Multiple-Sensor Fusion Target Tracking using 

*ClusterTrack* Algorithm

S.M.R. Farshchi  
ASR Centre for Embedded Systems Research  
Sadjad Institute of Technology  
Iran, Mashhad.  
Mr.farshchi@sadjad.ac.ir

Hossein Sharifi Noghabi  
School of Engineering and Design  
Sadjad Institute of Technology  
Iran, Mashhad.  
hsharifi219@sadjad.ac.ir

**Abstract**— Despite the minimal information provided by a binary proximity sensor, a network of these sensors can provide significant target tracking performance. This article deals with the performance examination of such a network for tracking multiple targets. We began with geometric arguments that address the problem of counting the number of distinct targets, given a snapshot of the sensor readings. Then necessary and sufficient criteria provided for an accurate target count in a one-dimensional setting, moreover, a greedy algorithm defined 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. Finally, an extensive set of simulations, as well as experiments with passive infrared sensors, are reported.

I. INTRODUCTION

Wireless sensor networks are set of tiny equipment’s that have sensing processing and communicating competences together and provides lots of applications [1]. Amongst them, target tracking is a unique and important application that requires a cooperative processing to obtain robust results [2]. We assay the problem of tracking targets using an energy-efficient target tracking of binary sensors. There are two kinds of binary sensing models: the ideal binary sensing model and imperfect model. In the ideal model, each node can detect the target whenever it appears in the node’s sensing range $R$ (Figure 1(a)). Actually, detection ranges often vary regarding the environmental situations, such as the positioning of the target and the sensor. These factors make target detection near the boundary of the sensing range less predictable. Mentioned fact leads to an imperfect model in which the target is always detected within an inner disk of radius $R_{in}$ but it is detected only with certain nonzero probability in an annulus between the inner disk and an outer disk of radius $R_{out}$. No detection will happen out of outer disk (Fig. 1(b)).

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, deciding how many distinct targets are in the range, or provide any location-specific information. Despite the minimal information provided by an individual sensor, a collaborative network has been shown in prior work [3] to yield respectable performance when tracking a single target: the resolution with which the target can be localized is inversely proportional to $p R^{d-1}$, where $p$ is the sensor density, $R$ is the sensing range, and $d$ is the dimension of the space. We study the problem of multiple targets tracking with binary sensors, without a priori knowledge of the number of targets.

Our focus is on the efficacy of collaborative tracking. 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 not considered here.

In this approach we present an innovative distributed, energy-efficient, and fault tolerant target tracking algorithm using binary sensor networks that is able to track a target in both ideal and imperfect binary sensing models. Each awake node can estimate target’s trajectory, location and speed locally, in cooperation with its neighbors. Likewise, our algorithm does not need synchronization for all networks and can track the target in real time under various paces and
moving in desultory directions. The algorithm in fault tolerant when a node fails to senses a target inside its range or lost packets because of collision.

The rest of the article is organized as follows: Section 2 literature review. Section 3 discusses our model and parameters briefly and then discusses the problem of target counting based on a snapshot of the sensor readings. Section 4, describes our particle filtering algorithm. Section 5 provides simulation results, while Section 6 describes our experimental setup and results. We end with Section 7, conclusion.

II. RELATED WORKS

The problem of tracking multiple targets using sensor networks has been explored in many references [4]. Owing to its simplicity and minimal requirements, the specific use of binary sensors for tracking applications has also drawn considerable attention of late. However, most of the work related to binary sensing has been applied to tracking a single target [6]. The tracking techniques employed in the large-scale deployment in [7] 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 in [8] contained a distributed tracking method for a binary sensor network, but assumed perfect knowledge about the number of targets and their identities, unlike our approach.

In our work, we investigate both target counting and tracking. Prior work on counting targets includes [9] but it assumed more detailed sensing capabilities than 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, [10] 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 [11]. However, most prior work about using particle filters for tracking in sensor networks [12] has assumed a richer sensing model than the binary model we consider. Exceptions are the prior work in [13] on the use of particle filters for tracking a single target using binary sensing, and also the preliminary results from our conference publication [5]. In this article, we build on [13] 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 [14] only considered a one-dimensional setting).

III. PROCEDURE FOR PAPER SUBMISSION

A. Network Model and Assumptions

There are $N$ nodes that randomly uniformly distributed a delimited two-dimensional region for monitoring. Each node has a unique ID and surely all the nodes together can cover entire region. To simplify, it is assumed that sensors have the same sensing range $R$ under ideal sensing or the same radii $R_{in}$ and $R_{out}$ under the imperfect sensing. Nonetheless, this approach can work even when ranges vary from sensor to sensor. Each node begets one bit of information only when there is a change in target’s status. Otherwise we get no information about features of the target. While there is no detection node remains silent to save energy and bandwidth. When it generates a new bit of information, in will be sent to the neighbors, nodes whose sensing ranges have intersect with node’s coverage range.

Another assumption is that sensors are immobile. For example, Unattended Ground Sensors (UGS) used in the military and security applications to make this case worthy of study. We also assume that each node knows its own location. This assumption can be satisfied by using some low-power GPS or localization methods [10]. Since in this algorithm each sensor estimates target’s position concerning to its location and range, in order to report these outputs, they do not need to be acquainted of their position in planer region. To estimate velocity, node requires position of neighbors adjacent to each other which can be organized via triangulation. Hence, it is not necessary for nodes to know geographical location of each node. In order to estimate trajectory, the location of neighbors specified by triangulation and then locating each node’s geographical situation on the basis of some of the neighbors having GPS is a proper procedure [13] that can be done at the network deployment stage, but it is not mentioned here. Sensors exchange their location information through communication at the network deployment stage. Each sensor has its own local timer and can time stamp sent or received messages. Additionally, we assume that the target moves with velocity that is low relative to the node’s sensing frequency. Consequently, time of discovery of the change in the target’s presence within the node’s sensing range differs little from the time at which the target moves within or out of this range under the ideal binary sensing model.

B. Snapshot-Based Inference

Our investigation began with 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.

C. Binary Sensing

Some spatial separations amongst the targets are 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, only this factor is not enough to count the number of targets accurately.

Even arbitrarily large individual separation is not sufficient to reliably count a set of targets using binary sensors. 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 \( m \).

D. 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 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 \( \{ C_i \} \), thus, 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. An example is shown in Figure 2.

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 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 \( x \in F \), 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 \( 2R \).

IV. Tracking Algorithm

To demonstrate our basic idea, we use the example in Figure 4, which shows a target moving through an area covered by three nodes. At first, the target is outside of the sensing ranges of the nodes. Later, it falls in the sensing range of node \( N_4 \) at time \( t_1 \), and then sensing ranges of \( N_5 \) at time \( t_2 \) and \( N_6 \) at time \( t_3 \). At last, it leaves sensing ranges of nodes, in that sequence, at times \( t_4, t_5, t_6 \) respectively.

The initial idea of the tracking algorithm under the ideal model was mentioned in [13, 14] and it can be mentioned briefly as follows. At the time of arrangement, first each node initializes its neighbor’s status to “0” on its own list. Whenever a sensor received “1” from a adjacent node it updates the status of that neighbor to value “1” on the list. At the moment at which the node senses a change in target’s existence within its range, it identifies the arc of its sensing range border circle that the target is crossing. The target location is estimated as the middle point of the corresponding arc and broadcasted to neighbors. We can use two different local times when the target tracked in different locations to estimate its velocity. A weighted line fitting method is used to find a line, approximating a fragment of the target trajectory, that best fits the estimated target locations.

A. Initialization and Information Update

First, each node generates a list of its neighbors. Each record of the list saves these information including: neighbor node identifier, intersection points of the sensing circles of the node and its neighbor, an angle corresponding to the arc defined by these intersection points, and one-bit information produced by the neighbor, initialized to “0”. As soon as receiving value “1” from a adjacent sensor, the sensor alters the status of the related neighbor in the list.

B. Location Estimate

To determine the location, we use different combinations of angles related to intersection points. As an example in Figure 3. If the neighbors both outcome bit “1”, the corresponding central angles are combined by “&” operation that returns the intersection of these two angles. As shown in Figure 3(a), the common angle of \( \angle 1o3 \) and \( \angle 2o4 \) is \( \angle 2o3 \), so the node \( N_4 \) estimates the target location as the
moved within its sensing range. One special instance is shown in Figure 3(b), where the common angle is just one of the two angles.

C. 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 ≤ 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. 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 \( P = (x[1], \ldots, x[t]) \) 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 \( \dot{x}[n] = \dot{x}[n+1] - \dot{x}[n] \). The instantaneous contribution to the cost, in moving from time \( n \) to \( n+1 \), is taken to be the norm of the change in velocity

\[
    c[n] = ||(\dot{x}[n+1] - \dot{x}[n]) - (\dot{x}[n] - \dot{x}[n-1])||
    = ||\dot{x}[n+1] - \dot{x}[n] - \dot{x}[n-1] - 2\dot{x}[n]||
\]

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 \( \dot{x}[n] \) at time \( n \) to \( \dot{x}[n+1] \) at time \( (n+1) \), given that it had moved from \( \dot{x}[n-1] \) to \( \dot{x}[n] \) between time instants \( (n-1) \) and \( n \). The net cost function associated with the particle \( P \) is simply taken to be the sum of the incremental costs:

\[
    \sum_{n=2}^{t-1} c[n].
\]

V. TARGET MODELING

Since a moving target has dynamics time variant, a number of models can be established to describe a target’s motion. Even though more models can give better overall estimates, it is less efficient since more time will be required to yield the estimate. In the sense of acceleration of a target, the

constant velocity (uniform) and acceleration (maneuvering) modes are most commonly considered to build models. In this project, these two models are also used. Linear accelerations are normally quite small and thus can be reasonably covered by a process noise in a nearly constant velocity model, i.e., the constant velocity motion plus a zero-mean noise with an appropriate covariance representing the small acceleration [8]. Alternatively, this mode can be described as a constant velocity model with no process noise. On the other hand, the acceleration mode has the acceleration increment during the sampling time, and this should be included in the state space model. When the state space equation is given by:

\[
    x(k) = Fx(k-1) + Gw(k-1)
\]

\( x \) is the state vector of a target defined as:

\[
    x = [\xi \ \dot{\xi} \ \ddot{\xi} \ \eta \ \dot{\eta} \ \ddot{\eta} ]'
\]

where \( \xi \) and \( \eta \) denote longitudinal and lateral position respectively. In (3.1), \( w \) signifies process noise, which is zero-mean, white, and Gaussian with covariance \( Q(k) \).

The state transition matrices and the noise gain matrices for each mode can be written in the following forms:
get ww exclude subscripts 1 and 2 in equations 3
\[ G_1 = \begin{bmatrix} 1 & T & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & T & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad G_2 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \]
and
\[ F_1 = \begin{bmatrix} 1 & T & 0.5T^2 & 0 & 0 & 0 \\ 0 & 1 & T & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & T & 0.5T^2 \\ 0 & 0 & 0 & 0 & 1 & T \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad F_2 = \begin{bmatrix} 0.5T^2 & 0 & T & 0 & 0 & 0 \\ T & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & T & 0.5T^2 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \]

where subscripts 1 and 2 in equations 3 and 4 denote mode 1 (constant velocity motion) and mode 2 (acceleration motion) and T is the sampling time. Process noise covariance is simplified under the assumption that the process noise variance in each coordinate is equal and constant. In this case,
\[ Q(k) = \sigma_{v_1}^2 = \sigma_{v_2}^2 = q \]
and q can be chosen by using the following inequality:
\[ 0.5\Delta a_{\text{max}} \leq \sqrt{q} \leq \Delta a_{\text{max}} \]
where \( \Delta a_{\text{max}} \) is the maximum acceleration increment over a sampling time.

The measurement model can be written as:
\[ z(k) = Hx(k) + v(k) \]
\[ H_1 = H_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \]

The measurement matrix implies that only the position of the target is measured from each sensor.

The mode sequence is assumed to be a first order Markov chain with transition probabilities:
\[ \pi_{ij} = \begin{bmatrix} 0.95 & 0.05 \\ 0.05 & 0.95 \end{bmatrix} \]

These transition probabilities imply that a target keeps its current mode with high probability rather than changes its mode. It is obvious that the results of our tracking algorithm become more accurate with an increasing number of neighbors reporting the status of the target. To evaluate the role of the adjacent nodes in tracking a target, following simulation configuration was used: 800 nodes distributed over an area with 800 units by 800 units. Each node’s sensing range, R (\( R_{\text{out}} \) for imperfect model), differed from 40 to 150 units. To evaluate impact of sensor spatial density, first we assume fix range for sensing range, 40m, then, increase it to 150. If we diminish the area from 800m by 800m to 213.3m by 213.3m, in both of them number of neighbors changes but in the second also the spatial density of nodes per square meter increases. In terms of simulation for this concern placement of each node in relation to other nodes, remains constant and velocity of the target is adjusted proportionally to the sensing range, and stay constant if calculated in sensing range units.

All the paths limited inside the square area with length of 800– \( R_{\text{max}} \) located in the middle of the simulated region in order to exclude the boundary effect, where \( R_{\text{max}} \) is the maximum range (150 units) in the simulation. For the random turn trajectory, the length of each linear piece of the trajectory is random but proportional to the sensing range. As in [17], we set \( R_{\text{min}} = 0.9 \times R_{\text{out}} \) under the imperfect binary sensing model.

### A. 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 \( \rho R \) 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).

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 our algorithm 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 “*”), 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. The accuracy of location estimation resulting from our algorithm with fewer number of neighbors (sensing range of 40 units which means a node has 5 neighbors) is the same with outcomes of algorithm (1) with high number of neighbors (sensing range 150 units which means a node has 87 neighbors) under ideal binary sensing model. This can be proved by two ways, distributing enough sensors for other methods which is high and unnecessary for our approach. First we can use all distributed nodes to gain better accuracy and make our algorithm more fault tolerant. Because sensors fail with different reasons but our algorithm will still provide adequate accuracy even with failing of some of redundant nodes for location estimation. On the other hand our algorithm can works with some scheduling methods which means turning off some of the adjacent sensors to store energy. Our tracking algorithm achieves the same location estimate accuracy as algorithm (1) even when almost 95% of adjacent nodes are turned off.

We also analyzed the number of messages exchanged and their corresponding energy cost. Let the target move from point $O_1$ to point $O_2$ with velocity $v(t)$ over time $dt$ as shown in Figure 7. Area $R_0$ contains sensors that will broadcast bit “0” when the target moves from $O_1$ to $O_2$, and equal size area $R_1$ contains sensors that would transmit bit “1”. Hence, the total number of messages generated by the target moving from $O_1$ to $O_2$ will be $2A\rho$, where $A$ is the size of area $R_0$, $\rho$ is the sensor density per unit square. $A$ can be computed from Eq. (2), yielding $A = (2a+\sin(2a))R_2 \approx 4R_2a$. Thus, the total number of messages generated is $4R_\rho v(t)dt$. If over time $t$, the target moves distance $D$, then the number of messages produced is:

$$\int_{t=0}^{t_f} 4R_\rho v(t)dt = 4R_\rho D.$$

To verify this analysis, we performed two groups of simulations over an area of size 800 by 800 covered by 800 sensor nodes (the same density in each group). We set the sensing range $R$, 40 units in one group and 150 units in another. In both simulations, the target moves along a random trajectory (the same for each group) with a constant velocity and over the same distance. We ran the simulation 20 times for each group changing the topology of the network in each run. The exchanged messages were counted. The total number of messages exchanged is 474 when $R = 40$ units and 1759 when $R = 150$ units. The ratio is $1759/474 = 3.71$ which is close to the ratio of sensing ranges $150/40 = 3.75$.

We used a small testbed with five PIR sensors placed uniformly along a line; see Figure 5. 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 setup 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 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 → 0 transition that was

VI. CONCLUSIONS

Target tracking is one of the most important applications of sensor networks that can be done by collaboration between sensors. In this approach authors proposed a new algorithm for binary sensing models. Reducing the target tracking error is one of the important criteria in developing novel system. To achieve this aim, the sensor fusion technique and the Interacting randomized algorithm are applied. Since the Kalman filter based on a single state space model has a defect in the case that a target changes its mode, the proposed algorithm using more than two different models is inevitable. Even though sensor fusion and proposed algorithm are totally different techniques, these can cooperate to provide the optimal estimates. By comparing the simulation using a single sensor and the proposed algorithm. The error reduction is greater when three sensor data are fused. The advantage in using the proposed algorithm is not only error reduction but also mode prediction.

The introduced algorithm which is real-time distributed target tracking scheme without time synchronization for both the ideal and imperfect binary sensing models energy efficient and fault tolerant. In simulation phase, the accuracy of new algorithm tested under ideal model and outcome showed great precision. The analysis also demonstrated that for the assumption simulated, the application of sensors that hardly sense the target by the algorithm served to enhance the accuracy of localization minimized the estimation error for 50% in comparison with utilizing only sensors that do sense the target so significantly. Results of comprehensive simulations of this algorithm performed under different conditions and scenarios also verified that the presented algorithm overcomes other algorithms in terms of its accuracy of measuring the target location, velocity and trajectory applying the binary sensor networks.

REFERENCES


