LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-CONF-636353

Big Data Scalability Issues in WAAS

J. Prokaj, X. Zhao, J. Choi, G. Medioni

May 10, 2013

Big Data Computer Vision (BDCV) 2013
Portland, OR, United States
June 24, 2013 through June 24, 2013

Disclaimer

Big Data Scalability Issues in WAAS*

Jan Prokaj, Xuemei Zhao, Jongmoo Choi, Gérard Medioni
University of Southern California
Los Angeles, CA 90089
{prokaj,xuemeiz,jongmooc,medioni}@usc.edu

Abstract

Wide Area Aerial Surveillance (WAAS) produces very large images at 1-2 fps or more. This data needs to be processed in real time to produce semantically meaningful information, then queried efficiently. We have designed and implemented a full system to detect and track vehicles, and infer activities. We address here the scalability issues, and propose solutions to have the tracker run in real time using different parallelism strategies. We also describe methods to efficiently query the data in forensic mode. Our methods are validated on large scale real world data, and have been transferred to a National Laboratory for deployment.

Figure 1. Example of WAAS imagery, full frame (left) and detail (right).

1. Introduction

The increased use of unmanned aerial vehicles, or drones, for aerial surveillance is resulting in large amounts of collected imagery. As this imagery often covers a geographic area of a few square kilometers, it is named Wide Area Aerial Surveillance (WAAS) imagery. This data is characterized by its large format (60-100 megapixels in every frame), multi-sensor capture, low sampling rate, and is in grayscale (see Figure 1). All of these properties pose challenges for computer vision analysis, but the main challenge we are concerned with here is the sheer amount of data to process, currently reaching about 100 megapixels a second, and going to nearly 2 gigapixels 10 times a second in the future. Simply storing the data is a challenge, not to mention semantic analysis, such as the detection and tracking of moving objects, and recognition of activities, which is our goal. These tasks have been studied in computer vision for a long time, but what has not received as much attention, is also accomplishing these tasks at frame rates.

As other big data problems, efficient and scalable algorithms and distributed computation are part of our solutions. Keck et al [7] proposed a distributed architecture for real-time tracking of vehicles in WAAS imagery, and adopted a classic multiple hypothesis tracker for tracking. We use a similar architecture with tile-to-tile handoff, but propose a modern and more efficient multiple-target tracking algorithm that avoids the computation of all possible data association hypotheses [12]. It accomplishes this by formulating a labeling problem, whose solution significantly reduces the data association search space. Other trackers for WAAS imagery have been proposed [11, 18, 15], but they have not been shown to work in a distributed environment, and process large-scale imagery at real-time speeds. The tracker of [12] is in fact in regular use by a national lab. To minimize the number of false alarms, we estimate motion patterns and integrate them into the tracker. The estimation of motion patterns is also designed for a distributed environment. Finally, we insert the estimated trajectories into a database and propose to use a scalable ERM framework for activity representation and inference. The ERM-based activity inference framework [5] enables us to utilize highly optimized scalable RDBMSs.

The distributed computation takes different form in different modules, and involves spatial or temporal division of the data. Whenever data are divided this way, the key is to ensure that you get the same result as without the division. Because some processes are interdependent, simple distributed massive parallelism is limited.

*Prepared by LLNL under Contract DE-AC52-07NA27344. Release #: LLNL-CONF-636353.

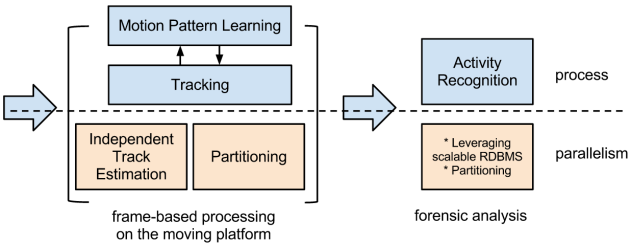

Figure 2. Overview of the proposed approach.

The contributions of this paper are a multiple-target tracker designed to handle large-scale problems, an algorithm for large-scale motion pattern estimation, and an activity inference framework for large trajectory datasets. Our approach is illustrated in Figure 2. Our proposed tracker along with integrated motion patterns can process WAAS imagery in real-time, generating tracking results with minimal number of errors. We analyze the scalability of the ERM based activity recognition, explain standard optimization techniques for large databases, and propose some task-specific strategies for the parallelism. We demonstrate that many activities can be effectively and efficiently defined and inferred using spatial and temporal partitioning techniques.

Our paper is organized as follows. In the next section we discuss our approach, explaining each module in detail, as well as our techniques for distributed computation. Then we present our results, and conclude the paper.

2. Our Approach

2.1. Tracking

We first stabilize the video stream, which is straightforward to implement in real time [6]. Our goal is to track all moving objects in the video from frame to frame. We adopt the efficient approach presented in [12], which determines the tracklets optimally by maximizing the joint probability of a set of detections over a temporal window. We extend it to take advantage of learned motion patterns during data association by incorporating motion pattern priors in the joint probability distribution as presented in [14]. This work is now briefly reviewed.

We take a hierarchical approach to tracking. That is, we first estimate short tracks, or tracklets, over a short temporal window and then associate them with an existing set of tracks. The input to our algorithm is a set of object detections (blobs) in each frame, which we estimate using background subtraction. Each detection in the first frame of the window is a potential object. Therefore, we find an optimal tracklet, or a set of tracklets, starting at each detection in the first window frame. This is not a problem, because for detections that are false alarms, the model of a valid tracklet (consistency of motion and appearance) is not satisfied, and the tracklet is discarded. Tracklets that start in the second or later frame of the window are found when the sliding window shifts to that frame.

Given a detected object in one frame, we know there must be another detected instance of that object located nearby in subsequent frames. By recursively applying this idea, a directed acyclic graph called an association graph, or detection graph is constructed. This graph stores all possible associations of object detections over time. The number of these associations (greater than or equal to the number of paths from the root down to leaves) is large and intractable to evaluate in entirety. The key idea of [12] is to first remove inconsistent detections in this graph by solving a binary labeling problem, which is very efficient. Once this has been done, the search space of paths in this tree is significantly reduced, and the few (often one) remaining possible paths are easily extracted. Missed detections, due to occlusion or background subtraction failure, are easily handled in this framework by generating virtual detections in this graph. The binary labeling problem is posed as MAP inference in a Bayesian network, constructed directly from the detection graph. This formulation allows easy incorporation of other evidence, such as motion patterns, into the problem [14].

2.1.1 Distributed Computation

Wide area imagery often covers a region of several square kilometers, which means the number of vehicles that needs to be tracked is in the thousands. In order to achieve real-time processing at this scale, the tracking task needs to be parallelized and divided among several processors. Since tracklets are estimated independently for each detection in the first frame of the temporal window, the estimation can proceed in parallel. This includes the construction of the association graph, solving the binary labeling problem, and extraction of tracklet(s) from the labeled graph. In our implementation, we launch as many threads as the number of available CPU cores, create a queue of detections for which tracklets should be estimated, and then let all threads process the queue until it is empty, each at its own pace. Even though tracklets are estimated independently, since we use a long temporal window for tracklet inference, the number of id switches is limited. At the same time, this independent estimation allows us to exploit parallelism to a great degree. This allows us to run the tracker at real-time speeds for imagery up to about $2K \times 2K$ in size (depending on the number of detections in each frame). For imagery larger than this, as WAAS imagery, multi-threaded processing is not enough, and we turn to distributed computation.

A natural way to divide up the work is to create a grid of "tiles", each covering an equal area of the monitored region (we assume the imagery has been stabilized and georegistered), and run an independent tracker on each one. This is

Figure 3. Parallel estimation of tracks and motion patterns on a cluster of computers is enabled by creating an overlapping grid of tiles.

illustrated in Figure 3. Even though each tracker estimates tracks on only a small portion of the region under surveillance, we can avoid track fragmentation arising from the grid by handing off, or linking, tracks for targets that move from the field of view of one tile to the next. This is facilitated by including a small overlap region. The size of the tiles should be set according to the tracking algorithms resource requirements. For example, we have experimented with tiles 2176×2176 in size.

We have implemented a simple track linking approach that works as follows. Whenever a new track is initiated in an overlap region, the initiating tile sends a "new-track" message to the overlapping neighbors, which contains the track's ID and its initial trajectory. At the same time, each tile maintains a set of tracks that have terminated in an overlap region. On every frame, the set of terminated tracks is matched against the set of new tracks. Whenever a terminated track matches with a new track, a "hand-off track" message is sent with the trajectory of the terminated track to the tile containing the new track. The historical trajectory is then merged into the new track. Matching of terminated tracks with new tracks can be done with the Hungarian algorithm for robustness. However, we have found that having a reasonable overlap region allows a greedy track matching algorithm with no loss in accuracy.

2.1.2 Experiments

We have evaluated the multi-target tracker on a publicly available dataset from Air Force Research Laboratory [3]. This dataset was captured by an array of six cameras at roughly 1 Hz, and it is in grayscale. We mosaicked the dataset using [13], stabilized it, and georeferenced it prior to tracking. A reference with a resolution of 0.30 meters (1 foot) per pixel was used, making vehicles, our targets of interest, about 20×10 pixels in size. A 1024-frame sequence of a 1408×1408 region was selected for evaluation. The dataset includes ground truth, which was manually generated, and contains 403 tracks in the region of interest.

Metrics	Performance
Object Detection Rate	0.36
False Alarm Rate	1.03
Track Swaps	0.48
Track Breaks	0.64
FPS non-distributed (no handoff)	8.83
FPS non-distributed (with handoff)	7.53
FPS distributed on 4 nodes*	11.2

Table 1. Quantitative evaluation of the proposed algorithm. *The distributed runtime was estimated.

Several metrics were used to measure performance: object detection rate, false alarm rate, mean cumulative swaps of tracks, and mean cumulative broken tracks. Object detection rate is defined as the fraction of detections in the ground truth found in the estimated tracks. False alarm rate is defined as the average number of false detections in estimated tracks in every frame. Mean cumulative swaps of tracks is defined as the average number of swaps (ID switches) in every ground truth track (over its lifetime). Mean cumulative broken tracks is defined as the average number of breaks in every ground truth track (over its lifetime). A break happens when a ground truth track is not matched to any ID in the next frame. These last two definitions are based on [16]. Computational efficiency was measured by the average number of frames processed per second (FPS) on an AMD FX-6300 CPU. This was measured with "cold cache", to reflect real-world conditions where the tracker is running in real-time and has not seen the dataset before. The results are shown in Table 1.

The results show that the proposed tracking algorithm is very good at making data associations. It makes a small number of tracking errors, such as id-switches and breaks. The detection rate is lower than we would expect, primarily due to the tracker not being able to track vehicles after they stop.

It is also clear that the proposed tracking algorithm is efficient. When the tracker runs on the full 1408×1408 imagery on one node, where no hand-off between tiles is needed, it can process the dataset at 8.83 frames per second, which is real-time considering the dataset was captured at 1 Hz. When the imagery is divided into 4 tiles, each 754×754 in size, and a 100 pixel common region between them, the tracker can process the dataset at 7.53 frames per second on one node. The overhead of 0.020 sec every frame is due to the necessary handoff processing. To estimate the performance of the tracker running on a 4-node cluster, we determined the maximum runtime of the tracker on a 754×754 tile, which was 11.9 frames per second. Then, we assumed the handoff cost would be uniformly distributed across nodes, a reasonable assumption, considering the cost is proportional to the number of tracks in each tile.

Taking this into account, the final runtime on a 4 node cluster would be 0.084 sec + 0.020 sec / 4 = 0.089 sec = 11.2 frames per second. Again, this shows we are able to run the tracker in real-time on a 4-node cluster.

2.2. Motion Flow Estimation from a Big Video

Multiple objects undergoing coordinated movements produce motion patterns [19, 20]. Motion patterns are important in video analysis because they convey rich information of the scene. For instance, motion patterns contain the information of moving objects regular movement, i.e., direction, speed, and they assist the detection of different movement. In wide area aerial surveillance videos, motion patterns are usually clear and informative, and they enable us to improve the tracker.

2.2.1 Motion Pattern Learning

When tracklet points are embedded into (x, y, v_x, v_y) feature space, points form into clusters, and manifold structures emerge [19, 20]. These manifolds are corresponding to motion patterns. In the learning framework, tracklets are first extracted [12] and used as input. In aerial scenes, parallax causes many false tracklets. Compared to the positive tracklets caused by real moving objects, false tracklets are usually distinguishable. For example, they are inconsistent and short, and the area scanned by the movement is usually small. So a large portion of false tracklets can be removed due to these properties. Once tracklets are pre-filtered, for each point on the remaining tracklets, velocity (v_x, v_y) is calculated, and 2D (x, y) information is transformed to (x, y, v_x, v_y) feature space, where points form into clusters, corresponding to the road networks in the aerial scene. Tensor Voting is then used to explore the geometric properties of the structures.

Tensor Voting Tensor Voting is a computational framework to estimate geometric information in N-D space. In this section, we only introduce the fundamental concepts related to our application, readers can refer to [9, 10] for the complete presentation and implementation details.

Suppose we have a set of samples in a high dimensional space, and these samples lie on a manifold of much lower dimension. Our objective is to infer the geometric structure of this manifold. In other words, we try to find the vectors that span the normal and tangent space at each point, and use them to characterize the manifold. Tensor voting is an unsupervised approach to estimate a structure tensor T at each point. Here, T is a rank-2, symmetric tensor, whose quadratic form is a symmetric, nonnegative definite matrix, representing underlying geometry.

Recall that a tensor can be decomposed as,

$$\begin{aligned} \boldsymbol{T} &= \sum_{i=1}^{N} \lambda_i \boldsymbol{e_i} \boldsymbol{e_i}^T \\ &= \sum_{i=1}^{N-1} (\lambda_i - \lambda_{i+1}) \sum_{k=1}^{i} \boldsymbol{e_k} \boldsymbol{e_k}^T + \lambda_N \sum_{i=1}^{N} \boldsymbol{e_i} \boldsymbol{e_i}^T \end{aligned} \tag{1}$$

where $\{\lambda_i\}$ are the eigenvalues arranged in descending order, and $\{e_i\}$ are the corresponding eigenvectors, and N is the dimensionality of the input space. Equation 1 provides a way to interpret the local geometry from T. The maximum difference between two consecutive eigenvalues, $\lambda_i - \lambda_{i+1}$, encodes the saliency of certain structure, whose normal space is i-D and the tangent space is $N - i$-D. λ_N is the saliency of an unoriented structure. Summing the saliency together for all i, λ_1 is an estimate of the probabilities for all possible manifold structures.

Outliers are usually brought in by false tracklets or false associations on the positive tracklets, and they indicate false structures in motion feature space. Compared to inliers which form clusters, outliers receive inconsistent and little support from their neighbors, and have low saliency. Thus, all points are ranked according to their λ_1, and the bottom ones are filtered out.

2.2.2 Distributed Computation

In order to achieve real-time processing at WAAS scale, a natural way is to divide up the work to create a grid of spatial tiles, each covering a certain area of the monitored region (we assume the imagery has been stabilized and georegistered), and Tensor Voting can be performed independently on each one.

Choosing tile size is important. It depends on the resolution of input data, and the coverage of the map. Once proper tile size is used, the number of points in the volume of each tile is both large enough to perform informative denoising and local structure inference, and small enough to achieve high efficiency. Also, the tiles should have small overlap regions. The proposed method is in nature suitable for parallel processing by dividing the data space into small volumes. That's because Tensor Voting is based on local neighborhood information, instead of the global structure. As shown in Figure 3, we first divide the spatial space into non-overlapping regions (in blue), then depending on the voting scale which determines the neighborhood size, we consider a larger area (in red) for each tile to perform Tensor Voting. After independent processing for each tile, only the results within the blue region of each tile is kept. Thus, the results are stitched back together with no need of further processing. The voting scale is determined empirically and is independent of the tile size.

Distributed Computation is more efficient from two perspectives. First, for each point, there is a smaller candidate pool for selecting nearest neighbors, since only the points in the tiles need to be considered. Second, the tiles are completely independent from each other, so that they can be processed by different processors simultaneously, and there is no further step to stitch the results back together. Assume we divide the whole region into $M * N$ tiles, distributed Tensor Voting is roughly M^2N^2/c^2 times faster than processing the whole region together. c is a factor indicating the effect of overlap region, and c is larger as M and N increase. Intuitively, the larger M and N are, the smaller each region is, and the larger the overlap region is needed.

Our framework can also handle online incoming long sequences. For every Y frames, the first X frames are used to learn and update motion patterns, suppose they are $\{X_i\}, i = 1, 2, 3, ,$ and each contains N_x points. From X_1, initial motion patterns can be learned. Instead of combining all the tracklet points from X_1 and X_2 together to learn motion patterns, we perform Tensor Voting in an incremental way. That means points from X_1 do not vote to each other, but vote with points from X_2, and points from X_2 vote for each other. Once $\{X_i\}, i = 1, 2, 3, k$ are processed, and X_{k+1} comes, the number of vote we need to perform is $2k * N_x * N_x + N_x^2$, instead of $(k + 1)^2 * N_x^2$. Therefore, motion patterns can be updated in an efficient online fashion.

2.2.3 Experiments

We have evaluated our motion pattern learning algorithm on a sequence from another wide area imagery dataset from Air Force Research Laboratory [1]. The dataset is captured at about 2 frames per second and contains significant parallax from campus buildings and trees. We have mosaicked, stabilized, and georeferenced the dataset to 0.75 meters per pixel resolution before tracking. For quantitative evaluation we selected a 1312×738 region in the middle of the persistently visible area and manually determined tracking ground truth for 100 frames. The selected sequence has 205 tracks of vehicles, each being about 10×5 pixels in size.

Given an input tracklet, we first calculate the movement of every tracklet point compared to the start of the tracklet, and remove the tracklet if the median of the movement is smaller than 6 pixels. That is because the tracklet points caused by parallax are often constrained in a small region, while tracklets caused by real moving objects occupy a large area. In outlier filtering, all points are ranked according to their λ_1, and the bottom 10% are filtered out. Figure 4 shows the motion pattern learning results, which more or less correspond to the road network.

For quantitative evaluation, we integrated the learned motion patterns into the tracker and compared tracking performance to a tracker without them. Several metrics were used to evaluate performance: object detection rate (ODR), false alarm rate (FAR), mean cumulative swaps of tracks (SWP), and mean cumulative broken tracks (BRK). Object detection rate, track swaps, and track breaks are defined the same way as in the previous section. However, FAR is defined as the number of false positive detections divided by the total number of estimated detections. The results are shown in Table 2.

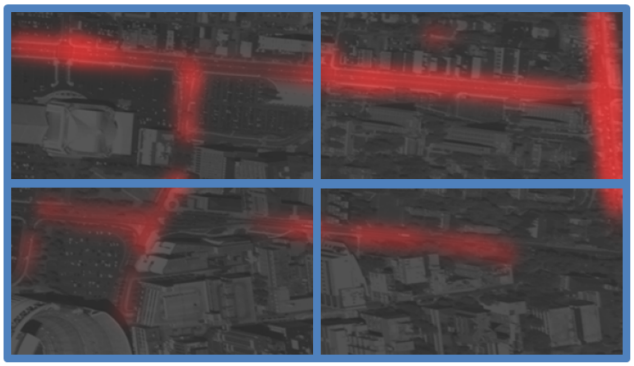

Figure 4. Parallel estimation results of motion patterns.

	Without MP	With MP
ODR	0.28	0.23
FAR	0.85	0.19
SWP	0.55	0.35
BRK	0.75	0.46

Table 2. Vehicle tracking performance with and without motion patterns (MP) on wide area imagery. Please see the text for metric definitions.

The results show that motion patterns significantly reduce the false alarm rate with a small corresponding decrease in the object detection rate. This is because most of the false alarms come from moving objects detections due to parallax and these are denoised by tensor voting. Furthermore, the number of ID switches and track fragmentation has also decreased with the use of motion patterns, as is evident in the decrease in the track swap rate and broken tracks rate. It's another indication that ambiguity during tracking has been reduced.

In terms of efficiency, the proposed algorithm processes the 1312×738 region as a whole at 1.89 frames per second. When the region is divided into 4 overlapping tiles, each 756×469 in size, the algorithm is able to process the data at 16.8 frames per second. This efficiency is determined by the longest processing time of each of the tiles, and there is no extra cost to combine the results from tiles. Since the data is captured at about 2 frames per second, it shows that our algorithm is able to process in real-time in a distributed way. Moreover, it's worth noting that learning motion patterns is an independent step of tracking, which means it doesn't

Table 3. Activity type: locality and the number of actors.

Types	Single actor	Multiple actors
local	U-turn, 2/3 point turn, on-road-X, Speeding, Stop-violation, Entry, Visit, Stay	source, sink, following, convoy, Brushpass, Dead drop, Coordinated movement
global	Loop, Traveling A to B	double meeting

slow down real time online tracking, but once it's done, the results can be integrated into the tracking module.

2.3. Large-Scale Aerial Activity Recognition

Our activity inference method is simple but very efficient [5]. Activities are defined as vehicular tracks associated with certain properties (e.g., U-turn, 3-point turn, loop, convoy, following, speeding). We extract a set of atomic portions of a track (we call it tracklets) from video input, along with physical attributes, and store to a standard RDBMS (Relational Database Management System). To infer activities, we define temporal and geo-spatial relationships between the database entities (tracklets, roads, etc.), and query the database. The relational model allows us to represent hierarchical structures, multiple actor activities, and context information. We use SQL (Structured Query Language) to define and infer activities. Table 3 shows typical geo-spatial activities.

Clearly, our system should be able to handle a large number of tracklets since the number of tracklets increases as the volume of 3-D+time region-of-interest increases. For instance, if we collect, process physical attributes (e.g., geo-tag, time-tag), and store tracklets into a RDBMS system from 1 hour long video, we might have 6,000,000 rows in the table.

Inherently, the scalability of our activity recognition system is equivalent to the scalability of traditional relational database systems. The most expensive operation in SQL is the outer join operator (or JOIN for short) whose complexity is $O(N_1 N_2 \cdots N_j \cdots N_r)$ in the worst case, where N_j is the number of data rows (e.g. tracklets) in the j-th table and r is the number of tables [8]. It is a basic operation because we often need to integrate more than one table to utilize the available information from independent data sources and this integration of two relations can be done using the JOIN operator. Most of the activities can be found using the JOIN operators on two relations. Even if some complex queries need more than two relations for the JOIN operations, an efficient algorithm can be applied to reduce the computational complexity of the JOIN operation [8].

2.3.1 Leveraging Scalable Relational Systems

One of the benefits of using an ERM model for activity recognition is that there exist highly optimized RDMBS commercial implementations such as [2]. Furthermore, there has been serious effort in making RDBMS perform equally well in distributed environments, under high load, and with limited downtime. Therefore, by expressing activity definitions in SQL, we can take advantage of existing, distributed, industrial parsers, making our proposed system very scalable.

Some of the ways that RDBMS achieve high efficiency is through the use of indexing and data slicing. We take advantage of both of these strategies.

An index structure organizes each row of tables based on common axises across all data. For example, when a query requires joining of several tables on "id" attribute, an index is built on this attribute in any (temporary) tables taking part in the join. Moreover, many of the properties can be indexed using typical spatial data formats (points, lines) and we can leverage the spatiotemporal index structures [17].

Similarly, when recognizing an activity happening over a relatively short time interval, we first slice the database into multiple time intervals (with overlap), and then query each interval in turn for this activity. This is much faster than querying the whole database at once (Sec. 2.3.3). Clearly, all known query optimization strategies can be adopted here.

Novel relational DBMSs can show excellent scaling properties on distribute systems as long as applications avoid cross-node operations [4]. Indeed, our activity recognition relies on read-only operations so that our system, in theory, can operate on hundreds of machines as a single database system where performance scales linearly with the number of machines.

2.3.2 Distributed computation

It has been shown that simple activities can be inferred by a standard RDBMS and standard techniques such as indexing and SQL optimization. However, using a standard method might not be sufficient when we need fast activity recognition from a large data set or when we infer complex activities that cannot be defined by a small number of JOIN operations. In this case, the strategy for distributed computation depends on the type of activities, because different activities require different spatial, temporal, inter/intra-track dependencies.

Our activities can be classified into "local" spatio-temporal and "global" spatio-temporal activities in term of that whether we have to gather information across distributed tiles or not. A local activity is defined in a bounded spatial and temporal volume. "U-TURN", "2-point-TURN", "Stop-violation", are defined with a "point"

Table 4. Activity type and strategy.

Types	Single actor	Multiple actors
local	indexing	spatial - temporal partitioning
global	track clustering	hierarchical inference

(a single geo-spot) by definition, in which considering a limited area is sufficient. The locality of an activity is important because many local activities can be efficiently inferred using simple partitioning methods.

Similarly, some activities require a "length" or "small area" between tracklets. "Following" is defined between two tracks that maintain a distance. Since these "length" or "area" can be small compared to the processing image tile, only the overlapping regions between tiles should be taken into account and the entire process can be parallelized well like a simple image processing algorithm that uses only local operations.

While a local activity can be inferred by simple spatial and temporal partitioning, in contrast, a global activity might not be inferred from an isolated tile. For instance, "double meeting" can be defined as an interesting activity where two cars meet a location A, move to other locations, and meet again in a new location B. In this case, a simple temporal or spatial division is not possible to infer a global activity. Similarly, "Loop" and "Traveling from A to B" are classified as global activities and may not be inferred from a single tile.

Even in case of global activities, single actor activities can be distributed easily because each track has its own identity. For example, traveling from A to B, a global and single-actor activity can appear across multiple tiles. However, because our tracker connects the identities, we can group long-range tracks, distribute into multiple processors, and infer the global activity in independent clusters.

Many of multiple actor activities, which need more than two relationships in the inference, can be efficiently decomposed into a series of 2-actor activities [5]. For instance, N-vehicle convoy can be decomposed to a series of 2-vehicle convoy and the computation scales linearly with the number of vehicles. However, if we use a distributed database, we might need a more complex strategy, because we cannot use simple grouping and some tracks across tiles might be related.

While considering all possible pairs of tracks requires $C(N, 2)$, we use hierarchical inference for global multiple actor activities. For instance, an activity "double meeting" requires to connect information across tiles. We can divide the query into two steps: the first step collects a set of meeting events in each tile and the second step infers "double meeting" by considering the spatial relationship.

First, we can identify all meetings in each tile as:

$$\begin{aligned} meet(i) := \{(x_1, x_2) \in T_i) | \\ x_1.id \neq x_2.id, \\ ||x_1.pos - x_2.pos|| < \theta_1, \\ |x_1.t - x_2.t| < \theta_2\}, \end{aligned} \tag{2}$$

where (x_1, x_2) is a pair of tracklets in i-th tile (T_i), $x.id$ is the identity number of the tracklet x, $||x_1.pos - x_2.pos||$ represents the Euclidean distance between two tracklets, $|x_1.t - x_2.t|$ is the absolute time difference, and $theta$ is a threshold.

Second, we can connect individual meetings to infer "double meeting" from the result of the first step as:

$$\begin{aligned} N_meetings(i) := \{(x_1, x_2) \in T_i, (x_3, x_4) \in T_j) | \\ x_1.id = x_3.id, x_2.id = x_4.id, \\ T_i \neq T_j\}, \end{aligned} \tag{3}$$

where we just need to find a corresponding pair of meetings $((x_1, x_2), (x_3, x_4))$ from different tiles $(T_i \neq T_j)$.

2.3.3 Quantitative Analysis

Data. We were able to acquire a proprietary GPS trajectory dataset. This dataset was collected over 7 hours, and contains about 50 tracks. These tracks were generated by directing several groups of vehicles to execute various types of activities over the data collection period. We defined three real-world activities. Brushpass is an activity where two cars meet for less than 1 minute and then go their separate ways. Coordinated movement is an activity where two cars move together for a long time. Dead drop is an activity where one car briefly stops at a location, goes away, and then sometime later, a different car goes to the same location, briefly stops there, and moves on.

Method. Tracklets were estimated from GPS trajectories and stored in a database.

To verify the scalability of our proposed system, and examine its behavior with respect to data slicing, we used a "brushpass" activity for the purpose of evaluation. This activity requires the join of two tables, twice, so it is a good candidate for evaluation.

The scalability of our system can be evaluated by measuring query completion time with respect to the number of tracklet rows. We varied the number of rows by temporally dividing the dataset into a various number of intervals $(28, 14, 7, 3, 1)$. Query completion time was then measured in each such time interval. This experiment was performed twice, each time with a different group of vehicles.

To evaluate the recognition performance, we executed a query for each activity, and measured (automatically) the precision and recall using the supplied ground truth. The best performance is obtained for brushpass that shows 0.44 and 0.65 for the precision and recall, respectively.

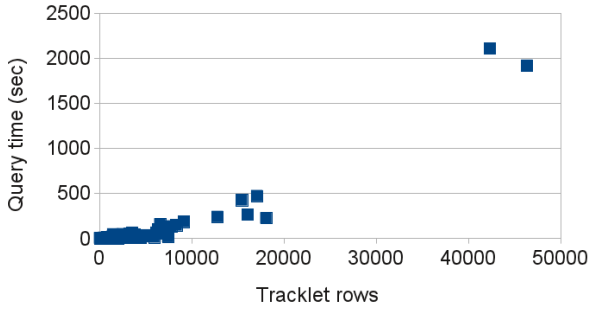

Figure 5. Query completion time rises slowly with the size of the dataset.

Table 5. Data slicing reduces the query completion time.

Intervals	Avg. Query Time (sec)
28	233.25
14	279.95
7	499.02
3	897.65
1	2010.95

Results. The query completion times with intervals are shown in Table 5. It shows that query time increases quadratically with the number of rows in the worst case, but the growth is linear for smaller database sizes. This kind of low-order polynomial growth allows the ERM framework to scale to large database sizes which appear in practice.

To understand the effects of data slicing on query completion time, we use the same data as in the previous experiment. We measure query completion time as the sum of all interval query completion times. These are averaged for the two groups of vehicles. The results are shown in Figure 5. It is clear that data slicing reduces the total query completion time. Therefore, when recognizing activities using ERM, it is more efficient to temporally divide the data into intervals (even with overlap) and run the query within each interval than to run the query on the entire database at once. Note that most of the activities described in Table 3 can be inferred in low order polynomial time with respect to the number of tracklets. Creating the necessary tables and indices takes negligible (amortized) time. Furthermore, this process only happens once, and thus its computational cost is not as important as that of activity inference.

3. Conclusion

We have presented a scalable system to process WAAS imagery. Three key modules have been considered in this paper, tracking, motion pattern estimation, and activity recognition, and we have demonstrated real-time and scalable processing of each. We have transferred the technology to a national lab for deployment. More problems need to be studied, including data compression and 3D modeling.

References

[1] CLIF 2006. https://www.sdms.afrl.af.mil/. 5
[2] http://www.mysql.com. 6
[3] WPAFB-21Oct2009. https://www.sdms.afrl.af.mil/. 3
[4] R. Cattell. Scalable sql and nosql data stores. *SIGMOD Rec.*, 39(4):12–27, 2011. 6
[5] J. Choi, Y. Dumortier, J. Prokaj, and G. Medioni. Activity recognition in wide aerial video surveillance using entity relationship models. In *International Conference on Advances in GIS*, SIGSPATIAL, pages 466–469, 2012. 1, 6, 7
[6] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd edition, 2003. 2
[7] C. S. Mark Keck, Luis Galup. Real-time tracking of low-resolution vehicles for wide-area persistent surveillance. In *Workshop on Applications of Computer Vision*, 2013. 1
[8] P. Mishra and M. H. Eich. Join processing in relational databases. *ACM Comput. Surv.*, 24(1):63–113, 1992. 6
[9] P. Mordohai and G. Medioni. Tensor voting: A perceptual organization approach to computer vision and machine learning. *Morgan and Claypool Publishers*, 2008. 4
[10] P. Mordohai and G. Medioni. Dimensionality estimation, manifold learning and function approximation using tensor voting. *JMLR*, 11:411–450, 2010. 4
[11] A. Perera, C. Srinivas, A. Hoogs, G. Brooksby, and W. Hu. Multi-object tracking through simultaneous long occlusions and split-merge conditions. In *IEEE Conference on CVPR*, volume 1, pages 666–673, 2006. 1
[12] J. Prokaj, M. Duchaineau, and G. Medioni. Inferring tracklets for multi-object tracking. In *IEEE Conference on CVPRW (WAVP)*, pages 37–44, 2011. 1, 2, 4
[13] J. Prokaj and G. Medioni. Accurate efficient mosaicking for wide area aerial surveillance. In *IEEE WACV*, pages 273–280, 2012. 3
[14] J. Prokaj, X. Zhao, and G. Medioni. Tracking many vehicles in wide area aerial surveillance. In *IEEE Conference on CVPRW (WCNWASA)*, pages 37–43, 2012. 2
[15] V. Reilly, H. Idrees, and M. Shah. Detection and tracking of large number of targets in wide area surveillance. In *ECCV*, volume 6313 of *LNCS*, pages 186–199. 2010. 1
[16] R. L. Rothrock and O. E. Drummond. Performance metrics for multiple-sensor multiple-target tracking. In *Proceedings of SPIE*, volume 4048, pages 521–531, 2000. 3
[17] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006. 6
[18] J. Xiao, H. Cheng, H. Sawhney, and F. Han. Vehicle detection and tracking in wide field-of-view aerial video. In *IEEE CVPR*, pages 679 –684, 2010. 1
[19] Q. Yu and G. Medioni. Motion pattern interpretation and detection for tracking moving vehicles in airborne video. In *IEEE CVPR*, pages 2671 –2678, 2009. 4
[20] X. Zhao and G. Medioni. Robust unsupervised motion pattern inference from video and applications. In *IEEE ICCV*, pages 715 –722, 2011. 4