajectory $B$, Lower bound $LB$
        Output: Maximum of $LB$ and noise-tolerant trajectory distance from $A$ to $B$
 3:     $R_A \leftarrow$ Create an R$^{on}$-tree for $A$
 4:     $R_B \leftarrow$ Create an R$^{on}$-tree for $B$
 5:     $PQ \leftarrow$ Create a priority queue in "descending order"
 6:     $PQ.push(R_A.root(), \infty)$
 7:     while $PQ$ is not empty do
 8:         $(N_A, d_N) \leftarrow PQ.pop()$
 9:         if $d_N \leq LB$ then
10:             return $LB$
11:         end if
12:         if $N_A$ is a leaf node then
13:             for each point $P_A$ in $N$ do
14:                 $d \leftarrow$ NT-DistToNN($P_A$, $R_B$)
15:                 $PQ.push(P_A, d)$
16:             end for
17:         else if $N_A$ is an internal node then
18:             for each child node $C_A$ in $N_A$ do
19:                 $d \leftarrow$ MaxDist($C_A.shrunkenMBR()$,
                                 $R_B.root().shrunkenMBR()$)
20:                 $PQ.push(C_A, d)$
21:             end for
22:         else                                          ▷ $N_A$ is a point
23:             return $d_N$
24:         end if
25:     end while
26: end function
```

is an internal node, then we compute the lower bound of the distance from P_A to each child node C_B in N_B and we insert the entry (C_B, d) into PQ. The iteration continues until N_B is a point, at which time d_N is returned as the desired distance.

The depth-first algorithm NT-DF-HausDist is given by Algorithm 6. NT-DF-HausDist itself simply invokes the starred algorithm NT-DF-HausDist* with third argument 0 (Line 1). NT-DF-HausDist*, with two trajectories A, B and a nonnegative number LB, traverses R_A for A in a depth-first manner by calling a recursive function NT-DF-HausDist†. This function takes a node N_A, an R^{on}-tree R_B, and a nonnegative number $MaxLB$, which implies a lower bound of the distance obtained so far. In NT-DF-HausDist†, if N_A is a point (Lines 11 to 12), then we compute the maximum of $MaxLB$ and the result of NT-DistToNN(N_A, R_B). If N_A is a leaf node or an internal node (Lines 17 to 27), then we compute the upper bound of the distance from each point P_A in N_A or from each child node C_A in N_A to the shrunken MBR of $R_B.root()$, respectively. Concerning the upper bound as a key, we recursively traverse points or child nodes in N_A in descending order. During this process, we update d_H to the largest distance computed so far, and update $MaxLB$ to the maximum of d_H and the current lower bound (Lines 31 to 33). If points or nodes have an upper bound smaller than $MaxLB$, then we do not visit them since the distance produced by them cannot exceed $MaxLB$.

Algorithm 7 describes how NT-BF-HausDist works. Similar to NT-DF-HausDist, the starred algorithm NT-BF-HausDist* is called with third argument 0 at the beginning (Line 1). NT-BF-HausDist* takes the same arguments and has the same role as NT-DF-HausDist*, but it traverses R_A for A in a best-first manner. At first, a priority queue PQ, which keeps entries (node N_A, number d_N) in descending order, takes an entry $(R_A.root(), \infty)$. For each iteration of the while loop (Lines 7 to 25), the entry (N_A, d_N) is dequeued from PQ. If $d_N \leq LB$, then the algorithm terminates since d_N is the current upper bound of the distance so that the distance would be at most LB. If N_A is a leaf node, then we compute the noise-tolerant distance d from each point P_A in N_A to B by calling NT-DistToNN(P_A, R_B), and then insert the entry (P_A, d) into PQ. If N_A is an internal node, then we compute the upper bound of the distance from each child node C_A in N_A to the shrunken MBR of $R_B.root()$ and insert the entry (C_A, d) into PQ. The while loop proceeds until N_A is a point, at which time d_N is returned as the desired distance.

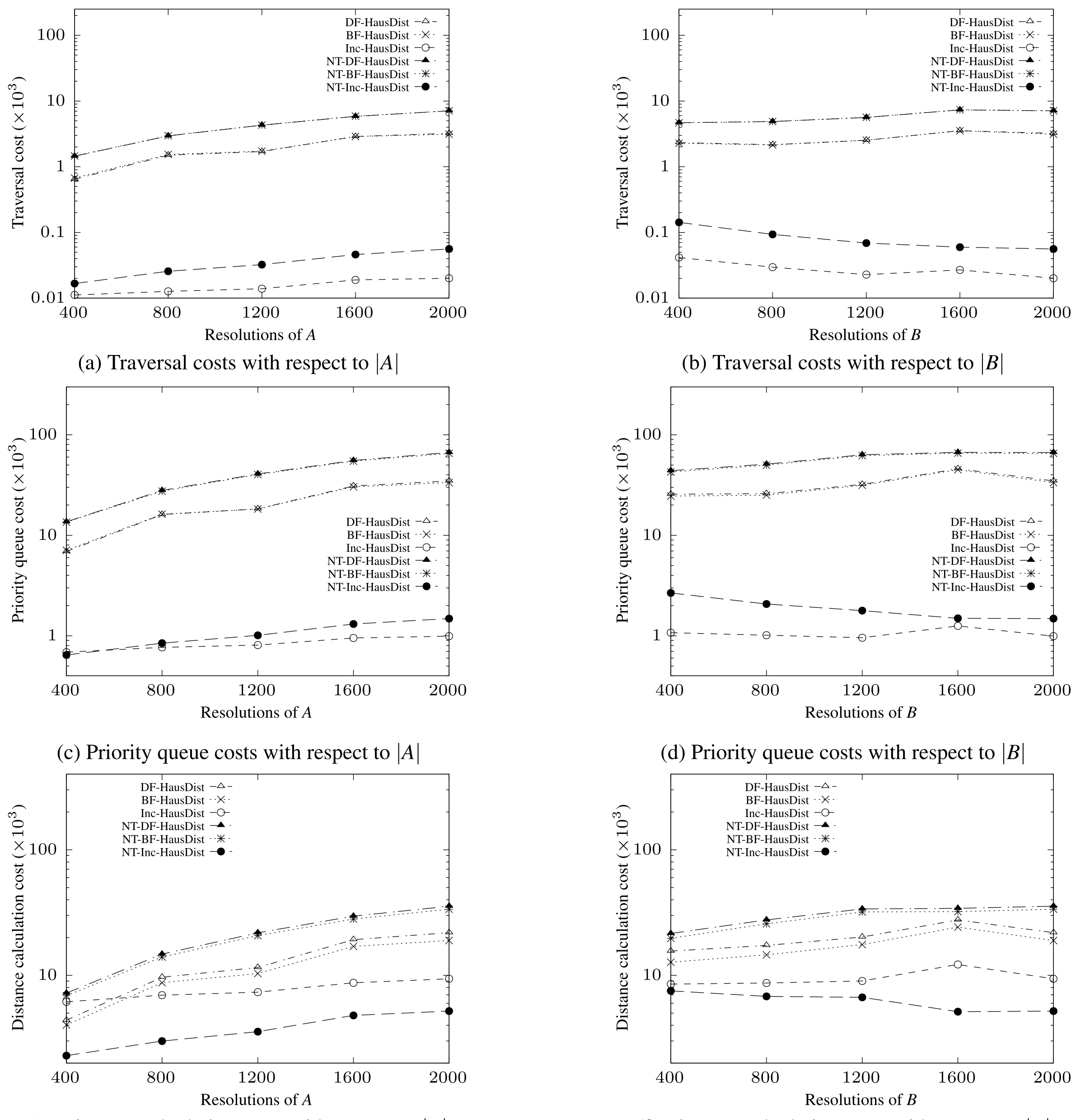
APPENDIX B PERFORMANCE MEASURES OF THE NOISE-TOLERANT TRAJECTORY DISTANCE COMPUTATION

Although we pay more attention to the closeness rather than the performance, it is anticipated that our noise-tolerant algorithms have reasonable performance. The purpose of this experiment is to investigate the behavior of the performance measures of our noise-tolerant algorithms (NT-DF-HausDist, NT-BF-HausDist, and NT-Inc-HausDist), compared to the original algorithms (DF-HausDist, BF-HausDist, and Inc-HausDist), respectively. The following performance measures, presented in [35], were used.

(a) *Tree traversal cost*: the number of R-tree, R^o-tree, or R^{on}-tree nodes visited.
(b) *Distance calculation cost*: the number of MinDist and MaxDist computations.
(c) *Priority queue cost*: the number of enqueue and dequeue operations.

We computed the performance measures according to different resolutions of A, while we set the resolution of B to 2,000, and vice versa. Each record was obtained by averaging the results of 200 different pairs of ordinary trajectories.

Figure 9 displays the performance measures of the three noise-tolerant algorithms compared to the original algorithms when both A and B are ordinary. The traversal costs and priority queue costs of NT-DF-HausDist, NT-BF-HausDist, and NT-Inc-HausDist are approximately twice as large as those of DF-HausDist, BF-HausDist, and Inc-HausDist, respectively. This is because the noise-tolerant algorithms compute lower and upper bounds by MinDist and MaxDist, so they visit more points or child nodes than the original algorithms that use HausDistLB and HausDistUB, which



(a) Traversal costs with respect to $|A|$

(b) Traversal costs with respect to $|B|$

(c) Priority queue costs with respect to $|A|$

(d) Priority queue costs with respect to $|B|$

(e) Distance calculation costs with respect to $|A|$

(f) Distance calculation costs with respect to $|B|$

FIGURE 9. Performance measures of the noise-tolerant algorithms when both *A* and *B* are ordinary trajectories. In the left column, the resolution of *A* varies, while the resolution of *B* is fixed, and conversely in the right column.

provide tighter lower and upper bounds for the trajectory distance, respectively. On the other hand, the distance calculation costs of NT-DF-HausDist and NT-BF-HausDist are still larger than those of DF-HausDist and BF-HausDist, but the performance gap becomes smaller than that of the other costs. In addition, the distance calculation cost of NT-Inc-HausDist is lower than that of Inc-HausDist. Although HausDistLB and HausDistUB provide tighter lower and upper bounds, they require much higher computation costs than MinDist and MaxDist. In particular, the distance calculation cost of Inc-HausDist per visited node is much higher than those of DF-HausDist and BF-HausDist, while NT-Inc-HausDist has a comparable distance calculation cost per visited node to NT-DF-HausDist and NT-BF-HausDist.

REFERENCES

[1] Z. Shao and Y. Li, "Integral invariants for space motion trajectory matching and recognition," *Pattern Recognit.*, vol. 48, no. 8, pp. 2418–2432, Aug. 2015, doi: 10.1016/j.patcog.2015.02.029.

[2] S. Wang, Z. Bao, J. S. Culpepper, T. Sellis, and X. Qin, "Fast large-scale trajectory clustering," *Proc. VLDB Endowment*, vol. 13, no. 1, pp. 29–42, Sep. 2019, doi: 10.14778/3357377.3357380.

[3] L. Li, S. Erfani, C. A. Chan, and C. Leckie, "Multi-scale trajectory clustering to identify corridors in mobile networks," in *Proc. 28th ACM Int. Conf. Inf. Knowl. Manag.* New York, NY, USA: Association for Computing Machinery, Nov. 2019, p. 2253, doi: 10.1145/3357384.3358157.

[4] D. Qiao, X. Yang, Y. Liang, and X. Hao, "Rapid trajectory clustering based on neighbor spatial analysis," *Pattern Recognit. Lett.*, vol. 156, pp. 167–173, Apr. 2022, doi: 10.1016/j.patrec.2022.03.010.

[5] J. Yang, Y. Liu, L. Ma, and C. Ji, "Maritime traffic flow clustering analysis by density based trajectory clustering with noise," *Ocean Eng.*, vol. 249, Apr. 2022, Art. no. 111001, doi: 10.1016/j.oceaneng.2022.111001.

[6] T. Wu, P. Lei, F. Li, and J. Chen, "Space-time tree search for long-term trajectory prediction," *IEEE Access*, vol. 10, pp. 117745–117756, 2022, doi: 10.1109/ACCESS.2022.3213691.

[7] M. Huynh and G. Alaghband, "Online adaptive temporal memory with certainty estimation for human trajectory prediction," in *Proc. IEEE/CVF Winter Conf. Appl. Comput. Vis. (WACV)*, Jan. 2023, pp. 940–949, doi: 10.1109/WACV56688.2023.00100.

[8] C. Yang and Z. Pei, "Long-short term spatio-temporal aggregation for trajectory prediction," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 4, pp. 4114–4126, Apr. 2023, doi: 10.1109/TITS.2023.3234962.

[9] J.-G. Lee, J. Han, X. Li, and H. Gonzalez, "TraClass: Trajectory classification using hierarchical region-based and trajectory-based clustering," *Proc. VLDB Endowment*, vol. 1, no. 1, pp. 1081–1094, Aug. 2008, doi: 10.14778/1453856.1453972.

[10] K. K. Santhosh, D. P. Dogra, P. P. Roy, and A. Mitra, "Vehicular trajectory classification and traffic anomaly detection in videos using a hybrid CNN-VAE architecture," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 8, pp. 11891–11902, Aug. 2022, doi: 10.1109/TITS.2021.3108504.

[11] Q. Gao, F. Zhou, T. Zhong, G. Trajcevski, X. Yang, and T. Li, "Contextual spatio-temporal graph representation learning for reinforced human mobility mining," *Inf. Sci.*, vol. 606, pp. 230–249, Aug. 2022, doi: 10.1016/j.ins.2022.05.049.

[12] C.-C. Hung, W.-C. Peng, and W.-C. Lee, "Clustering and aggregating clues of trajectories for mining trajectory patterns and routes," *VLDB J.*, vol. 24, no. 2, pp. 169–192, Apr. 2015, doi: 10.1007/s00778-011-0262-6.

[13] D.-W. Choi, J. Pei, and T. Heinis, "Efficient mining of regional movement patterns in semantic trajectories," *Proc. VLDB Endowment*, vol. 10, no. 13, pp. 2073–2084, Sep. 2017, doi: 10.14778/3151106.3151111.

[14] Y. Kwon, K. Kang, J. Jin, J. Moon, and J. Park, "Hierarchically linked infinite hidden Markov model based trajectory analysis and semantic region retrieval in a trajectory dataset," *Exp. Syst. Appl.*, vol. 78, pp. 386–395, Jul. 2017, doi: 10.1016/j.eswa.2017.02.026.

[15] X. Li, K. Zhao, G. Cong, C. S. Jensen, and W. Wei, "Deep representation learning for trajectory similarity computation," in *Proc. IEEE 34th Int. Conf. Data Eng. (ICDE)*, Apr. 2018, pp. 617–628, doi: 10.1109/ICDE.2018.00062.

[16] T.-Y. Fu and W.-C. Lee, "Trembr: Exploring road networks for trajectory representation learning," *ACM Trans. Intell. Syst. Technol.*, vol. 11, no. 1, pp. 1–25, Feb. 2020, doi: 10.1145/3361741.

[17] C. Gao, Z. Zhang, C. Huang, H. Yin, Q. Yang, and J. Shao, "Semantic trajectory representation and retrieval via hierarchical embedding," *Inf. Sci.*, vol. 538, pp. 176–192, Oct. 2020, doi: 10.1016/j.ins.2020.05.107.

[18] C. Feng, Z. Pan, J. Fang, J. Xu, P. Zhao, and L. Zhao, "Aries: Accurate metric-based representation learning for fast top-k trajectory similarity query," in *Proc. 31st ACM Int. Conf. Inf. Knowl. Manag.* New York, NY, USA: Association for Computing Machinery, Oct. 2022, pp. 499–508, doi: 10.1145/3511808.3557239.

[19] A. D. Pazho, C. Neff, G. A. Noghre, B. R. Ardabili, S. Yao, M. Baharani, and H. Tabkhi, "Ancilia: Scalable intelligent video surveillance for the artificial intelligence of things," *IEEE Internet Things J.*, vol. 10, no. 17, pp. 14940–14951, Sep. 2023, doi: 10.1109/JIOT.2023.3263725.

[20] W. Liu, G. Wei, Y. Wang, and R. Wu, "Indoor multipedestrian multicamera tracking based on fine spatiotemporal constraints," *IEEE Internet Things J.*, vol. 10, no. 11, pp. 10012–10023, Jun. 2023, doi: 10.1109/JIOT.2023.3235148.

[21] M. Fernández-Sanjurjo, M. Mucientes, and V. M. Brea, "Real-time multiple object visual tracking for embedded GPU systems," *IEEE Internet Things J.*, vol. 8, no. 11, pp. 9177–9188, Jun. 2021, doi: 10.1109/JIOT.2021.3056239.

[22] J.-S. Ham, D. H. Kim, N. Jung, and J. Moon, "CIPF: Crossing intention prediction network based on feature fusion modules for improving pedestrian safety," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2023, pp. 3665–3674.

[23] H. Shin, K. Na, J. Chang, and T. Uhm, "Multimodal layer surveillance map based on anomaly detection using multi-agents for smart city security," *ETRI J.*, vol. 44, no. 2, pp. 183–193, Apr. 2022, doi: 10.4218/etrij.2021-0395.

[24] S. Y. Nikouei, Y. Chen, A. J. Aved, and E. Blasch, "I-ViSE: Interactive video surveillance as an edge service using unsupervised feature queries," *IEEE Internet Things J.*, vol. 8, no. 21, pp. 16181–16190, Nov. 2021, doi: 10.1109/JIOT.2020.3016825.

[25] J. Bian, D. Tian, Y. Tang, and D. Tao, "Trajectory data classification: A review," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 4, pp. 1–34, Jul. 2019, doi: 10.1145/3330138.

[26] Y. Chen, P. Yu, W. Chen, Z. Zheng, and M. Guo, "Embedding-based similarity computation for massive vehicle trajectory data," *IEEE Internet Things J.*, vol. 9, no. 6, pp. 4650–4660, Mar. 2022, doi: 10.1109/JIOT.2021.3107327.

[27] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge, "Comparing images using the Hausdorff distance," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, no. 9, pp. 850–863, Sep. 1993, doi: 10.1109/34.232073.

[28] J. Ribera, D. Güera, Y. Chen, and E. J. Delp, "Locating objects without bounding boxes," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 6472–6482, doi: 10.1109/CVPR.2019.00664.

[29] X. Wang, K. T. Ma, G.-W. Ng, and W. E. L. Grimson, "Trajectory analysis and semantic region modeling using nonparametric hierarchical Bayesian models," *Int. J. Comput. Vis.*, vol. 95, no. 3, pp. 287–312, Dec. 2011, doi: 10.1007/s11263-011-0459-6.

[30] M. Pálková, O. Uhlík, and T. Apeltauer, "Calibration of pedestrian ingress model based on CCTV surveillance data using machine learning methods," *PLoS ONE*, vol. 19, no. 1, Jan. 2024, Art. no. e0293679, doi: 10.1371/journal.pone.0293679.

[31] K. Liu, H. Ma, L. Zhang, Z. Cai, and H. Ma, "Strip adjustment of airborne LiDAR data in urban scenes using planar features by the minimum Hausdorff distance," *Sensors*, vol. 19, no. 23, p. 5131, Nov. 2019, doi: 10.3390/s19235131.

[32] B. H. Menze et al., "The multimodal brain tumor image segmentation benchmark (BRATS)," *IEEE Trans. Med. Imag.*, vol. 34, no. 10, pp. 1993–2024, Oct. 2015, doi: 10.1109/TMI.2014.2377694.

[33] M.-P. Dubuisson and A. K. Jain, "A modified Hausdorff distance for object matching," in *Proc. 12th Int. Conf. Pattern Recognit.*, 1994, pp. 566–568, doi: 10.1109/icpr.1994.576361.

[34] T. Xia and D. Zhang, "Improving the R*-tree with outlier handling techniques," in *Proc. 13th Annu. ACM Int. Workshop Geographic Inf. Syst.* New York, NY, USA: Association for Computing Machinery, Nov. 2005, pp. 125–134, doi: 10.1145/1097064.1097083.

[35] S. Nutanong, E. H. Jacox, and H. Samet, "An incremental Hausdorff distance calculation algorithm," *Proc. VLDB Endowment*, vol. 4, no. 8, pp. 506–517, May 2011, doi: 10.14778/2002974.2002978.

[36] D. Zhang, F. He, S. Han, L. Zou, Y. Wu, and Y. Chen, "An efficient approach to directly compute the exact Hausdorff distance for 3D point sets," *Integr. Comput.-Aided Eng.*, vol. 24, no. 3, pp. 261–277, Jul. 2017, doi: 10.3233/ica-170544.

[37] H.-J. Son, S.-H. Kim, and J.-S. Kim, "Text image matching without language model using a Hausdorff distance," *Inf. Process. Manag.*, vol. 44, no. 3, pp. 1189–1200, May 2008, doi: 10.1016/j.ipm.2007.11.004.

[38] A. Fischer, C. Y. Suen, V. Frinken, K. Riesen, and H. Bunke, "Approximation of graph edit distance based on Hausdorff matching," *Pattern Recognit.*, vol. 48, no. 2, pp. 331–343, Feb. 2015, doi: 10.1016/j.patcog.2014.07.015.

[39] D. Karimi and S. E. Salcudean, "Reducing the Hausdorff distance in medical image segmentation with convolutional neural networks," *IEEE Trans. Med. Imag.*, vol. 39, no. 2, pp. 499–513, Feb. 2020, doi: 10.1109/TMI.2019.2930068.

[40] S. Cho, J. Paeng, and J. Kwon, "Densely-packed object detection via hard negative-aware anchor attention," in *Proc. IEEE/CVF Winter Conf. Appl. Comput. Vis. (WACV)*, Jan. 2022, pp. 1401–1410, doi: 10.1109/WACV51458.2022.00147.

[41] Y. Gao, M. Wang, R. Ji, X. Wu, and Q. Dai, "3-D object retrieval with Hausdorff distance learning," *IEEE Trans. Ind. Electron.*, vol. 61, no. 4, pp. 2088–2098, Apr. 2014, doi: 10.1109/TIE.2013.2262760.

[42] M. Narendra, M. L. Valarmathi, and L. J. Anbarasi, "Watermarking techniques for three-dimensional (3D) mesh models: A survey," *Multimedia Syst.*, vol. 28, no. 2, pp. 623–641, Apr. 2022, doi: 10.1007/s00530-021-00860-z.

[43] G. Mastorakis, T. Ellis, and D. Makris, "Fall detection without people: A simulation approach tackling video data scarcity," *Exp. Syst. Appl.*, vol. 112, pp. 125–137, Dec. 2018, doi: 10.1016/j.eswa.2018.06.019.

[44] M.-E. Yadamjav, Z. Bao, B. Zheng, F. M. Choudhury, and H. Samet, "Querying recurrent convoys over trajectory data," *ACM Trans. Intell. Syst. Technol.*, vol. 11, no. 5, pp. 1–24, Oct. 2020, doi: 10.1145/3400730.

[45] Y. Zheng, "Trajectory data mining: An overview," *ACM Trans. Intell. Syst. Technol.*, vol. 6, no. 3, pp. 1–41, May 2015, doi: 10.1145/2743025.

[46] A. A. Taha and A. Hanbury, "An efficient algorithm for calculating the exact Hausdorff distance," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 11, pp. 2153–2163, Nov. 2015, doi: 10.1109/TPAMI.2015.2408351.

[47] Y. Chen, F. He, Y. Wu, and N. Hou, "A local start search algorithm to compute exact Hausdorff distance for arbitrary point sets," *Pattern Recognit.*, vol. 67, pp. 139–148, Jul. 2017, doi: 10.1016/j.patcog.2017.02.013.

[48] D. Zhang, L. Zou, Y. Chen, and F. He, "Efficient and accurate Hausdorff distance computation based on diffusion search," *IEEE Access*, vol. 6, pp. 1350–1361, 2018, doi: 10.1109/ACCESS.2017.2778745.

[49] J. Ryu and S.-I. Kamata, "An efficient computational algorithm for Hausdorff distance based on points-ruling-out and systematic random sampling," *Pattern Recognit.*, vol. 114, Jun. 2021, Art. no. 107857, doi: 10.1016/j.patcog.2021.107857.

[50] W.-C. Lee and J. Krumm, "Trajectory preprocessing," in *Computing With Spatial Trajectories*. New York, NY, USA: Springer, 2011, pp. 3–33, doi: 10.1007/978-1-4614-1629-6_1.

[51] T. He, J. Bao, R. Li, S. Ruan, Y. Li, C. Tian, and Y. Zheng, "Detecting vehicle illegal parking events using sharing bikes' trajectories," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*. New York, NY, USA: Association for Computing Machinery, Jul. 2018, pp. 340–349, doi: 10.1145/3219819.3219887.

[52] B. Custers, M. van de Kerkhof, W. Meulemans, B. Speckmann, and F. Staals, "Maximum physically consistent trajectories," in *Proc. 27th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.* New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 79–88, doi: 10.1145/3347146.3359363.

[53] D. Zhang and T. Xia, "A novel improvement to the R*-tree spatial index using gain/loss metrics," in *Proc. 12th Annu. ACM Int. Workshop Geographic Inf. Syst.* New York, NY, USA: Association for Computing Machinery, Nov. 2004, pp. 204–213, doi: 10.1145/1032222.1032253.

[54] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*. New York, NY, USA: Association for Computing Machinery, May 1990, pp. 322–331, doi: 10.1145/93597.98741.

[55] H. Samet, *Foundations of Multidimensional and Metric Data Structures*. Burlington, MA, USA: Morgan Kaufmann, 2006.

[56] T. Brinkhoff, "A framework for generating network-based moving objects," *GeoInformatica*, vol. 6, no. 2, pp. 153–180, 2002, doi: 10.1023/A:1015231126594.

[57] H. Su, S. Liu, B. Zheng, X. Zhou, and K. Zheng, "A survey of trajectory distance measures and performance evaluation," *VLDB J.*, vol. 29, no. 1, pp. 3–32, Jan. 2020, doi: 10.1007/s00778-019-00574-9.

[58] K. Fernande, P. Gharani, and V. Raghu, "TRAJEDI: Trajectory dissimilarity," in *Sustainable Interdependent Networks II*. Cham, Switzerland: Springer, 2019, pp. 135–146, doi: 10.1007/978-3-319-98923-5_8.

[59] H. Du, L. Chen, J. Qian, J. Hou, T. Jung, and X.-Y. Li, "PatronuS: A system for privacy-preserving cloud video surveillance," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1252–1261, Jun. 2020, doi: 10.1109/JSAC.2020.2986665.

[60] Y. Tao, K. Yi, C. Sheng, and P. Kalnis, "Efficient and accurate nearest neighbor and closest pair search in high-dimensional space," *ACM Trans. Database Syst.*, vol. 35, no. 3, pp. 1–46, Jul. 2010, doi: 10.1145/1806907.1806912.

[61] D. Xu, R. Hu, Z. Xiong, Z. Wang, L. Luo, and D. Li, "Trajectory is not enough: Hidden following detection," in *Proc. 29th ACM Int. Conf. Multimedia*. New York, NY, USA: Association for Computing Machinery, Oct. 2021, pp. 5373–5381, doi: 10.1145/3474085.3475664.

[62] B. Park and S. Lee, "Robust range-only beacon mapping in multipath environments," *ETRI J.*, vol. 42, no. 1, pp. 108–117, Feb. 2020, doi: 10.4218/etrij.2018-0614.

[63] H. Alt and M. Godau, "Computing the fréchet distance between two polygonal curves," *Int. J. Comput. Geometry Appl.*, vol. 5, pp. 75–91, Mar. 1995, doi: 10.1142/s0218195995000064.

[64] Y. Tao, A. Both, R. I. Silveira, K. Buchin, S. Sijben, R. S. Purves, P. Laube, D. Peng, K. Toohey, and M. Duckham, "A comparative analysis of trajectory similarity measures," *GISci. Remote Sens.*, vol. 58, no. 5, pp. 643–669, Jul. 2021, doi: 10.1080/15481603.2021.1908927.

YONGJIN KWON received the B.S. degree in computer science and engineering from POSTECH, Republic of Korea, in 2009, and the M.S. degree in computer science and engineering from Seoul National University, Republic of Korea, in 2012. He is currently a Senior Researcher with the Visual Intelligence Research Section, Superintelligence Creative Research Laboratory, Electronics and Telecommunications Research Institute (ETRI), Republic of Korea. His research interests include machine learning, information theory, and human behavior forecasting.

JINYOUNG MOON received the B.S. degree in computer engineering from Kyungpook National University, Daegu, Republic of Korea, in 2000, and the M.S. degree in computer science and the Ph.D. degree in industrial and systems engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Republic of Korea, in 2002 and 2018, respectively. She is currently a Principal Researcher with the Visual Intelligence Research Section, Superintelligence Creative Research Laboratory, Electronics and Telecommunications Research Institute (ETRI), Republic of Korea. Her research interests include machine learning, video understanding, video action detection and anticipation, and temporal moment localization by natural language.

YEONSEUNG CHUNG received the B.S. degree in statistics from Korea University, Seoul, Republic of Korea, in 2000, and the M.S. degree in biostatistics and the Ph.D. degree in biostatistics from the University of North Carolina at Chapel Hill, USA, in 2005 and 2009, respectively. She is currently an Associate Professor with the Department of Mathematical Sciences, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Republic of Korea. Her research interests include nonparametric Bayesian modeling, environmental epidemiology, and other statistical applications in biomedical research.