Multiple-Target Tracking With Binary Proximity Sensors

JASPREET SINGH, Samsung Telecommunications America
RAJESH KUMAR, AppFolio Inc.
UPAMANYU MADHOW and SUBHASH SURI, University of California at Santa Barbara
RICHARD CAGLEY, Toyon Research Corporation

Recent work has shown that, despite the minimal information provided by a binary proximity sensor, a network of these sensors can provide remarkably good target tracking performance. In this article, we examine the performance of such a sensor network for tracking multiple targets. We begin with geometric arguments that address the problem of counting the number of distinct targets, given a snapshot of the sensor readings. We provide necessary and sufficient criteria for an accurate target count in a one-dimensional setting, and provide a greedy algorithm that determines the minimum number of targets that is consistent with the sensor readings. While these combinatorial arguments bring out the difficulty of target counting based on sensor readings at a given time, they leave open the possibility of accurate counting and tracking by exploiting the evolution of the sensor readings over time. To this end, we develop a particle filtering algorithm based on a cost function that penalizes changes in velocity. An extensive set of simulations, as well as experiments with passive infrared sensors, are reported. We conclude that, despite the combinatorial complexity of target counting, probabilistic approaches based on fairly generic models of trajectories yield respectable tracking performance.

Categories and Subject Descriptors: I.4.8 [**Image Processing and Computer Vision**]: Scene Analysis—*Tracking, sensor fusion*; G.2 [**Discrete Mathematics**]: Combinatorics—*Counting problems*; G.3 [**Probability and Statistics**]: *Probabilistic algorithms*

General Terms: Algorithms, Theory, Experimentation

Additional Key Words and Phrases: Target tracking, sensor networks, binary sensing, counting resolution, particle filters

ACM Reference Format:
Singh, J., Kumar, R., Madhow, U., Suri, S., and Cagley, R. 2011. Multiple-target tracking with binary proximity sensors. ACM Trans. Sen. Netw. 8, 1, Article 5 (August 2011), 26 pages.
DOI = 10.1145/1993042.1993047 http://doi.acm.org/10.1145/1993042.1993047

5

A preliminary version of this article was presented at the 2007 International Conference on Information Processing in Sensor Networks (IPSN).
This work was supported by the National Science Foundation under grants CCF-0431205, CNS-0520335, CNS-0626954, and CCF-0514738; by the Office of Naval Research under grants N00014-06-1-0066 and N00014-06-M-0260; and by the Institute for Collaborative Biotechnologies under grant DAAD19-03-D-0004 from the U.S. Army Research Office.
This work was performed while J. Singh was with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, and R. Kumar was with the Department of Computer Science, University of California, Santa Barbara.
Authors' addresses: J. Singh, Samsung Telecommunications America, 1301 E. Lookout Dr., Richardson, TX 75082; email: singh.ece@gmail.com; R. Kumar, AppFolio Inc., 55 Castilian Dr., Goleta, CA 93117; email: rajesh.cs@gmail.com; U. Madhow, Department of Electrical and Computer Engineering, University of California, Santa Barbara, Santa Barbara, CA 93106; email: madhow@ece.ucsb.edu; S. Suri, Department of Computer Science, University of California, Santa Barbara, Santa Barbara, CA 93106, email: suri@cs.ucsb.edu; R. Cagley, Toyon Research Corporation, Goleta, CA 93117; email: rcagley@toyon.com.

DOI 10.1145/1993042.1993047 http://doi.acm.org/10.1145/1993042.1993047

1. INTRODUCTION

We investigate the problem of tracking targets using a network of binary proximity sensors. Each sensor produces a single bit of output, which is 1 when one or more targets are in its sensing range and 0 otherwise. These sensors are not able to distinguish individual targets, decide how many distinct targets are in the range, or provide any location-specific information. Despite the minimal information provided by an individual binary sensor, a collaborative network of these sensors has been shown in prior work [Shrivastava et al. 2006, 2009] to yield respectable performance when tracking a single target: the resolution with which the target can be localized is inversely proportional to ρR^{d-1}, where ρ is the sensor density, R is the sensing range, and d is the dimension of the space. In this article, we investigate the problem of multiple target tracking with binary sensors, without a priori knowledge of the number of targets.

We have chosen to focus on the simple and minimalistic setting of binary sensors because the cost and power consumption of sensor nodes is a severe constraint in large-scale deployments, and both can be significantly reduced by restricting the nodes to provide binary output. Thus, by constraining ourselves to a binary sensing model, we can work with low-power, low-cost sensor nodes that can form the basis for a highly scalable architecture for wide-area surveillance. This information can, of course, be augmented by a small number of more capable sensors (e.g., cameras), although we do not explore such enhancements in this article.

Examples of sensor modalities that are suitable for low-cost nodes include [Akyildiz et al. 2002] seismic, acoustic, passive infrared (PIR), active infrared, ultra-wide-band radar imaging, millimeter wave radar, magnetometer, and ultrasonic. For many types of sensors, it is possible to use simple thresholding to get a binary reading or perform onboard signal processing for rough classification. The former option requires drastically reduced processing, and leads to significant power savings. As an example, for acoustic sensing (e.g., the Knowles EA-21842 sensor) and magnetometer sensing (e.g., the Honeywell HMC1002 sensor), the power consumption can be reduced fivefold by using binary mode rather than classification mode. In our lab-scale experiments, we employed PIR sensors due to their good performance, low cost, and ease of systems integration [Moghavvemi and Seng 2004].

As shown in Shrivastava et al. [2009], the binary sensing model is analogous to coarse-grained analog-to-digital conversion that filters out rapid variations in the target's trajectory. This motivates algorithms that attempt to track only "lowpass" versions of the trajectory. For multiple targets, however, we encounter significant additional difficulties, since we cannot tell how many targets are within a sensor's range when it outputs a 1. Our first task in this article, therefore, is to understand how well we can count the number of targets, given a snapshot of the sensor readings. We employ geometric arguments to characterize when an accurate count is possible, and provide a lower bound on the number of targets, based on a greedy algorithm for explaining the sensors' observations with the minimum number of targets. While these arguments bring out the difficulty of target counting and localization based on a snapshot, they do not preclude the possibility of accurate counting and tracking when we account for the evolution of the sensor readings in time, using a model for the targets' behavior. To this end, we develop a particle filtering algorithm which employs a cost function penalizing changes in velocity. It is shown by simulations that the particle filter algorithm is effective in tracking targets even when their trajectories have significant overlap. The algorithm is general enough to incorporate a simple model for nonideal sensing, and provides acceptable tracking performance for our experimental system with PIR sensors even when one of the sensors fails.

For a large part of the article, we restrict our attention to one-dimensional (1D) systems. This enables us to gain fundamental insight, as well as to easily display multiple trajectories on two-dimensional (2D) space-time plots. However, many of our geometric target counting arguments, and also the particle filtering algorithm, generalize to higher dimensions: we comment on the generalization of the geometric target counting arguments to higher dimensions as we go along, while the development of the particle filter algorithm is itself general, and its application to a two-dimensional system is validated through a sample simulation study.

Our focus in this article is on the efficacy of collaborative tracking rather on the communication protocols used by the sensor nodes. Thus we assume that all of the sensor readings are available at a centralized processor, which can then estimate the targets' locations and trajectories. Distributed implementations of our algorithms, in which neighbors collaborate to estimate segments of trajectories, are possible, but are not considered here.

The rest of the article is organized as follows. Section 2 provides a summary of the prior related work. Section 3 discusses the problem of target counting based on a snapshot of the sensor readings. In Section 4, we describe our particle filtering algorithm. Section 5 provides simulation results, while Section 6 describes our experimental setup and results. We end with the conclusions in Section 7.

2. RELATED WORK

The problem of tracking multiple targets using sensor networks has been explored in many prior references [Jung and Sukhatme 2002; Liu et al. 2004; Oh et al. 2005; Reid 1979; Shalom and Li 1979; Shin et al. 2003]. Owing to its simplicity and minimal communication requirements, the specific use of binary proximity sensors for tracking applications has also drawn considerable attention of late. However, most of the work related to binary sensing has been applied to the case of tracking a single target [Aslam et al. 2003; Kim et al. 2005; Shrivastava et al. 2009]. The tracking techniques employed in the large-scale deployment in Arora et al. [2004] can be loosely interpreted in terms of a binary sensing model, even though a variety of sensing modalities and a variety of targets were considered. Oh and Sastry [2005] contained a distributed tracking algorithm for a binary sensor network, but assumed perfect knowledge about the number of targets and their identities, unlike the present work.

In our work, we investigate both target counting and tracking. Prior work on counting targets includes Fang et al. [2002], but it assumed more detailed sensing capabilities than our simple binary model. The classical framework for tracking is based on Kalman filtering, with a linear model for the sensor observations corrupted by Gaussian noise; for example, McErlean and Narayanan [2002] investigated the use of Kalman filtering for distributed tracking. In recent years, the use of particle filters, which can handle more general observation models, has become popular [Arulampalam et al. 2002]. However, most prior work on the use of particle filters for tracking in sensor networks [Coates 2004; Khan et al. 2003, 2005] has assumed a richer sensing model than the binary model we consider. Exceptions are the prior work in Shrivastava et al. [2006, 2009] on the use of particle filters for tracking a *single* target using binary sensing, and also the preliminary results from our conference publication [Singh et al. 2007]. In this article, we build on Singh et al. [2007], providing new analytical design criteria that assist in the efficient and reliable operation of our particle filter algorithm, and present a more detailed simulation-based analysis to evaluate the performance of the algorithm. In addition, we include simulation results and new theoretical proofs for two dimensions (Singh et al. [2007] only considered a one-dimensional setting).

3. SNAPSHOT-BASED INFERENCE: TARGET COUNTABILITY

We begin our investigation by asking under what circumstances an algorithm can reliably determine the number of distinct targets in the field, given a snapshot of the sensor readings. In order to develop fundamental geometric insights, we restrict attention in this section to an idealized model in which each sensor's coverage area is a circular disk of radius R: each sensor detects a target without fail if it falls within this disk, and does not produce false positives or negatives. While we develop our basic ideas and theorems in one dimension, we comment on their relevance and extensions to higher dimensions as appropriate.

3.1. Target Counting with Binary Sensing

Some *spatial separation* among the targets is clearly a necessary precondition for accurately disambiguating among different targets, but what does that mean, and how much separation is enough? For instance, is the following simple condition adequate: *each* target moves sufficiently (arbitrarily) far from the remaining targets at some point during the motion. Let us call this the condition of *individual separation*. Unfortunately, as the following simple result shows, this alone is not enough to count the number of targets accurately.

THEOREM 1. *Even arbitrarily large* individual *separation is not sufficient to reliably count a set of targets using binary sensors.*

PROOF. We give a construction in one dimension establishing the claim. Imagine a group of m targets moving at uniform speed along a straight line L. Initially, all targets are together and appear as one target to the sensor field. Now let target 1 speed up and move away from the rest of the group. Once it moves sufficiently far to the right, we can infer that there are at least two targets. Next, target 1 stops and waits until the rest of the group meets up with it, and then they all resume their motion. Then target 2 separates from the rest of the group and repeats the action of target 1, and so on. One can easily see that, in this scenario, *every* target achieves large individual separation from the rest, and yet no binary sensing-based algorithm can ever decide whether there are two targets or m targets, for an arbitrary value of m. □

On the other hand, if the group of targets has *pairwise* separation more than $4R$, then binary sensing permits precise counting of targets.

THEOREM 2. *Suppose every pair in a set of targets has separation more than* $4R$ *in d-dimensional Euclidean space, where* R *is the sensing range, and suppose that the average sensor density (per unit area) is* ρ*. Then, using binary proximity sensors, we can precisely determine the number of distinct targets as well as localize each target within a spatial error of at most* $\Theta(1/\rho R^{d-1})$.

PROOF. Suppose there are m targets, and let S_i be the set of sensors that sense target i. Because each sensor's range has radius R, by the assumption of pairwise target separation, we must have $S_i \cap S_j = \emptyset$, for any two targets i and j. (This follows because the union of two overlapping ranges has a diameter less than $4R$, while any two targets are assumed to be more than $4R$ apart.) As a result, the "on" sensors are naturally partitioned into m groups, one per target: all sensors in the ith group are on precisely because of one target. Thus the target sensed by the ith group S_i can be localized to the common intersection of all the ranges in S_i and the complement of the ranges of all the "off" sensors. The prior analysis for single-target localization [Shrivastava et al. 2009] has shown that the diameter of this intersection region (which need not be connected) is $\Theta(1/\rho R^{d-1})$. This completes the proof. □

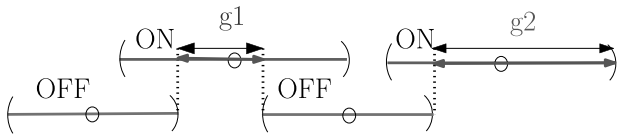

Fig. 1. A sample illustration for the feasible target space (F). Here, $g1$ and $g2$ represent the contributions of the "ON" sensors to F.

In some sense, the preceding example and the theorem settle the "easy" case: when the objects are pairwise far apart, they can be counted as well as localized quite precisely, but individual separation does not help in tracking. We now delve into the more complex (and interesting) situation when these easy conditions do not hold. We point out that there is no *local* fundamental limit based purely on *minimum* separation among targets: two targets no matter how close can always be disambiguated if two sensors with nonoverlapping sensing ranges detect them. At the same time, simply increasing the sensor density to disambiguate nearby targets does not seem possible either. (However, as prior work [Shrivastava et al. 2009] has shown, the "localization" of an individual target does improve linearly with the increasing density.) It seems that we need a more global argument to understand the limit of target counting.

We now focus on one-dimensional space. Much of the difficulty in the binary sensing model has less to do with the dimension of the ambient space and more to do with the "interference" between the influence of different targets on the sensor readings. Any impossibility or hardness results we prove in one dimension naturally hold in higher dimensions as well.

3.2. The Geometry of Target Counting

We begin with some geometric preliminaries. Suppose we have N binary proximity sensors deployed along a line. Each sensor's range is then an interval of length $2R$. We use the notation C_i to denote the interval covered by sensor i (that is, sensor i outputs a 1 if and only if a target falls in C_i). We assume that the domain of interest is covered by the union of the $\{C_i\}$, that is, that there are no gaps in coverage.

Any positioning of targets along the line leads to a vector of binary outputs from the sensors. In particular, we have contiguous groups of "on-sensors" separated by groups of "off-sensors." Geometrically, the on-sensors inform us about the intervals on the line where the targets might be, and the off-sensors tell us about the regions where there are no targets. If we let I be the set of sensors whose binary output is 1 and Z be the set of sensors whose output is 0, then all the targets must lie in the region F, which we call the *feasible target space*:

$$F = \bigcup_{i \in I} C_i - \bigcup_{j \in Z} C_j$$

The region F is a subset of the line, whose connected components are unions of portions of the sensing ranges of the on-sensors. In particular, for sensor i, the portion of its sensing range that appears in F is $g_i = C_i - \bigcup_{j \in Z} C_j$, namely, the part *not* clipped by the off-sensors. An example is shown in Figure 1. The feasible target space is simply the union of these (overlapping) subintervals: $F = \bigcup_{i \in I} g_i$.

The feasible target space has an interesting geometric structure. While each on or off sensor contributes exactly 1 bit, the *information* content of the off sensors seems richer, especially in localizing the targets: the 1 bit only tells us that there is at least one target *somewhere* in the sensor's range, the 0 bit assures us that there is no target

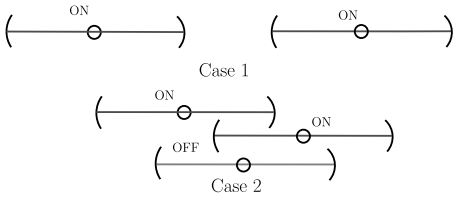

Fig. 2. Positively independent sensors: case 1 shows two sufficiently far apart on sensor; case 2 shows two on sensors separated by an off sensor.

anywhere in the sensor's range. This observation leads to the following geometric property of the region F.

LEMMA 1. *Any two connected components of the feasible target space F are separated by at least distance $2R$.*

PROOF. Choose a point x that is between two connected components of F. Since x must lie in the range of some sensor, and $x \notin F$, that sensor must have binary output 0. A sensor with binary output zero eliminates length $2R$ of the line for possible locations of the targets, and so the "gap" containing the point x must be at least as wide as $2R$. □

3.2.1. Fundamental Counting Resolution. We now use this geometric framework to establish a theorem on the fundamental limit of target counting. Toward that goal, let us define two sensors to be *positively independent* if (i) they both have binary output 1, and (ii) either their sensing ranges are *disjoint* or they belong to different connected components of the feasible target space F. (Note that the independence property is defined with respect to a particular instant, for a given vector of sensor outputs.) In other words, as illustrated in Figure 2, two sensors are positively independent if they are both detecting targets and are either sufficiently far apart (case 1) or are separated by an off sensor (case 2). Then, the following result gives a necessary and sufficient condition for correctly counting the number of targets along a line.

THEOREM 3. *A set of k targets on a line can be counted correctly if and only if there exist k (pairwise)* positively independent *sensors.*

PROOF. We recall that by definition independent sensors have output 1. The "if" part of the claim is therefore immediate: due to their independence, no two sensors can be on because of the same target, and so there must be at least k targets. In order to show the "only if" part, we argue that, if k independent sensors do not exist, then the counting is not guaranteed to be correct. In other words, the sensors cannot distinguish between two scenarios, one with k targets and one with fewer than k targets, thereby violating the correctness.

Without loss of generality, let us number the targets $1, \ldots, k$ in the left-to-right order along the line, and generate a list of sensors $s_1, s_2, \ldots, s_j$ as follows. Let s_1 be the *leftmost* sensor with binary output 1. In general, let s_i be the leftmost sensor with output 1 that is independent of s_{i-1}. Since we have assumed that k independent sensors do not exist, we must have $j < k$. By the pigeon-hole principle, therefore, there must be a sensor among $s_1, s_2, \ldots, s_j$ whose range in F includes at least two targets. We now observe that the binary outputs of none of the sensors will be affected if we translated all the targets to the right until each target was at the *rightmost* point of their independent sensor's range g_i. The sensor with two or more targets clearly has a redundancy, and the binary outputs will not change if one of those targets was

eliminated. It follows that the counting algorithm cannot distinguish between the case of k targets and the $k-1$ targets. This completes the proof. □

3.2.2. Remark on Counting Resolution. By the previous theorem, the number of distinct targets that can be "resolved" at any snapshot of the sensing output equals the number of positively independent sensors. Each such sensor is either distance $2R$ away from its left neighbor (if that neighbor is in the same connected component), or it is preceded by a sensor with binary output 0, which guarantees that no target is present in its coverage area of length $2R$. This can be interpreted "geometrically" to mean that in a space of length $2\ell R$, at most $\ell + 1$ targets can be resolved. Thus, irrespective of sensor density, we can only hope to achieve the counting resolution of about one target per distance $2R$. The payoff of a higher-density deployment comes either in tracking widely separated targets or in resolving two closely spaced targets.

3.3. A Lower Bound on the Target Count

Given the ambiguity in the mapping between sensor readings and target locations, it is of interest to ask what the simplest explanation for a given snapshot of sensor readings is. This Occam's razor viewpoint translates to determining the *minimum* number of targets consistent with the sensor readings. Interestingly, in one dimension, this minimum number matches precisely the maximum number of independent sensors used in Theorem 3.

THEOREM 4. *Given a one-dimensional field of binary proximity sensors, let F be the feasible target space corresponding to their signals at a particular time. Let T be a* minimal *set of targets consistent with F and let S be a* maximum *set of positively independent sensors for F. Then, we must have $|T| = |S|$.*

PROOF. Let $s_1, s_2, \ldots, s_m$ be the sensors with binary output 1, and let $g_1, g_2, \ldots, g_m$ be the intervals they contribute to F. (Recall that g_i is just the range of s_i clipped by the off sensors' ranges.) We can now think of T as the minimum number of points needed to "hit" all the intervals $g_1, g_2, \ldots, g_m$, and S as the maximum number of pairwise nonoverlapping intervals in this collection. That these quantities are equal can be seen by the following simple greedy scheme, illustrated in Figure 3:

> sort the intervals $g_1, g_2, \ldots, g_m$ in the increasing order of their right endpoints; pick the first interval (call it h) in this order and add it to S; delete all intervals that overlap with h; pick the next available interval; and repeat until no more intervals are left.

A simple analysis shows that this greedy scheme finds the maximum possible nonoverlapping intervals in the set, and this correctly returns S. It is also clear that, by putting a target at the right endpoint of each of these intervals, we get the minimum possible set T: since intervals of S are disjoint, we clearly need at least one member in T for each member in S; that this is also sufficient follows because the only intervals not considered are the ones that overlap with members of S at their right endpoints, where the target point is placed. This shows that $|T| = |S|$, and the proof is finished. □

The previous theorem establishes a pleasing fact that a minimal target hypothesis is consistent with the fundamental limit of target countability using binary sensors. The greedy algorithm in the proof of the theorem can also be used to determine a set of target locations that provides a minimal explanation for the readings. The algorithm is highly efficient as well: it requires a single sorting and a scan, so takes $O(n \log n)$ time, if n is the number of sensors.

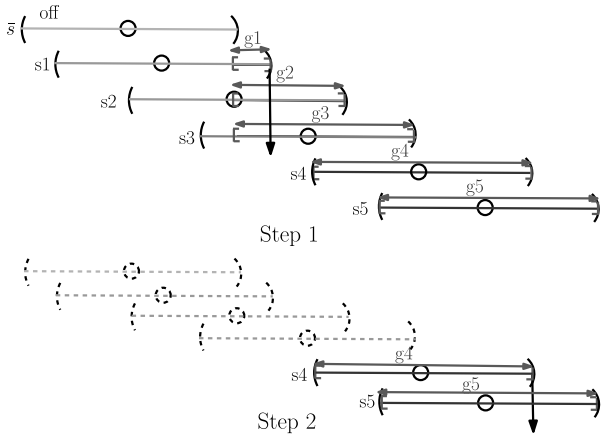

Fig. 3. Illustration of the greedy scheme in Theorem 4. $\bar{s}$ indicates an off sensor, while other sensors are on. The interval h in step 1 is $g1$, while in step 2, it is $g4$.

The ideas of the minimum target set T as well as the maximum independent sensor set S extend naturally to two or more dimensions, although computing them becomes provably intractable (NP-complete). In two or more dimensions, however, they do not always have the same value, although the inequality $|S| \leq |T|$ is always satisfied. That is, the maximum number of independent sensors is a lower bound on the minimum number of targets that are consistent with F. See Appendix A for detailed proofs.

Having analyzed the intrinsic limits of target counting using sensor snapshots, we now move on to the problem of tracking the targets across multiple snapshots. One possible approach to do this is to exploit our preceding geometric results to perform snapshot based inferences at each time instant, and then piece the snapshots together. For instance, we could use our greedy algorithm to determine a set of target locations that provides a minimal explanation for the readings at each time instant, and then merge them across time to obtain a set of possible target trajectories. In a worst-case scenario, where the targets move along arbitrary (adversarial) paths with arbitrarily changing velocities, we cannot hope to do any better than this. However, in a more benign and practical setting where the targets' motion exhibits a certain degree of temporal correlations, we can possibly count and track the targets more accurately by exploiting these correlations. Specifically, instead of taking a greedy minimalistic approach, we can rather work with a potentially large pool of candidate trajectories (obtained by sampling the feasible target spaces), and hope that, on letting these trajectories evolve over time, only a subset of them (which actually correspond to the true target paths) would exhibit the desired temporal correlations. In the following section, we develop a particle filter algorithm that does precisely this, and show through simulations and lab-scale experiments that it is quite effective in tracking multiple targets.

4. TRACKING ACROSS SNAPSHOTS: PARTICLE FILTER ALGORITHM

We consider a centralized model in which a tracker node collects the information gathered by all the sensors over a certain interval of time, and processes the collected data to estimate the trajectories. This centralized architecture may be the most practical option in many settings, given the minimal communication needed to convey the binary sensor readings. However, there are many possible approaches for obtaining

distributed or hierarchical versions of our algorithms, and some of these may be fruitful topics for future work.

Before providing details of the particle filter algorithm, we first provide a formal problem statement. Suppose that there are Q targets, moving in a field of binary proximity sensors. Each sensor reports its 1-bit reading, regarding the presence or absence of targets within its range, at the discrete set of time instants $t \in \{1, 2, \ldots, T\}$. Based on the sensor readings, let the set of feasible target spaces be $\mathcal{F} = \{F[t]\}$, where $F[t]$ denotes the feasible target space at instant t. Denote the location of the qth target at the time instant t by $x_q[t]$, for $q \in \{1, \ldots, Q\}$. The true trajectory of the qth target is given by the set of its locations at the T time instants, that is, $\{x_q[t] : t \in \{1, \ldots, T\}\}$. Given the set $\mathcal{F}$, and without any prior information about the number of targets Q, we wish to generate estimates of the target trajectories, denoted by $\{y_q[t] : t \in \{1, \ldots, T\}\}$, where $y_q[t]$ is an estimate of the location of the qth target at instant t.

The particle filter approach for tracking a single target has been used before in Shrivastava et al. [2009]. We next provide an outline of this approach, discuss its limitations in the setting of multiple targets, and then present a modified version tailored to the multiple-targets problem.

4.1. Tracking A Single Target

The particle filter algorithm for a single target ($Q = 1$ known beforehand) works as follows. We begin at $t = 1$, and proceed step by step to $t = T$, while maintaining a (large) set of K candidate trajectories (or particles) at each instant. Each of the K particles at an intermediate time t is a candidate for the estimated trajectory till time t, that is, a candidate for $\{y_1[t'] : t' \in \{1, \ldots, t\}\}$. Let us denote the kth particle at time t by $\mathbf{P}_k^t$, for $1 \leq k \leq K$. For each k, $\mathbf{P}_k^t$ is a vector of length t, and let it be specified by the set of locations $(\hat{x}_k[1], \cdots, \hat{x}_k[t])$. For instance, at $t = 3$, each particle would be a vector of length 3, and would be a candidate for an estimate of the true target path for the first three time instants. The algorithm is initialized by picking K points in a uniform manner from the set $F[1]$ to get the set of K particles at the first time instant $\{\mathbf{P}_k^1\}$. Each of these is extended to $t = 2$ by picking a point randomly (in a uniform manner) from the set $F[2]$. This generates the set $\{\mathbf{P}_k^2\}$. Now, given K particles at time $t \geq 2$, the K particles at time $t+1$ are obtained in the following manner. Each of the particles $\mathbf{P}_k^t$ is extended to time $t + 1$ by choosing $m > 1$ candidates for $\hat{x}_k[t + 1]$, using uniform sampling over the feasible set $F[t + 1]$. This produces a total of mK particles, each a vector of length $(t+1)$. Based on a *cost function*, the K lowest cost particles out of these mK particles are picked and designated as the K surviving particles at time $(t + 1)$. The algorithm proceeds in a similar manner till the last instant T, and, that time, the particle with the smallest cost function out of the K particles $\{\mathbf{P}_k^T\}$ is picked and designated as $\{y_1[t] : t \in \{1, \ldots, T\}\}$, that is, it is our estimate of the true target trajectory. The basic premise underlying the particle filtering approach is that, if the choice of the cost function is in accordance with the actual motion of the target, then the particles that do not conform to this motion will eventually drop out due to large cost functions, while the surviving particles will be good estimates of the true trajectory.

While we pick the lowest cost particle at the last time instant as our estimate of the true path, if we look at the entire set of the K surviving particles $\{\mathbf{P}_k^T\}$, it is very likely that a significant fraction of them would not differ appreciably from the best one. In other words, we would expect to see a *cluster* of good particles around the best one. This observation is crucial as we now consider multiple targets, since the clustering of particles enables us to distinguish between, and track, multiple targets. Specifically, if the paths taken by the different targets are (reasonably) separable over time, then we would expect that the K surviving particles at the last time instant would comprise

of distinct clusters of particles, with each cluster corresponding to one of the actual targets. This leads to the intuition that the particle filtering approach can be employed to track multiple targets by looking for clusters of particles among the survivors at the last time instant, instead of choosing a single best particle. Unfortunately, this naive extension of single-target tracking does not serve our purpose completely: while in general we expect clusters of particles in the vicinity of each of the target paths, in certain situations, we may end up getting clusters around only a subset of the target paths. As a simple example, consider an instance when one of the targets (say q_1) is far from the others, and moves in a manner that is much more amenable to the cost function than the rest of the targets. Since the particle filter algorithm retains the K best particles, it is quite possible that all of these "lock onto" the trajectory of target q_1, discarding particles corresponding to other targets. A brute force approach to tackle this problem of *monopolization* would be to increase the number of particles that we store (i.e., increase K), but the number of particles needed to make this work, and the associated computational complexity, can be excessive. Instead, as described in the next section, we propose an algorithm in which we identify cluster formation as we go, and limit the number of particles allocated to each cluster.

4.2. Tracking Multiple Targets: The CLUSTERTRACK Algorithm

We call our proposed scheme CLUSTERTRACK. The method is specifically designed to prevent a subset of targets from monopolizing all of the available particles. To this end, instead of looking for clusters at the end, we monitor their formation throughout the tracking process, and limit the number of particles per cluster. We still retain K particles at each time instant. However, instead of picking the K best particles, we pick the K best particles subject to the constraint that the number of particles per cluster does not exceed a threshold H. A cluster is defined as a group of particles that are "similar," where similarity between two particles is measured in terms of a distance metric to be specified. Thus we scan the set of particles in increasing order of cost functions as before, but we retain a particle only if the number of similar particles retained thus far is less than the threshold H. This procedure enhances the likelihood that the particle filter catches all of the targets. In order to ensure that we do not end up scanning the entire sequence of particles at each instant, we can also put a limit L ($L > K$) on the number of particles that we consider. In this case, we stop the search for particles when either K particles have been retained, or L of them have been scanned, whichever happens first. The actual number of particles retained at time t is denoted by K_t, where $K_t \leq K$.

At the final time instant, we take the best particle from each of the clusters obtained, and designate it as our estimate of the trajectory followed by one of the targets. An alternative would be to choose a "consensus path" (e.g., based on a median filter at each time instant) for each cluster.

The pseudo code description for the CLUSTERTRACK at a particular time instant t is given in Algorithm 1. Cluster$_j$ represents the jth cluster, count$_j$ denotes the number of particles retained in Cluster$_j$, N_C is the number of clusters, H is the maximum number of particles to be retained from a particular cluster, and L is the maximum number of particles to be inspected in order to find the surviving particles at time t.

4.2.1. Sampling Strategy and Cost Function. Step 2 of the algorithm requires us to extend each of the surviving particles to the current time instant t, while step 3 requires assigning a cost to each particle. The first task is performed by picking samples uniformly from $F[t]$, with a fixed sampling density of m_o samples per unit area. Note that the total number of samples picked is thus a function of time (it depends on the size of the set $F[t]$), and is hence labeled as m_t.

Algorithm 1: CLUSTERTRACK ($\mathcal{F}$) at time (t)

1: Get the set $\{\mathbf{P}_k^{t-1}\}$ of K_{t-1} surviving particles from time $t-1$.
2: Extend this set to time t, generating a total of $m_t K_{t-1}$ particles.
3: Sort the $m_t K_{t-1}$ particles in ascending order of cost to get the set $\{\hat{\mathbf{P}}_1, \ldots, \hat{\mathbf{P}}_{m_t K_{t-1}}\}$
4: Put $\hat{\mathbf{P}}_1$ in Cluster$_1$, $\mathbf{P}_1^t = \hat{\mathbf{P}}_1$, $N_C = 1$, Count$_1 = 1$, $i = 2, k = 1$
5: **while** ($i \leq L$ and $k \leq K$) **do**
6: **if** ($\hat{\mathbf{P}}_i \in$ Cluster$_j$ for some j) **then**
7: **if** Count$_j < H$ **then**
8: Count$_j \leftarrow$ Count$_j + 1$, $k \leftarrow k + 1$
9: Retain $\hat{\mathbf{P}}_i$ and $\mathbf{P}_k^t = \hat{\mathbf{P}}_i$
10: **else**
11: Abandon $\hat{\mathbf{P}}_i$
12: **end if**
13: **else**
14: Make new cluster for $\hat{\mathbf{P}}_i$, $N_C \leftarrow N_C + 1$, $k \leftarrow k + 1$
15: $\mathbf{P}_k^t = \hat{\mathbf{P}}_i$
16: **end if**
17: $i \leftarrow i + 1$
18: **end while**

We work under the assumption of smooth target trajectories (i.e., the targets do not have abrupt velocity changes), and hence pick a cost function that penalizes changes in velocity. Let $\mathbf{P} = (\hat{x}[1], \ldots, \hat{x}[t])$ denote a particle. The instantaneous estimate of this particle's velocity vector at any time $n \in [1, t-1]$ is the increment in position $\hat{x}[n+1] - \hat{x}[n]$. The instantaneous contribution to the cost, in going from time n to $n+1$, is taken to be the norm of the change in velocity

$$\begin{aligned} c[n] &= ||(\hat{x}[n+1] - \hat{x}[n]) - (\hat{x}[n] - \hat{x}[n-1])|| \\ &= ||\hat{x}[n+1] + \hat{x}[n-1] - 2\hat{x}[n]|| \end{aligned}$$

where $||\cdot||$ denotes Euclidean norm. Assuming that rapid accelerations are unlikely in smooth paths, the cost $c[n]$ should be inversely related to the probability that a target moves from the location $\hat{x}[n]$ at time n to $\hat{x}[n+1]$ at time $(n+1)$, given that it had moved from $\hat{x}[n-1]$ to $\hat{x}[n]$ between time instants $(n-1)$ and n. The net cost function associated with the particle $\mathbf{P}$ is simply taken to be the sum of the incremental costs: $\sum_{n=2}^{t-1} c[n]$.

4.2.2. Choice of Algorithm Parameters and Clustering Criterion. The performance of CLUSTERTRACK depends on the criterion we adopt to cluster different particles, and also on the choice of the following tunable parameters: m_o (the sampling density); K, L and H (the maximum number of particles retained at any time, the maximum number of particles scanned at any time, and the maximum number of particles retained in a cluster at any time, respectively). We focus first on the choice of parameters.

Choice of the parameters. If the motion of the targets is in accordance with the choice of our cost function, increasing the value of m_o, and/or H (with corresponding increase in K and L) is expected to provide improved performance in terms of generating lower-cost estimated trajectories, since it allows us to populate the sample space with a larger number of particles, and/or to retain a larger number of particles at each step. Of course, the level to which we can increase them is governed by the complexity we

can tolerate. We assume that the values of m_o and H are fixed, and try to analyze the values of K and L that we should use for ClusterTrack.

Assuming for the moment that we have an estimate $\hat{Q}$ for the number of targets, it is easy to obtain suitable rules of thumb for K and L. Specifically, if we retain H particles per target, then the total number of particles we retain should be $K \approx H\ \hat{Q}$. In order to get a suitable value for L, we need to answer the following question: what is the maximum number of particles we need to scan to make sure that particles corresponding to *all* targets are caught? To answer this question, consider the H particles corresponding to a particular target (say $q1$) that were retained at the previous time instant. Denoting the size of the feasible target space F at the current time instant by S_F, each of these H particles will be extended to the current time by picking $m_o\ S_F$ samples from F, so as to generate a total of $H\ m_o\ S_F$ particles. However, out of this huge set, we expect that the set of good particles (in terms of the cost function) would be restricted to those for which the current sample is picked in that portion of F which was actually *contributed* by the target $q1$. It is hard to precisely quantify the contribution which a particular target makes to F at any given time instant, since it is governed not only by its own location, but also by the location of the targets in its vicinity. However, an empirical estimate $\hat{A}$ of the average contribution can be obtained from the collected sensor data as follows: $\hat{A} \approx \frac{\sum_1^T S_F(t)}{T\ \hat{Q}}$, where $S_F(t)$ denotes the size of the feasible target space at time t, T is the total number of time instants, and $\hat{Q}$ is the estimated number of targets. Given this empirical estimate, we expect that, on average, the number of good particles corresponding to a particular target would be close to $H\ m_o\ \hat{A}$. Since we need to catch the good particles corresponding to each of the targets, a good design choice for L, therefore is, $(H\ m_o\ \hat{A})\ \hat{Q} = H\ m_o\ \frac{\sum_1^T S_F(t)}{T}$. Note that this does not directly depend on the estimated number of targets, but only on the size of the resulting feasible target space. This makes intuitive sense, since, if we keep adding more targets without changing the feasible space, the total number of smooth particles that we can populate the space with would not change.

Clustering criterion. The decision in step 6 of the algorithm (whether the particle under consideration, $\hat{\mathbf{P}}_i$, belongs to any of the existing clusters) is made as follows. For each of the N_C existing clusters, denote by $\mathbf{CH}_j$ the first particle that joined the jth cluster, where $j \in \{1, \cdots, N_C\}$. We refer to this first particle $\mathbf{CH}_j$ as the *cluster-head* of the jth cluster. For time instant t, both $\hat{\mathbf{P}}_i$ and $\mathbf{CH}_j$ are vectors of length t. Define the *distance* between them to be the mean of the absolute differences between their components, that is, $D(\hat{\mathbf{P}}_i, \mathbf{CH}_j) = \frac{1}{t}\sum_{l=1}^{t} |\hat{\mathbf{P}}_i[l] - \mathbf{CH}_j[l]|$. Compute $D(\hat{\mathbf{P}}_i, \mathbf{CH}_j)$ for each j, and compare the minimum of these distances against a threshold D_0. If the minimum is smaller than the threshold, conclude that the particle $\hat{\mathbf{P}}_i$ belongs to that cluster whose cluster-head has the minimum distance from it. Otherwise, conclude that the particle does not belong to any of the existing clusters. To pick the threshold, we need to answer the following question: for what maximum mean separation between two distinct particles should we consider them to correspond to the same true trajectory? From the preceding discussion, we already have an empirical estimate $\hat{A}$ for the average contribution that a target makes to the area of the feasible space. Thus a suitable choice for the threshold D_o is $(\hat{A})^{\frac{1}{n}}$ (where n denotes the dimensionality of the space), since we can expect that two particles that correspond to the same target would have maximum average separation close to $(\hat{A})^{\frac{1}{n}}$.

It remains to specify an estimator for the number of targets $\hat{Q}$. In one dimension, as discussed in Section 3.3, we can obtain a lower bound on the target count at each

time instant. The maximum of these lower bounds provides a good lower bound on $\hat{Q}$, which would actually be tight if the targets separate out just enough even at only one time instant. In our simulations, we investigated the performance of the algorithm for different choices of $\hat{Q}$ greater than the maximum lower bound. For two or more dimensions, computation of the bounds on the target count becomes intractable. In such a scenario, a simple lower bound on the target count is the maximum number (over all time instants) of disjoint regions that form the feasible space. This lower bound would be tight if the targets were separated widely enough even at only one time instant.

Finally, note that in spite of using the suggested analytical rules for the various parameters, there is no theoretical guarantee that the CLUSTERTRACK algorithm will catch all targets (even if our estimate $\hat{Q}$ is correct). However, once the algorithm has been run, we can perform a simple test to check whether the generated trajectories at least satisfy the instantaneous lower bounds on the target count. As described in detail in Appendix B, if we find that the number of trajectories is smaller than the lower bound, then we run the algorithm again to generate additional trajectories.

Next, we present simulation results to evaluate the performance of our tracking algorithm.

5. SIMULATION RESULTS

Most of our simulations were for a one-dimensional system with sensors placed uniformly along a line (see Section 5.3 for 2D simulations). We denote the sensor radius by R, the sensor density by ρ, the sampling density by m_o, and the location of the qth target at instant t by $x_q(t)$. Further, let X denote the collection of the true target locations, for all the targets, over all the time instants.

The geometry of our sensing model is best revealed by expressing our results in terms of scale-invariant parameters. Note first that a system with parameters (R, ρ, m_o, X) has the same performance as that of a system with parameters $(\frac{R}{\alpha}, \rho\alpha, m_o\alpha, \frac{X}{\alpha})$, for any positive scale factor α, except that all trajectories also scale by $\frac{1}{\alpha}$. Consequently, we analyze and report our results as a function of the following normalized parameters: $(\rho R, m_o R, \frac{X}{R})$. Note that ρR is the number of sensors per unit radius, $m_o R$ is the number of samples we pick per unit radius, and $\frac{X}{R}$ denotes the positions of the targets in units of R.

We begin with an ideal sensing model, wherein each sensor detects the targets within its range without any misses. We then show that our algorithm also works well for a simple model for nonideal sensing.

5.1. Tracking with Ideal Sensing

We considered five targets, and generated trajectories over 20 time instants for each of them. In keeping with our assumption of smooth target trajectories (i.e., no abrupt velocity changes), we picked the velocity of a particular target, at each instant, randomly within $\pm$ 20 % of some mean value (using a uniform distribution). The model applies, for instance, if we consider the motion of vehicles on a freeway, over a reasonably short time window. The parameter ρR was taken to be 1 (i.e., the separation between consecutive sensors was equal to the sensing radius, so that the coverage areas for two adjacent sensors had 50% overlap). For the CLUSTERTRACK algorithm, we took the maximum number of particles per cluster, H = 25, and the sampling density $m_o R$ = 15. Each of the plots shown ahead is a location v/s time plot. Solid curves (colored red) denote the actual target paths, while dashed and/or dotted curves (colored blue or black) denote the estimated trajectories.

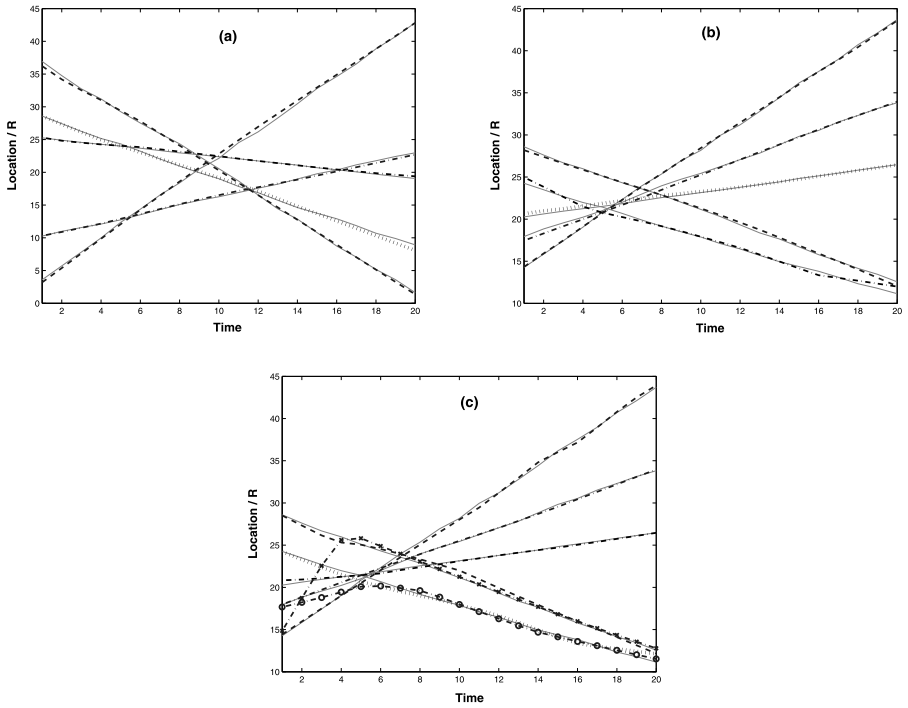

Fig. 4. Example scenarios to depict the performance of CLUSTERTRACK with roughly constant velocity motion, for two different settings (plots (a) and (b)). (Solid curves are used to denote the actual target paths, while dashed and/or dotted curves denote the estimated trajectories). Plot (c) shows the results for the same setting as in plot (b), in a simulation run that resulted in some spurious estimated trajectories as well (marked out by the special characters).

With (roughly) constant velocity motion, as long as the velocities of two targets are not equal, they are guaranteed to separate out at some point of time. We therefore simulated two types of scenarios: (a) targets starting out well separated, getting close to each other, and then separating out again; (b) targets starting in close proximity to begin with, and then separating out. We found that CLUSTERTRACK performed fairly well in both settings. Sample plots are shown in Figures 4(a) and 4(b), each corresponding to a single simulation run. We see that the algorithm succeeded in catching and tracking all targets. We note that the performance of the algorithm varied across simulation runs, and, over multiple runs, the algorithm generated between five and seven trajectories, with five of the trajectories almost invariably providing good approximations of the true paths. For example, the results from a simulation resulting in seven estimated trajectories are shown in Figure 4(c), where the additional spurious estimates are marked by the special characters. Note that we got spurious estimates of both types, low-cost smooth estimates (the estimate marked by $'\circ'$), and also high-cost estimates with sharp transitions (the estimate marked by $'*'$). In general, the emergence of low-cost spurious estimates was governed by the nature of the true trajectories: if the true trajectories allow smooth transitions from one to another, low-cost spurious estimates can arise. On the other hand, the high-cost spurious estimates were seen to emerge only in (a subset of) those cases when the algorithm had to be rerun, because the trajectories generated in the first go could not satisfy the lower bounds on the target count. As explained in Appendix B, this rerun of the algorithm constrains the new trajectories to pass through some

particular connected components of the feasible target space, so that it can result in the generation of rapidly fluctuating trajectories as well. Of course, given their high cost, such spurious estimates are very unlikely to correspond to any actual target.

To obtain the preceding results, we took $\hat{Q}$, the estimate of the number of targets, to be exactly equal to the maximum of the instantaneous lower bounds on the target count. (Remember that we need $\hat{Q}$ in order to decide the values for some of the parameters of the algorithm.) Although this estimate was accurate, we found that, in the first go, the algorithm usually managed to catch four (occasionally three, or, five) of the five targets. The remaining targets were caught when the algorithm was run again in order to satisfy the lower bounds. Given this observation, it is worthwhile to evaluate the performance of our algorithm for larger values of $\hat{Q}$. This increases the likelihood that the algorithm catches all the targets in one go, albeit at the cost of generating additional spurious trajectories. For the same settings as in Figures 4(a) and 4(b), we tested the performance with $\hat{Q}$ as high as twice the maximum of the instantaneous lower bounds. We found that the algorithm did increasingly manage to catch all the targets in one go, and there was no significant increase in the number of spurious estimates.

We also note that, if the overlap between the trajectories is significantly increased, the algorithm can fail to catch some of the targets. An example of such a setting is provided later, when we consider the impact of variations in the system setup on the performance of the algorithm.

For the preceding scenario (relatively constant target velocities), the low-cost spurious estimates formed by joining pieces of the true paths are less likely to emerge as compared to the estimates that actually correspond to the true paths. This is simply because transitioning from one constant velocity path to another requires a change in velocity (resulting in a higher cost function), as opposed to continuing on one particular path. As a result, low-cost spurious estimates emerge quite rarely. However, if we now consider a model that allows for some velocity variations to begin with, while still working with a cost function that penalizes changes in velocity, the likelihood of getting spurious estimates increases. Specifically, if the true paths are such that the estimates obtained by merging them actually have smaller cost functions than the estimates corresponding to the true paths, the algorithm is likely to generate the former category of estimates. An example scenario is shown in Figure 5(a), where we consider constant acceleration motion, so that the target velocities vary over time. As is evident in this particular situation, the estimates formed by combining pieces of the two trajectories have smaller cost functions than the true trajectories themselves. While we constructed this example specifically to show how our decision to penalize velocity variations may not be the most appropriate choice when the targets do vary their velocities, we note that there can be many other nonconstant velocity scenarios where it can still provide good performance. Figure 5(b) shows such an instance, with five targets moving with constant accelerations. We found that the algorithm tracked all five targets correctly in almost all the simulation runs, while generating two to three spurious estimates in some of the runs.

Remark. In the simulation results we have presented, all the targets did separate out at least for one time instant, so that our lower bound on the estimate of the target count $\hat{Q}$ is tight. We also tried to simulate scenarios in which this was not the case, so that the lower bound was not tight. While we managed to find example instances in which the algorithm still was able to catch all the targets (this required us to use a $\hat{Q}$ significantly larger than the lower bound), in general, there is no guarantee that the algorithm will succeed. An interesting open problem is to determine conditions under

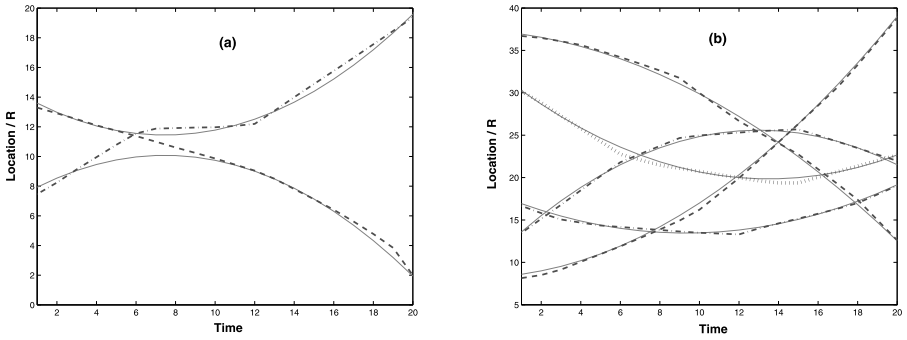

Fig. 5. Example scenarios to depict the performance of CLUSTERTRACK with constant acceleration motion. Plot (a) shows a situation where spurious estimates can have smaller cost functions than the estimates corresponding to the true trajectories, so that the true trajectories may not be generated by the algorithm. However, in less averse scenarios, CLUSTERTRACK still performs very well, as shown in plot (b).

which we can exploit the temporal evolution of trajectories to get an accurate target count, even if all targets do not separate out at any given time instant.

5.1.1. Impact of Variations in the System Parameters. In this subsection, we examine the impact of variations in the system parameters on the performance of CLUSTERTRACK.

Variation in ρR. We first consider the impact of varying ρR, the number of sensors per unit radius, while keeping the target locations $\frac{X}{R}$ fixed. Assuming that all the targets can be identified and tracked correctly, increasing the sensor density should provide improved localization performance. This is because, by putting in additional sensors, the size of each of the connected components comprising the feasible target space F can only decrease. To see this, consider a point x that is not currently in F. It must be in the coverage region of at least one off sensor, say S_x. Even if we put in more sensors, x will still belong to the coverage region of S_x, so that the size of F cannot increase by adding more sensors. On the other hand, a point x that currently belongs to F may end up falling into the coverage region of an off sensor when we add more sensors, so that F may shrink.

To quantify the effect of varying ρR, we considered the same set of target trajectories as in Figure 4(a), and ran CLUSTERTRACK multiple times for each value of ρR. In each simulation run, we took the error in the estimation of a particular target's trajectory to be the mean (over all time instants) error from that estimated trajectory which was nearest to the true trajectory of the target. The average estimation error was then computed by averaging over all targets and all simulation runs. The performance improvement obtained by increasing the density in our simulation setting is depicted in Figure 6(a).

Another minor effect of using a large sensor density that is worth a quick mention is the emergence of spurious estimates such as the ones shown in Figure 6(b). These estimates closely match other true estimates for the most part, but suddenly deviate from them toward the end. An intuitive explanation for their emergence follows from the preceding discussion. For large ρR, the size of the feasible target space becomes small, so that the clustering threshold D_o would also be small. Thus two candidate trajectories that are together for a long time but suddenly have a large separation may not be clustered together. Of course, these spurious estimates have large cost functions, and hence can be eliminated by inspection. Note also that these spurious estimates may also arise at the intermediate stages of the algorithm (rather than just

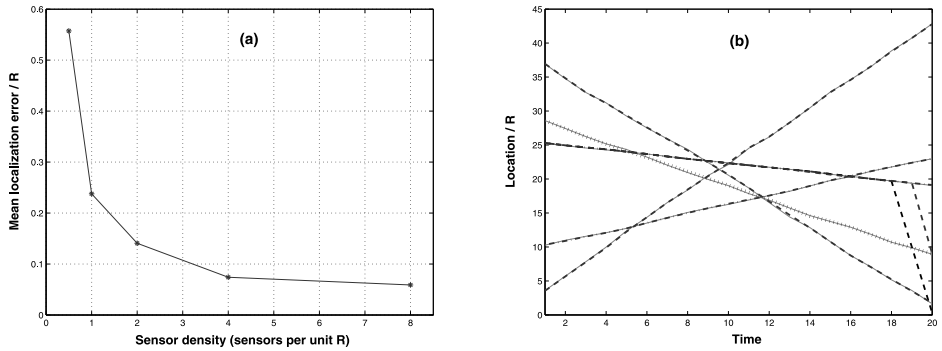

Fig. 6. Impact of change in sensor density. Plot (a) shows the localization error performance of CLUSTER-TRACK versus the sensing density. Plot (b) shows the trajectories obtained in one particular setting, with the sensor density $\rho R = 0.125$, in order to highlight spurious estimates that only deviate from good estimates toward the end.

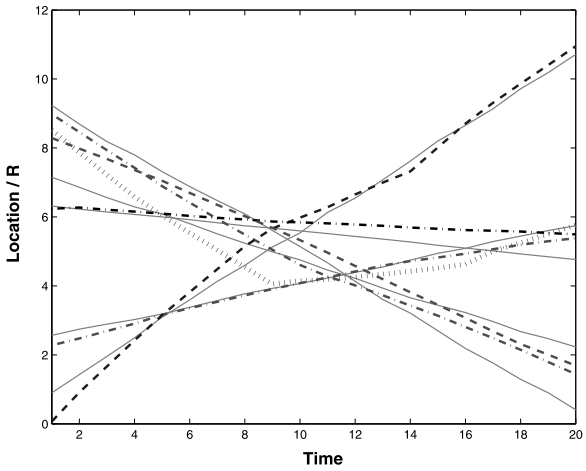

Fig. 7. Impact of changing the target separation on the performance. If the target trajectories have significant overlap, the algorithm may not be able to disambiguate them.

at the end), but due to the large cost function, they would automatically be pruned out when the algorithm proceeds to the next time step.

Variation in $\frac{X}{R}$. We now investigate the impact of variations in the target locations $\frac{X}{R}$, for fixed sensor density ρR. Intuitively, scaling $\frac{X}{R}$ up should improve performance, since it should be easier to resolve widely separated targets, while if we scale down $\frac{X}{R}$ enough, we should become unable at some point to resolve all targets. To illustrate this, we consider the same scenario as in Figure 4(a), but scale down the target locations to bring them closer. Figure 7 shows the performance degradation when we scale down the separation by a factor of 4: the algorithm tracks some of the targets well, but not all. Note however, that, in such scenarios (which can potentially be identified on the basis of no conclusive evidence on the target count and their trajectories over multiple simulation runs), if we let the target trajectories evolve further (hoping that they may separate out later), then the algorithm may recover and eventually identify and track all the targets.

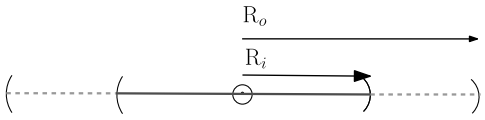

Fig. 8. The two radius nonideal sensing model.

Variation in H and $m_o R$. We also simulated the effect of variation in the two base parameters H (maximum number of particles retained per cluster) and $m_o R$ (sampling density per unit R). Increasing $m_o R$ implies we can sample the feasible space with a greater resolution, while increasing H simply means that the set of trajectories we retain is more dense. Intuitively, we thus expect that increasing either of these parameters would make it possible to find lower-cost trajectories propagating through the feasible space. Indeed, we observed this to be true in our simulations, as the costs of the estimated trajectories we obtained tended to be smaller when we increased $m_o R$ and/or H. Note, however, that this does not necessarily reflect an improvement in terms of the localization error, since there is no apparent correlation between the localization error and the cost function of the estimated trajectories. Our simulation results were inconclusive as well, and did not show a consistent pattern in localization error performance on changing $m_o R$ and H, and we leave further study of the impact of m_o and H as an open issue.

Next we consider tracking with nonideal sensing.

5.2. Tracking with Nonideal Sensing

For real-world deployments with imperfect and noisy sensors, it is necessary to extend the ideal sensing model considered thus far. For instance, a sensor may fail to detect a target within its nominal sensing range, or may sometimes detect targets outside the range. We use a simple model for this nonideal behavior (Figure 8). A target within the inner interval of radius R_i is always detected, and a target outside the outer interval of radius R_o is never detected. The interval between R_i and R_o is a region of uncertainty, and the algorithm that we consider does not require a specific model for the sensor output when the target falls in this region. This is because we use a worst-case interpretation of the model to generate the feasible target space from the sensor data, assuming the maximum uncertainty consistent with the sensor readings. An on-sensor tells us that the target is somewhere inside the outer interval of radius R_o, while an off-sensor indicates that there is no target inside the inner interval of radius R_i. Despite its simplicity, this is a fairly generic model for nonideal behavior, since it arises naturally if sensors integrate noisy samples over a reasonable time scale to make binary decisions regarding target presence or absence.

The setup for simulation is as before: a one-dimensional system with uniformly placed sensors. We continue to express our results in terms of the scale-invariant parameters introduced earlier, with the two additional parameters $\alpha < 1$ and $\beta > 1$ specifying our nonideal sensing model in terms of the ideal sensing radius R: $R_i = \alpha R$, and $R_o = \beta R$. In order to simulate the sensor readings, we assumed that a target falling in the region of uncertainty of a particular sensor is detected with probability 0.5 by that sensor. We evaluated the performance of CLUSTERTRACK for the same scenario as in the simulation with ideal sensing (Figure 4(a)), with $(\alpha, \beta) = (0.7, 1.3)$, and $(\alpha, \beta) = (0.5, 1.5)$. We found that the algorithm was still able to resolve and track all the targets well, although there was a more consistent emergence of spurious estimates as compared to ideal sensing. Figure 9 shows the results for one simulation with $(\alpha, \beta) = (0.5, 1.5)$. We see that the algorithm generated eight trajectories, five of which approximated the true paths well. Out of the three spurious estimates (marked

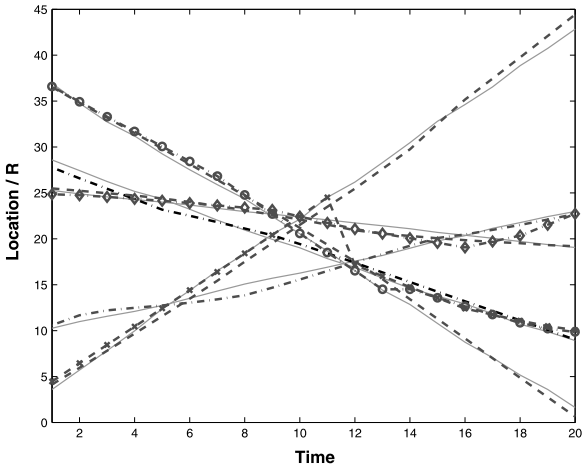

Fig. 9. Performance of CLUSTERTRACK with nonideal sensing, for the same example scenario as considered for ideal sensing in Figure 4(a). The plot shows the results obtained in one simulation run, with $R_i = 0.5R$ and $R_o = 1.5R$.

by the special characters), two had relatively high cost functions and hence may be eliminated by inspection. While the performance in terms of identifying the various targets did not degrade in the face of reasonable levels of nonideality, there was some degradation in the error in localizing the targets. For instance, with $R_i = 0.7\ R$, and $R_o = 1.3\ R$, the mean localization error over multiple simulation runs was found to be $0.287R$, as opposed to $0.238R$ for ideal sensing (the data point in Figure 6(a) for $\rho = 1$). With $R_i = 0.5\ R$, and $R_o = 1.5\ R$, the error increased to $0.377R$.

Our results demonstrate that the particle filter approach was robust for nonideal sensing. In Section 6, we test the performance of our approach on a lab-scale experimental testbed with PIR sensors.

5.3. Two-Dimensional Simulation

Before proceeding to the experimental results, we illustrate the applicability of the particle filter algorithm to a two-dimensional system. Our aim here is to provide a quick simulation to demonstrate that the algorithm extends naturally to two dimensions (note that the development of the algorithm in Section 4.2 is indeed general, and not restricted to just one dimension), leaving more extensive analysis and performance evaluation as a topic for future investigation. To this end, we considered a two-target scenario, and tested the performance of a simple one-run version[1] of our algorithm. The simulation setup was as follows: the sensors were placed on a uniform 2D grid, with the separation between consecutive sensors in either dimension being R units (so that $\rho R^2 = 1$); the maximum number of particles retained per cluster $H = 50$; and the sampling density m_o was such that $m_o R^2 = 50$. (Note that, for a two-dimensional system, the performance is characterized in terms of the normalized parameters: $\{\rho R^2, m_o R^2, \frac{X}{R}\}$, with $X = (X_1, X_2)$ denoting the 2D location of the targets.) Figure 10 shows the results obtained for an example simulation run, wherein the targets started out well separated, approached each other, and then separated out, all the while moving with near constant velocities. The solid (respectively, dashed) curves show the true (respectively, estimated) trajectories of the two targets, while the corresponding

[1]*One-run version* simply means that we stop after the algorithm has been run once, without checking whether all the lower bounds on the target count have been satisfied.

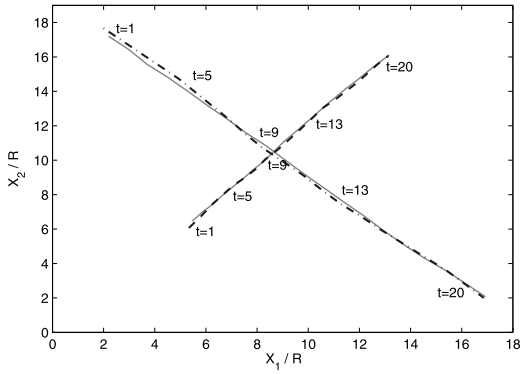

Fig. 10. Example scenario to depict the performance of CLUSTERTRACK in a two-dimensional setting. Solid (respectively, dashed) curves show the true (respectively, estimated) trajectories, and the corresponding time instants are marked out on the curves.

time instants are marked out on the respective curves. We can see that the algorithm succeeded in catching and tracking both targets.

Our sample simulation results show that the particle filter algorithm can be applied to 2D as well. However, there are a number of ways in which 2D is more complicated than 1D. First, note that the algorithm relies on the availability of the feasible target space $\mathcal{F}$. In 1D, it is straightforward to specify $\mathcal{F}$ given the sensor reading: since $\mathcal{F}$ is a union of intervals, we may simply specify the start and end points of each of these intervals. For two (or more) dimensions, $\mathcal{F}$ is a union of multidimensional sets, and it is not immediately evident how best to compute and specify these sets (in particular, closed-form expressions for these sets appear to be elusive). A reasonably accurate approximation for $\mathcal{F}$, which we used for our 2D simulations, is to discretize the 2D space into a (fine) grid, and to test each point in the grid separately as to whether it belongs in $\mathcal{F}$. However, this approach does not specify the different disjoint components comprising $\mathcal{F}$, knowledge of which is needed in order to get a lower bound on the target count, and to check if the algorithm needs to be rerun in case of any unused components. For our 2D simulations, we used a single-run version of the algorithm, and assumed that the maximum number of disjoint components (i.e., a lower bound on the target count) was somehow available.

In short, while the particle filtering approach extends naturally to higher dimensions, the preceding discussion shows that further investigation is needed into efficient mechanisms that ensure that the algorithm catches all targets with high probability.

6. EXPERIMENTS

We used a small testbed with five PIR sensors placed uniformly along a line; see Figure 11. Each sensor sent a measurement to the base station when it changed state, and the base station was interfaced to a PC through a serial port. The data got time stamped at the PC, so that each of the final set of measurements included: value, position (mapped from node ID), and time. For the ground truth regarding target trajectories, the (human) targets were provided with separate sensor nodes (equipped with localization engines) with buttons, which they pressed as they passed by a set of known locations on the way.

While each sensor in our experimental set up sent a measurement when it changed state, our problem formulation in Section 4 is based on the assumption that all sensors send their measurements at regular time instants. To apply our algorithm, therefore, we sampled the collected data at regular time instants, and assumed that the reading

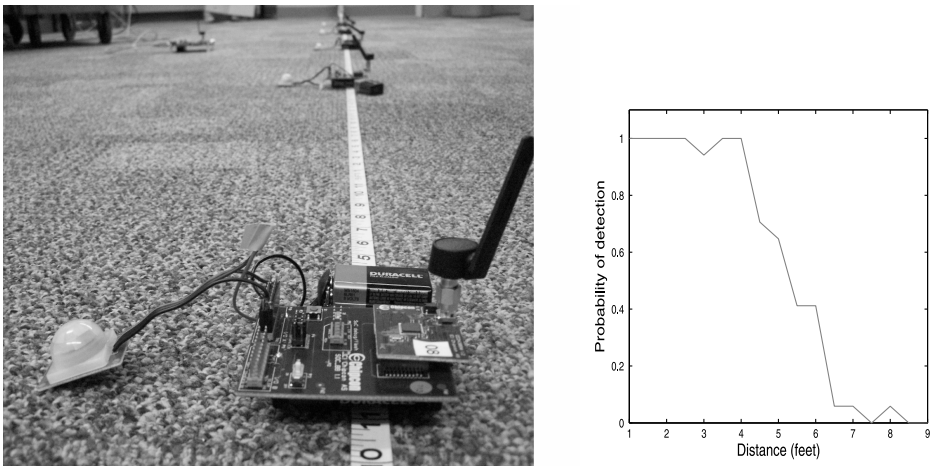

Fig. 11. Experimental setup and sensor characterization: The figure on the left shows the experimental setup, with the sensor modules placed uniformly along a line. The plot on the right shows the probability of target detection versus distance for a particular sensor module.

of a particular sensor at any time was the same as the one after its last toggle. Another implementation issue we faced was that, even when a target was detected as it entered the field of a sensor, the sensor output became 0 immediately after the detection, and kept toggling between 0 and 1 as the target moved toward the sensor. A probable reason for this is that the modules we used are meant for triggering a relay that resets after a certain amount of time, with the aim of minimizing false alarms, at the cost of some missed detections. To deal with this issue, we simply decided to neglect every $1 \rightarrow 0$ transition that was immediately followed by another $0 \rightarrow 1$ transition.

6.1. Sensor Characterization

We first performed some experiments to characterize an individual sensor module. The readings obtained were far from ideal. The probability of target detection with distance is depicted in Figure 11. In order to fit the sensor behavior to our nonideal model of Figure 8, we set $R_i = 3$ ft and $R_o = 7$ ft.

6.2. Tracking Performance

In our experiments, we placed the sensors uniformly along a line, separated by 4 ft (represented by the circles in Figure 12(a)). We considered two targets, which started from opposite ends and crossed each other. The severe nonideal behavior of the sensors was evident as one of the sensors, placed at the location of 16 ft (shown by an asterisk $*$ inside the circle) completely missed the presence of target T_1. Despite the missed detection, we found that, on running the algorithm multiple times, in about 65% of the runs the algorithm succeeded in catching and tracking both the targets reasonably well (with an additional spurious estimate emerging in about every third run). In about 30% of the runs, the algorithm caught one of the targets and generated a spurious estimate. The remaining 5% of runs resulted in two spurious estimates. The performance in one of the better runs is shown in Figure 12(a).

Given the significant variation in performance over different runs, it can be useful to plot the results obtained over multiple runs simultaneously in one figure. Such a figure can be used for a quick visual inference about the number of targets and their trajectories. Figure 12(b) shows these results. We plot the results obtained over 100

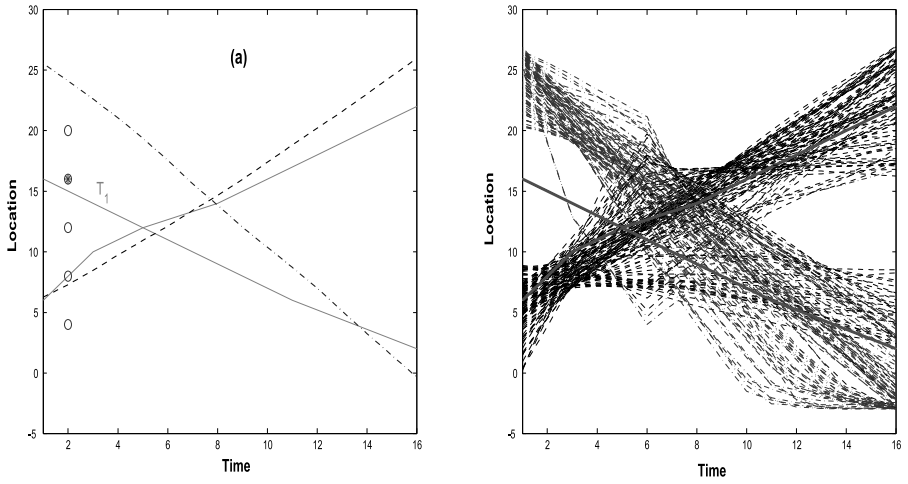

Fig. 12. Performance of CLUSTERTRACK in an experiment in which one sensor completely missed a target. Plot (a) shows the performance in one of the good simulation runs, while plot (b) shows the trajectories obtained over multiple simulation runs.

simulation runs, without including the trajectories which had a significantly high cost function (greater than 10 time the lowest cost trajectory). Based on the plots, we may conclude (with high probability) that there were two targets. In general, a systematic procedure to estimate the number of targets, using the results obtained over multiple simulation runs, could be to perform another clustering operation on the trajectories generated in these multiple runs, and to use the number of clusters thus obtained as an estimate for the target count. This estimate can further be biased based on the number of trajectories obtained in each of these clusters. We leave this as an open issue for further exploration.

The mean localization error for the preceding experimental results was found to be $0.7162R$, with R being $(R_i + R_o)/2 = 5$ ft.[2] While the localization errors for ideal sensing plotted in Figure 6(a) were for a different setting (different number of targets with different trajectories), an error of $0.7162R$ is still significantly larger than the values we obtained there. (The sensing density for our experiments was $\rho R = 1.25$). Clearly, the missed detection of target T_1 by one of the sensors led to this jump in the localization error. A trivial method to capture such severe nonideality (i.e., missed detection), while still working with our two-radius model for nonideal sensing, is to pick $R_i = 0$. However, a small choice of R_i, while guarding against missed detections, also has the obvious drawback that when the off-sensors are actually reliable, they provide us no useful information, leaving us with a large feasible target space. It is an open issue as to how best to trade off these conflicting objectives. One possible approach worth investigation might be to pick a small R_i, but to penalize those particles that traverse the off-sensors' ranges.

7. CONCLUSIONS

The promising results obtained here, as well as prior results in Kim et al. [2005] and Shrivastava et al. [2006, 2009] for the same sensing model, indicate that binary

[2]To compute the mean localization error, we need to know the ground truth at all time instants. The actual ground truth data was available only for a subset of the instants: the targets pressed the buttons only as they passed some known locations, and even some of these messages were not received at the base station. To obtain the ground truth at all times, we simply interpolated the collected ground truth data.

proximity sensors, can form the basis for a robust architecture for wide area surveillance and tracking. Our target counting results show that interesting conclusions can be drawn regarding the number of targets and the feasible target space even without any model for the target paths. On the other hand, when the target paths are smooth and reasonably well separated over time, our CLUSTERTRACK particle filter algorithm gives excellent performance in terms of identifying and tracking different target trajectories.

A host of questions remain to be investigated in future work, of which we provide a partial list as follows. We employ our combinatorial results on target counting in order to obtain rules of thumb for the various parameters in our particle filter algorithm. However, is there a more direct way of combining these two techniques to enhance the performance even further? How broadly does our particle filter algorithm apply, in terms of robustness to different models for the targets' trajectories? When precisely does it break down? How does the tracking performance depend on the dimension of the space we operate in? (Note that we already presented sample simulations to show that the algorithm works in two dimensions; see also Bathula et al. [2009] for an application of the algorithm for tracking a single target in two dimensions.) How can the algorithm be adapted to provision for appearance of new targets, or disappearance of existing targets? Finally, the algorithm currently works in a post hoc manner: a batch of data is first collected, and then processed. While the sampling and cost computations can be performed in real time (since they depend only on data collected up to the present time instant), we do perform a global analysis of the data up front to obtain a good estimate of the target count, which is needed to pick the various parameters. Can this restriction be relaxed? What are its possible implications?

From a sampling theory perspective, it would be interesting to investigate the dependence of the particle filter algorithm's performance on the rate at which the sensors gather the data. Clearly, too low a sampling rate may be insufficient to capture all the required information, as mandated by the frequency content of the different targets' trajectories. On the other hand, too high a sampling rate can reduce the effectiveness of our cost function, which penalizes the instantaneous velocity variations in order to rule out the spurious estimates that jump from one true path to another. A high sampling rate could make the cost of jumping from one path to another less significant, thereby encouraging the emergence of such spurious estimates. This issue can perhaps also be investigated in terms of finding a suitable cost function which can guarantee an improvement in the performance with an increase in the sampling rate.

APPENDIXES

APPENDIX A: PROOFS FOR GEOMETRIC ARGUMENTS IN TWO DIMENSIONS

PROOF. Computing the minimum set of targets T consistent with F in two dimensions is at least as hard as the hitting set problem for unit-radius disks, which has the following formulation:

Given a set D of n unit-radius disks in the plane, and an integer k, does there exist a set H of k points (chosen anywhere in the plane) that intersect all the disks? In other words, each disk of D hits at least one point of H.

This problem is known to be NP-complete [Fowler et al. 1981]. To show that computing T is also hard, we can reduce the hitting set problem to our problem, as follows. Given an instance D of the hitting set problem, we identify each disk of D with the sensing range of a sensor. All of these sensors are set to be on, and there are no off-sensors. Then the feasible space target space F is simply the union of these disks, and the smallest number of point targets consistent with F is k if only if the set of disks can be hit with k points. Therefore, computing T is NP-complete.

Similarly, the maximum number of positively independent sensors S can be shown to be NP-complete using a reduction from the maximum independent set problem for unit-disk graphs.

A collection of n unit-radius disks define a unit-disk graph G as follows. Each disk corresponds to a node of G, and there is an edge between two nodes of G if the disks corresponding to those nodes overlap. The maximum independent set of G is the maximum number of nodes in G no two of which are joined by an edge. The maximum independent set problem is known to be NP-complete even for unit-disk graphs, as shown in Clark et al. [1990].

To reduce this problem to the problem of computing S, we take each disk of the unit-disk graph as the sensing range of a sensor, and set all sensors to be on. Then the maximum number of independent nodes in G equals the maximum number of sensors whose ranges are pairwise disjoint. Therefore, computing S is also NP-complete.

Finally, to show that $|S| <= |T|$, it is easy to see that the size of the hitting set $|T|$ has to be at least as large as the number of pairwise disjoint disks, namely, S, because no two disks of S can be hit by the same point. □

APPENDIX B: RERUN OF THE ALGORITHM TO SATISFY THE LOWER BOUNDS

Once the CLUSTERTRACK algorithm has been run, we verify whether the generated trajectories satisfy the lower bounds on the target count, for each of the connected components, over all time instants. In case the lower bound is not satisfied for some component(s) (i.e., there are some underutilized components), we run the algorithm again to generate new trajectories, over and above the ones already obtained, and keep repeating the procedure until all the lower bounds are satisfied. Each time this rerun is performed, we constrain the algorithm to ensure that the new trajectories traverse one of the underutilized components, as explained next.

Let $\{1, \ldots, T\}$ be the set of time instants. We scan the different time instants to see if there are any underutilized components. Let t_o be a time instant at which we find an underutilized component, say c_o. We now rerun the algorithm, with the feasible target space being the original space for all time instants, except t_o, for which the feasible space consists solely of the component c_o. The estimate for $\hat{Q}$, the number of new trajectories we are looking to generate, is taken to be difference between the lower bound for the component c_o, and the actual number of trajectories that already pass through it. Once new trajectories have been generated, we scan the different time instants again to see if there are still any underutilized components, and the process is repeated till all the lower bounds are satisfied.

Two final points that need a mention: first, the scan to look for underutilized components is done in the following order of time instants: $\{1, T, 2, T-1, 3, T-2, \ldots\}$ (rather than as $\{1, 2, 3, \ldots, T\}$), and, if the time $t_o > \frac{T}{2}$, then before rerunning the algorithm, the time order of the collected data is inverted. This is just done to enhance the likelihood that the new trajectories we get are smooth. For instance, assume that the only unused component c_o is at time $t_o = T$. In this case, if we run the algorithm as usual, progressing from time 1 to time T, till time $T-1$, we would not be accounting for the fact that the unused component is at time T. When progressing from time $T-1$ to T, we would suddenly force the trajectories to pass through c_o, which can lead to a sudden fluctuation. Rather, if we begin at time T, and proceed backward, we are more likely to obtain a smooth estimate. Second, a new trajectory generated by the rerun of the algorithm may actually be similar to one of the trajectories we already have, in which case it is no use retaining it. Hence, we perform this check once a new trajectory has been obtained, and retain it only if it can not be clustered with any of the already obtained trajectories. The clustering criterion is exactly the same as the one employed

during the course of running the algorithm. If however, we exceed a certain number of reruns (10 for our simulations) in this process, then this check is not imposed, and all the new trajectories generated from that point onward are retained.

ACKNOWLEDGMENTS

The authors thank Raghuraman Mudumbai (now at University of Iowa) and Sriram Venkateswaran (University of California Santa Barbara) for several useful discussions. They also thank the anonymous reviewers for their helpful and constructive comments.

REFERENCES

AKYILDIZ, I., SU, W., SANKARASUBRAMANIAM, Y., AND CAYIRCI, E. 2002. A survey on sensor networks. *IEEE Comm. Mag. 40*, 102–114.

ARORA, A., DUTTA, P., BAPAT, S., KULATHUMANI, V., ZHANG, H., NAIK, V., MITTAL, V., CAO, H., DEMIRBAS, M., GOUDA, M., CHOI, Y., HERMAN, T., KULKARNI, S., ARUMUGAM, U., NESTERENKO, M., VORA, A., AND MIYASHITA, M. 2004. A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Int. J. Comp. Telecom. Netw. 46*, 605–634.

ARULAMPALAM, M. S., MASKELL, S., GORDON, N., AND CLAPP, T. 2002. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Trans. Signal Process. 50*, 174–188.

ASLAM, J., BUTLER, Z., CONSTANTIN, F., CRESPI, V., CYBENKO, G., AND RUS, D. 2003. Tracking a moving object with a binary sensor network. In *Proceedings of the ACM SIGOPS International Conference on Embedded Networked Sensor Systems*.

BATHULA, M., RAMEZANALIAND, M., PRADHAN, I., PATEL, N., GOTSCHALL, J., AND SRIDHAR, N. 2009. A sensor network system for measuring traffic in short-term construction work zones. In *Proceedings of the IEEE International Conference on Distributed Computing in Sensor Systems*. 216–230.

CLARK, B. N., COLBOURN, C. J., AND JOHNSON, D. S. 1990. Unit disk graphs. *Discrete Math. 86*, 165–177.

COATES, M. 2004. Distributed particle filters for sensor networks. In *Proceedings of the International Conference on Information Processing in Sensor Networks*. 99–107.

FANG, Q., ZHAO, F., AND GUIBAS, L. 2002. Counting targets: Building and managing aggregates in wireless sensor networks. Tech. rep. Palo Alto Research Center, Palo Alto, CA.

FOWLER, R., PATERSON, M., AND TANIMOTO, S. 1981. Optimal packing and covering in the plane are np-complete. *Inf. Proc. Lett. 12*, 133–137.

JUNG, B. AND SUKHATME, G. S. 2002. Tracking targets using multiple robots: The effect of environment occlusion. *Auton. Robots 13*, 3, 191–205.

KHAN, Z., BALCH, T., AND DELLAERT, F. 2003. Efficient particle filter-based tracking of multiple interacting targets using an MRF-based motion model. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*.

KHAN, Z., BALCH, T., AND DELLAERT, F. 2005. MCMC-based particle filtering for tracking a variable number of interacting targets. *IEEE Trans. Patt. Anal. Mach. Intell. 27*, 11, 1805–1819.

KIM, W., MECHITOV, K., CHOI, J., AND HAM, S. 2005. On target tracking with binary proximity sensors. In *Proceedings of the International Conference on Information Processing in Sensor Networks*.

LIU, J., CHU, M., LIU, J., REICH, J., AND ZHAO, F. 2004. Distributed state representation for tracking problems in sensor networks. In *Proceedings of the International Conference on Information Processing in Sensor Networks*.

MCERLEAN, D. AND NARAYANAN, S. 2002. Distributed detection and tracking in sensor networks. In *Proceedings of the 36th Asilomar Conference on Signals, Systems and Computers*. Vol. 2. 1174–1178.

MOGHAVVEMI, M. AND SENG, L. C. 2004. Pyroelectric infrared sensor for intruder detection. In *Proceedings of the IEEE TENCON Conference*. 656–659.

OH, S. AND SASTRY, S. 2005. Tracking on a graph. In *Proceedings of the International Conference on Information Processing in Sensor Networks*.

OH, S., HWANG, I., ROY, K., AND SASTRY, S. 2005. A fully automated distributed multiple-target tracking and identity management algorithm. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*.

OH, S., SCHENATO, L., AND SASTRY, S. 2005. A hierarchical multiple-target tracking algorithm for sensor networks. In *Proceedings of the International Conference on Robotics and Automation*.

REID, D. 1979. AAN algorithm for tracking multiple targets. *IEEE Trans. Aut. Contr. 24*, 843–854.

SHALOM, Y. B. AND LI, X. R. 1979. *Multisensor, Multitarget Tracking: Principles and Techniques*. YBS Publishing, Storrs, CT.

SHIN, J., GUIBAS, L., AND ZHAO, F. 2003. A distributed algorithm for managing multi-target identities in wireless ad-hoc sensor networks. In *Proceedings of the International Conference on Information Processing in Sensor Networks*.

SHRIVASTAVA, N., MUDUMBAI, R., MADHOW, U., AND SURI, S. 2006. Target tracking with binary proximity sensors: Fundamental limits, minimal descriptions, and algorithms. In *Proceedings of the ACM SIGOPS International Conference on Embedded Networked Sensor Systems*.

SHRIVASTAVA, N., MUDUMBAI, R., MADHOW, U., AND SURI, S. 2009. Target tracking with binary proximity sensors. *ACM Trans. Sens. Netw. 5*, 4, 1–33.

SINGH, J., MADHOW, U., KUMAR, R., SURI, S., AND CAGLEY, R. 2007. Tracking multiple targets using binary proximity sensors. In *Proceedings of the International Conference on Information Processing in Sensor Networks*.

Received December 2009; revised June 2010; accepted August 2010