Tracking Multiple Targets Using Binary Proximity Sensors *

Jaspreet Singh
Upamanyu Madhow
Electrical and Computer Engineering, University of California, Santa Barbara
CA 93106 USA
{ jsingh , madhow }
@ece.ucsb.edu

Rajesh Kumar
Subhash Suri
Computer Science
University of California
Santa Barbara
CA 93106 USA
{ rajesh , suri }
@cs.ucsb.edu

Richard Cagley
Toyon Research Corporation
Goleta, CA 93117, USA
rcagley@toyon.com

ABSTRACT

Recent work has shown that, despite the minimal information provided by a binary proximity sensor, a network of such sensors can provide remarkably good target tracking performance. In this paper, 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 across 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 for the trajectories yield respectable tracking performance.

Categories and Subject Descriptors

I.4.8 [**Scene Analysis**]: Tracking, Sensor fusion;
G.2 [**Discrete Mathematics**]: Counting Problems;
G.3 [**Probability And Statistics**]: Probabilistic algorithms

Keywords

Target Tracking, Sensor Networks, Binary Sensing, Counting Resolution, Particle Filters

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 a single binary sensor, a collaborative network of binary sensors has been shown in prior work [18] to yield respectable tracking performance: the resolution with which a 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 paper, we extend the work of [18], which considered a single target, and investigate the problem of tracking multiple targets, 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 detection. 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 paper.

Examples of sensor modalities that are suitable for low-cost nodes include [1] 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

*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 US Army Research Office.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
IPSN'07, April 25-27, 2007, Cambridge, Massachusetts, USA.
Copyright 2007 ACM 978-1-59593-638-7/07/0004 ...$5.00.

EA-21842 sensor) and magnetometer sensing (e.g., the Honeywell HMC1002 sensor), the power consumption can be reduced five-fold by using binary mode rather than classification mode. In our lab-scale experiments, we employ PIR sensors due to their good performance, low cost, and ease of systems integration [11].

As shown in [18], 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 paper, 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 sensor's observations with the minimum number of targets. While these arguments bring out the difficulty of target counting 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 non-ideal sensing, and provides acceptable tracking performance for our experimental system with PIR sensors even when one of the sensors fails.

We restrict attention to one-dimensional systems throughout this paper. This enables us to gain fundamental insight, as well as to easily display multiple trajectories on two-dimensional space-time plots. However, both our geometric target counting arguments and the particle filtering algorithm generalize to higher dimensions.

Our focus in this paper 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. We note that the binary sensing model has minimal communication requirements, hence this assumption of centralized processing is quite practical: a sensor need only convey the intervals at which it switches "on" and "off" (assuming that the readings are averaged so as to remain reasonably steady, this is far more efficient than sending a sample of the sensor's readings at regular intervals).

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

Related Work

The problem of tracking multiple targets using sensor networks has been explored by many prior references [15, 16, 13, 5, 19, 9, 17]. 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 [3, 8, 18]. The tracking techniques employed in the large-scale deployment in [2] can be loosely interpreted in terms of a binary sensing model, even though a variety of sensing modalities and a variety of targets are considered. Reference [12] contains a distributed tracking algorithm for a binary sensor network, but assumes 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 [14], but it assumes more detailed sensing capabilities than our simple binary model. The classical framework for tracking is based on Kalman filtering, with Gaussian assumptions for the sensor readings; for example, [10] investigates 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. However, most prior work on the use of particle filters for tracking in sensor networks [4, 6, 7] assumes a richer sensing model than the binary model we consider. An exception is our own prior work in [18] on the use of particle filters for tracking a single target using non-ideal binary sensing. In this paper, we build on these ideas to develop particle filters for tracking multiple targets.

2. TARGET COUNTABILITY

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.

We want to understand and articulate the conditions under which an algorithm can track multiple targets with provable guarantees. A first step for any tracking algorithm must deduce how many distinct targets are present in the field, and so 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. This is a worst-case model which applies, for example, when the rate at which the sensors report their readings is low compared to the rates at which the targets cross the boundaries of the sensors' coverage areas. Put another way, this section addresses the most general scenario, in which we have no model for the targets' trajectories. As we shall see in Section 3, when we do employ a plausible model corresponding to a scenario in which the sensor readings are available at a "high enough" rate, then it is indeed possible to do better than what is promised by the worst-case model considered in this section.

2.1 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 clustered 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 spatial error 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 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 analysis of our single target localization [18] shows that the diameter of this intersection region (which need not be connected) is $\Theta(1/\rho R^{d-1})$. This completes the proof. □

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 *non-overlapping* sensing ranges detect them. At the same time, simply increasing the sensor density to disambiguate closeby targets does not seem possible either. (However, as our earlier work shows 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.

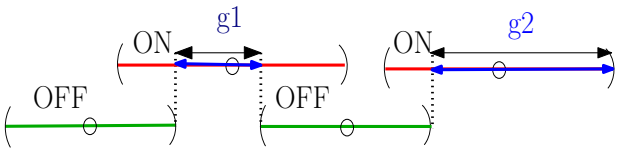

Figure 1: A sample illustration for the feasible target space (F). Here, $g1$ and $g2$ represent the contributions of the 'ON' sensors to F.

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.

2.2 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 $\{C_i\}$, i.e., 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 one 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 *anywhere* in the sensor's range. This observation leads to the following geometric property of the region F.

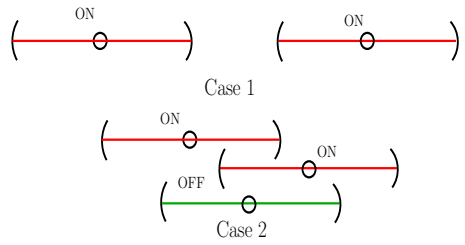

Figure 2: Positively independent sensors: Case 1-sufficiently far apart, Case 2-separated by an off sensor.

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 $2R$. □

Fundamental Counting Resolution

We now use this geometric framework to establish a theorem on the fundamental limit of target counting. Towards 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 2 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. □

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 1 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.

2.3 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 non-overlapping 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 non-overlapping intervals in the set, and this correctly returns S. It is also clear that by putting

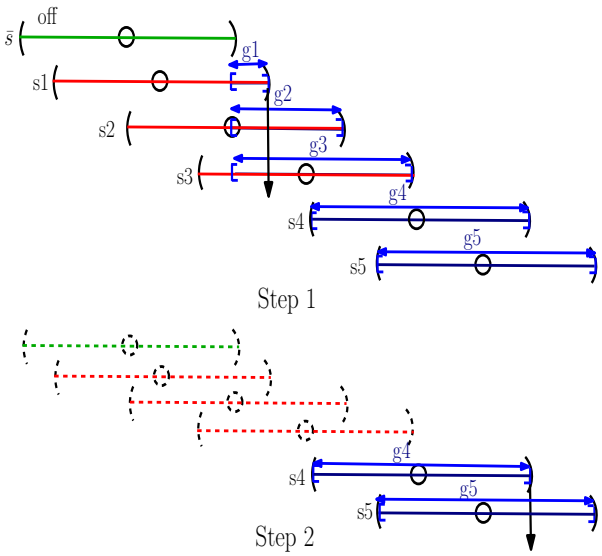

Figure 3: Illustration of the greedy scheme in Theorem 4. $\bar{s}$ indicates an off sensor, while other sensors are on. The interval h in Step 1 is $g1$, while in Step 2, it is $g4$.

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. Moreover, the greedy algorithm in the proof can be used to determine a set of target locations that provides a minimal explanation for the readings, and can be used as a building block for tracking across snapshots. The algorithm is also highly efficient: it requires a single sorting and a scan, so takes $O(n \log n)$ time, if n is the number of intervals. However, we find that probabilistic, model-based, techniques for tracking are more effective in exploiting the continuity of trajectories in time. The latter approach is pursued in Section 3, where we investigate particle filter algorithms for tracking.

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 addition, while both these quantities offer an Occam-like minimalism, in two or more dimensions, they do not always have the same value. In general, however, we always have the inequality that $|S| \leq |T|$; that is, the maximum number of independent sensors is a lower bound on the minimum number of targets that are consistent with F.

The preceding geometric arguments highlight the intrinsic limits of target counting and tracking using sensor snapshots only. In a worst-case, where the targets move along arbitrary (adversarial) trajectories with arbitrarily changing velocities, we cannot hope to do better. However, in a more benign and practical setting where targets move smoothly, the temporal correlation in their trajectories can be exploited to track them with much greater resolution than promised by our worst-case results. In the following section, we develop a particle filter algorithm that takes advantage of our geometric framework for its sampling, and show through simulations and lab-scale experiments that it is quite effective in tracking multiple targets.

3. 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 $t \in \{1, 2, \cdots T\}$. 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 q^{th} target at the time instant t by $x_q[t]$, for $q \in \{1, \cdots Q\}$. The true trajectory of the q^{th} target is given by the set of its locations at the T time instants, that is, $\{x_q[t] : t \in \{1, \cdots, T\}\}$. 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 $\{y_q[t] : t \in \{1, \cdots T\}\}$, where $y_q[t]$ is an estimate of the location of the q^{th} target at instant t.

The use of particle filters for tracking a single target has been illustrated before in [18]. We next provide an outline of the approach used in [18], discuss its limitations in the setting of multiple targets, and then present its modified version tailored to the multiple target problem.

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, i.e., a candidate for $\{y_1[t'] : t' \in \{1, \cdots t\}\}$. Let us denote the k^{th} 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 till 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 choos-

ing $m > 1$ 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 of length $(t+1)$. Based on a *cost function* (specified shortly), the K lowest cost particles out of these mK particles are picked and designated as the K surviving particles at time $(t+1)$. At the last instant T, the particle with the smallest cost function out of the K particles $\{\mathbf{P}_k^T\}$ is picked and designated as $\{y_1[t] : t \in \{1, \cdots T\}\}$, i.e., it is our estimate of the true target trajectory. The basic premise underlying this approach is that, if the cost function is picked in accordance with the anticipated model of the target motion (e.g., small change in velocity between successive time instants), then particles that do not conform to this anticipated motion will eventually drop out due to large cost functions, while the surviving particles would be good estimates of the true trajectory.

We now specify the cost function used to prune the set of particles at each time instant. In keeping with the notion that high frequency variations in a trajectory cannot be captured by binary sensors [18], we choose the cost function to be an additive metric that penalizes changes in velocity. Let $\mathbf{P} = (\hat{x}[1], \cdots \hat{x}[n])$ denote a particle. The instantaneous estimate of this particle's velocity vector at time t is the increment in position $\hat{x}[t+1] - \hat{x}[t]$. The corresponding increment in the cost function in going from time t to $t+1$, which is taken to be the norm of the change in velocity, therefore is

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

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

While the "best" particle (i.e., the lowest cost particle at the last time instant T) provided by the above algorithm would hopefully fit the true trajectory the best, we also expect that a large fraction of the K particles $\{\mathbf{P}_k^T\}$ would not differ appreciably from each other, and would tend to form 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 any two targets are reasonably separable over time, we would expect that the K surviving particles at the last time instant would be comprised of two distinct clusters, corresponding to the two 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 quite serve our purpose. First, the naive scheme is likely to fail to distinguish between targets whose trajectories have significant overlap, since the corresponding clusters of particles would not be very distinct. Moreover, even when all trajectories are clearly distinguishable, the naive scheme can miss some of the targets. This can happen, for instance, when one of the targets (say q_1) is far from the others, and moves in a regular manner that leads to a small value of the cost function. 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 this problem would be to increase the number of particles as a function of time, but the number of particles needed to make this work, and the associated computational complexity, can be excessive. Instead, we propose an algorithm in which we identify cluster formation as we go, and limit the number of particles allocated to each cluster.

3.1 The ClusterTrack Algorithm

We call our proposed scheme ClusterTrack. The method is specifically designed to prevent a single target 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 which 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 last 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 j^{th} 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.

Step 2 of the Algorithm requires the generation of samples from the feasible target space $F[t]$. We employ the following simple sampling strategy. Suppose that $F[t]$ consists of N_t disjoint intervals (for higher dimensions, we would need to consider connected sets rather than intervals). These intervals are obtained from the set of subintervals g_i mentioned in section 2.2, by repeatedly merging overlapping subintervals till a disjoint set is obtained. We pick m_o samples randomly (in a uniform manner) from each of these N_t intervals, thereby generating a total of $m_t = m_o N_t$ samples. (This is done for each of the K_{t-1} surviving particles from time $t-1$).

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

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**

clusters, denote by $\mathbf{CH}_j$ the first particle that joined the j^{th} cluster, where $j \in \{1, \cdots, N_C\}$. We refer to this first particle $\mathbf{CH}_j$ as the cluster-head of the j^{th} 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 sum of the absolute differences between their components, that is, $D(\hat{\mathbf{P}}_i, \mathbf{CH}_j) = \sum_{l=1}^{t} |\hat{\mathbf{P}}_i[l] - \mathbf{CH}_j[l]|$. Compute $D(\hat{\mathbf{P}}_i, \mathbf{CH}_j)$ for each j, and compare the minimum of these distances against a threshold $D_0(t)$. (Note that the threshold is a function of the length of the particles). 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. The performance of ClusterTrack depends on the choice of the threshold sequence $D_0(t)$. Our numerical results bring out this dependence, and provide guidance for arriving at good choices for it.

For any particular time instant t, it is not guaranteed that the particles eventually selected by ClusterTrack will cover all the disjoint intervals that form the feasible target space $F[t]$. In practice, the particles do cover the feasible target space when sensors behave reliably, but this may not be the case when sensors exhibit severe non-idealities. In this situation, a simple extension of ClusterTrack can be used to generate new trajectories until all "unused" intervals are covered. For simplicity of exposition, we omit a detailed description of this scheme, although some of the presented results rely on this modification. (Details of this modification are available from the authors upon request).

4. SIMULATION RESULTS

We now present simulation results to evaluate the performance of our tracking algorithm. We begin with an ideal sensing model for each sensor, wherein each sensor has a fixed range, and detects the targets within its range without any misses.

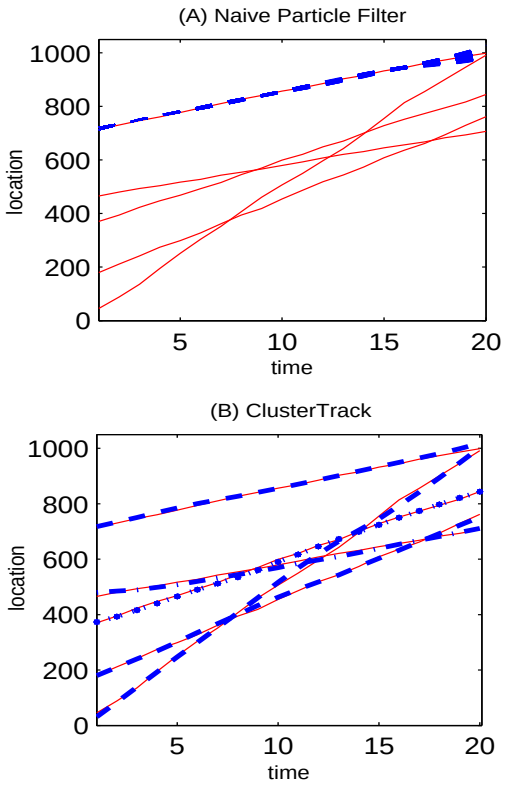

Figure 4: Comparison of the naive particle filter with ClusterTrack. The naive approach locks onto a single clearly distinguishable path. ClusterTrack tracks all paths quite well.

4.1 Tracking with Ideal Sensing

We consider a one-dimensional system with 30 sensors placed uniformly along a straight line (X-axis) starting from 0, and separated by a distance of 35 units each. The sensing radius of each sensor is 30 units (i.e., each sensor covers an interval of length 60). Five targets were considered, and we generated trajectories of 20 time instants for each one of them. The velocity of a particular target at each instant was picked randomly within 20% (on either side) 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. Each of the plots shown ahead is an x–t plot (location along the X-axis plotted against time). Solid curves are used to denote the actual target paths, while dashed curves denote the estimated trajectories.

We took $K = 500$ and $m_0 = 12$. For different choices of mean target velocities, we tested the performance of ClusterTrack, and also the naive particle filter. Fig. 4 depicts the results obtained for one such case. As can be seen, the naive approach is not able to detect all the targets and ends up giving a big cluster around the target that has a smooth path far away from the other trajectories. Insight into the failure of the naive algorithm is obtained from Fig. 5, which shows the surviving particles at different time instants. We can see that the particles corresponding to other targets drop out as the algorithm progresses.

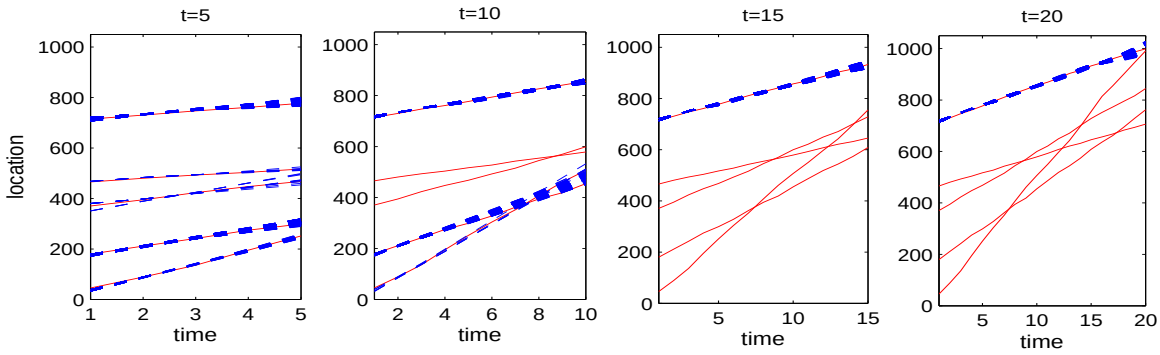

Figure 5: The progress of the Naive particle filter. All surviving particles lock onto one path as time progresses.

For ClusterTrack, we took the threshold sequence $D_0(t) = 45t$: that is, the threshold at time t is proportional to the length t of the particles being compared. The maximum number of particles retained for any cluster, H, was taken to be 30, while the maximum number of particles scanned at any particular time instant, L, was taken to be $5K = 2500$. The plot in Fig. 4(B) shows that the performance is quite good. The algorithm generates 5 clusters finally, and the best particle from each cluster is plotted in the figure. However, the dependence of ClusterTrack on the choice of parameters we make becomes evident if for the same example we take $D_0(t) = 15t$. Since the threshold has been lowered, we expect more clusters to arise [1], and indeed, the algorithm now generates 9 clusters. Picking the best particle from each of them provides us 9 estimated trajectories. However, we observe that 5 amongst these 9 still provide excellent approximations for the actual target paths. Furthermore, the additional spurious trajectories that we obtain are typically seen to be built out of portions of the true trajectories, and depending on the nature of the true trajectories[2], a subset of these spurious paths can usually be identified by their relatively higher cost functions. Indeed, such high-cost trajectories could be a perfectly good explanation of the sensor readings if our model allowed for trajectories which could exhibit rapid changes on occasion.

In general, for sensing radius R, we find that a threshold $D_0(t)$ between Rt and $2Rt$ works well for ensuring that ClusterTrack catches all of the paths most of the time. While our simulations yield useful design criteria, obtaining analytical rules of thumb for design of the threshold sequence $D_0(t)$ is an important topic for future work.

Next, we consider target motion where velocities vary appreciably, but smoothly with uniform accelerations. We evaluated the performance for different choices of accelerations, and observed that ClusterTrack still gives acceptable performance. Fig. 6 shows the performance for 5 targets moving with constant accelerations; the estimated trajectories are fairly good representations of the true paths. For multiple simulation runs, we observe that the number of trajectories obtained varies between 5 and 10, with 5 of the best 6 almost always approximating the true paths.

[1]While we intuitively expect more clusters for low thresholds, this may not always be true. For instance, making the threshold too small means that ClusterTrack approaches the naive scheme, which in this example would generate just one big cluster.

[2]If the true trajectories allow smooth transitions from one to another, then we may get spurious estimates that have low cost functions.

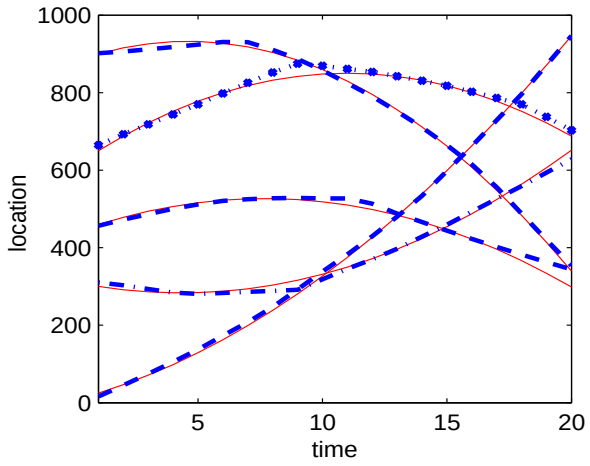

Figure 6: An example of the performance of ClusterTrack with constant acceleration motion.

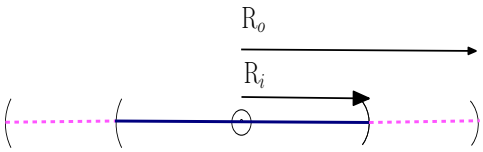

Figure 7: The non-ideal sensing model.

Next, we consider tracking with non-ideal sensing.

4.2 Tracking with Non-Ideal 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 non-ideal behavior (Fig. 7). 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

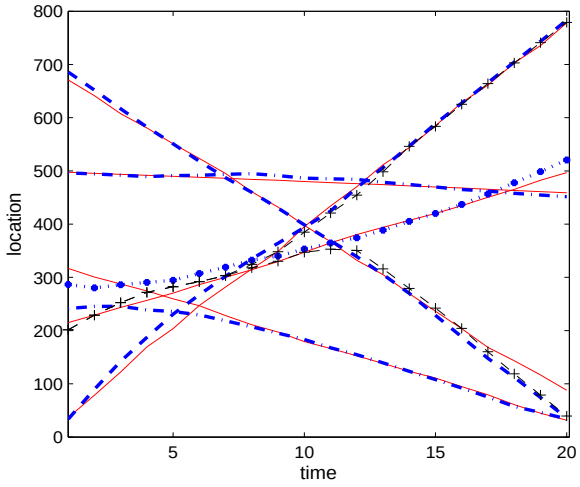

Figure 8: Performance with non-ideal sensing. In this particular example, ClusterTrack finds 7 trajectories, 5 of which approximate the actual paths.

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 non-ideal 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 set up for simulation is kept the same as before, with the modification that each sensor now has an $R_i = 30$ units, and an $R_o = 50$ units. In order to simulate the sensor readings, we assume that a target falling in the region of uncertainty of a particular sensor is detected with probability 0.5 by that sensor. 5 targets are considered, each moving within 20% (on either side) of its mean velocity. As shown in Fig. 8, ClusterTrack performs well. For different choices of $D_0(t)$, the number of trajectories obtained varies between 5 and 7. The figure shows a simulation run in which 7 trajectories were output. The best 5 are quite close to the actual paths. The remaining two trajectories are marked with plus signs (+), and we can see that they are made up of pieces of true paths. The costs associated with these 7 trajectories are [31.54 38.08 41.67 54.54 57.36 80.45 95.87] units, with the last two corresponding to the spurious paths. The results above demonstrate the robustness of the particle filter approach to non-ideal sensing.

Finally, we present some experimental tracking results from a lab-scale testbed with PIR sensors.

5. EXPERIMENTS

We use a small testbed with 5 PIR sensors placed uniformly along a line; see Figure 9. Each sensor sends a measurement to the base station when it changes state, and the base station is interfaced to a PC through a serial port. The data gets time stamped at the PC, so that each of the final set of measurements includes : 'value, position (mapped from node ID), and time'. For the ground truth regarding target trajectories, the (human) targets are provided with separate sensor nodes (equipped with localization engines)

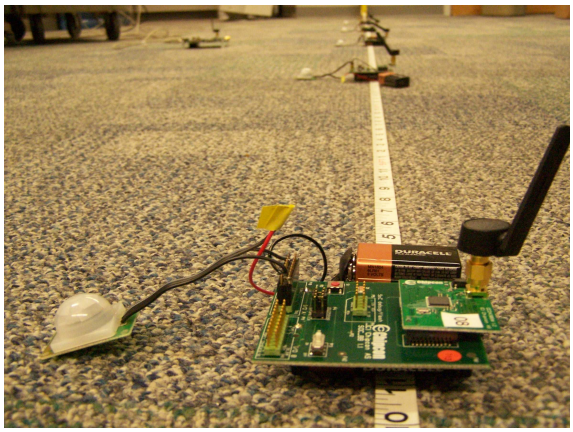

Figure 9: A view of the experimental set-up, with sensor modules placed uniformly along a line.

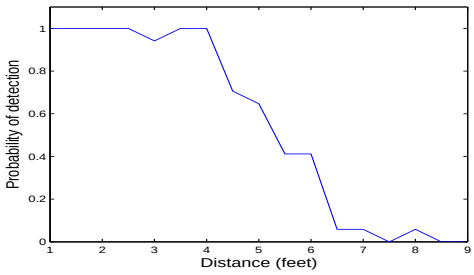

Figure 10: Probability of target detection against distance for a particular sensor module.

with buttons, which they press as they pass by a set of known locations on the way.

5.1 Sensor Characterization

We first performed some experiments to characterize an individual sensor module. The readings obtained were highly non-ideal, with the sensor completely failing to detect a nearby target on some occasions. Moreover, 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 towards the sensor. This is probably because these modules 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. In order to fit the sensor behavior to our non-ideal model of Fig. 7, we estimated the probability of detection with distance. Based on the results we obtained, shown in Fig. 10, we set $R_i = 3$ feet and $R_o = 6$ feet.

5.2 Tracking Performance

In our experiments, we placed the sensors uniformly along a line, separated by 4 feet (represented by the circles in Fig. 11). We considered two targets, which started from opposite ends and crossed each other. The non-ideal behavior of the sensors was evident as one of the sensors, placed at the

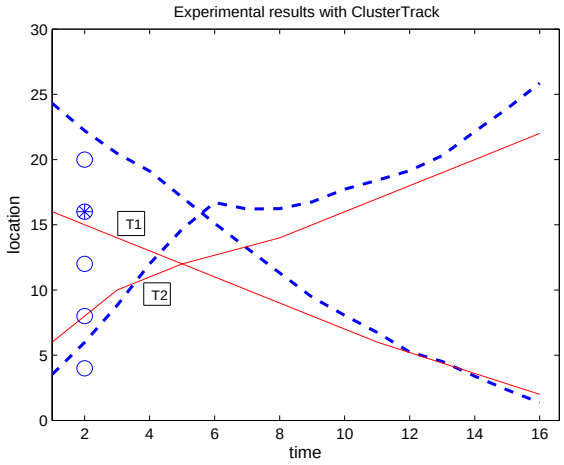

Figure 11: Experimental results with the sensor modules. Respectable performance is achieved, in spite of one sensor not even detecting a target.

location of 16 feet (shown by an asterisk $*$ inside the circle) completely missed the presence of target T_1. With the threshold sequence $D_0(t) = 2R_0t$, we ran ClusterTrack multiple times. In spite of the missed detection, we found that respectable to good tracking performance was achieved in most of the simulation runs (about 70%), as shown in the figure for one such good run.

6. CONCLUSIONS

The promising results obtained here, as well as prior results in [8, 18] for the same sensing model, indicate that binary 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 enough, 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. Is there a good way of combining the combinatorial techniques for target counting with the probabilistic techniques of particle filtering to enhance performance? Are there analytical rules of thumb for design of the threshold sequence $\{D_0(t)\}$ used in ClusterTrack? How broadly does our particle filter algorithm apply, in terms of robustness to different models for the targets' trajectories? When does it break down? How does tracking performance depend on the dimension of the space we operate in (in particular, how well can we track and count targets in two dimensions)?

7. REFERENCES

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40:102–114, Aug. 2002.

[2] A. Arora and et. al. A line in the sand: A wireless sensor network for target detection, classification, and tracking. *The International J. of Computer and Telecom. Networking*, 46:605–634, Dec. 2004.

[3] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, and D. Rus. Tracking a moving object with a binary sensor network. In *Proc. ACM SenSys*, 2003.

[4] M. Coates. Distributed particle filters for sensor networks. In *Proc. of IPSN*, pages 99–107, 2004.

[5] B. Jung and G. S. Sukhatme. Tracking targets using multiple robots: The effect of environment occlusion. *Autonomous Robots*, 13(3):191–205, Nov 2002.

[6] Z. Khan, T. Balch, and F. Dellaert. Efficient particle filter-based tracking of multiple interacting targets using an mrf-based motion model. In *Proc. IEEE/RSJ Conference on Intelligent Robots and Systems*, 2003.

[7] Z. Khan, T. Balch, and F. Dellaert. Mcmc-based particle filtering for tracking a variable number of interacting targets. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 27(11):1805–1819, Dec. 2005.

[8] W. Kim, K. Mechitov, J.-Y. Choi, and S. Ham. On target tracking with binary proximity sensors. In *Proc. IPSN*, 2005.

[9] J. Liu, M. Chu, J. Liu, J. Reich, and F. Zhao. Distributed state representation for tracking problems in sensor networks. In *Proc. of IPSN*, 2004.

[10] D. McErlean and S. Narayanan. Distributed detection and tracking in sensor networks. In *Proc. of 36th Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 1174–1178, 2002.

[11] M. Moghavvemi and L. C. Seng. Pyroelectric infrared sensor for intruder detection. In *Proc. TENCON*, pages 656 – 659, Nov. 2004.

[12] S. Oh and S. Sastry. Tracking on a graph. In *Proc. of IPSN*, April 2005.

[13] S. Oh, L. Schenato, and S. Sastry. A hierarchical multiple-target tracking algorithm for sensor networks. In *Proc. International Conference on Robotics and Automation (ICRA)*, April 2005.

[14] F. Z. Qing Fang and L. Guibas. Counting targets: Building and managing aggregates in wireless sensor networks. In *Palo Alto Research Center (PARC) Technical Report*, June 2002.

[15] D. Reid. Aan algorithm for tracking multiple targets. *IEEE Trans. Automatic Control*, 24:843–854, Dec 1979.

[16] Y. B. Shalom and X. R. Li. *Multisensor, Multitarget Tracking: Principles and Techniques*. YBS Publishing, 1979.

[17] J. Shin, L. Guibas, and F. Zhao. A distributed algorithm for managing multi-target identities in wireless ad-hoc sensor networks. In *Proc. of IPSN*, April 2003.

[18] N. Shrivastava, R. Mudumbai, U. Madhow, and S. Suri. Target tracking with binary proximity sensors: Fundamental limits, minimal descriptions, and algorithms. In *Proc. of ACM SenSys*, 2006.

[19] K. R. Songhwai Oh, Inseok Hwang and S. Sastry. A fully automated distributed multiple-target tracking and identity management algorithm. In *Proc. AIAA Guidance, Navigation, and Control Conference*, August 2005.