Proc. IEEE Globecom, New Orleans, LA, December 2008, pp. 1-5

Distributed Target Tracking with Imperfect Binary Sensor Networks

Zijian Wang, Eyuphan Bulut, and Boleslaw K. Szymanski
Department of Computer Science and Center for Pervasive Computing and Networking, RPI, Troy, NY

***Abstract*—We study target tracking with wireless sensor networks in its most basic form, assuming a binary sensing model in which each sensor can return only 1-bit information regarding target's presence or absence in its sensing range. A novel, real-time and distributed target tracking algorithm for imperfect binary sensing models is proposed, which is an extension of our previous work on the ideal binary sensing model. The algorithm estimates target's location, velocity and trajectory in a distributed and asynchronous manner. Extensive simulations show that our algorithm achieves high performance and outperforms other algorithms in terms of accuracy of the estimation of target's location, velocity and trajectory.**

Keywords-target tracking; binary sensor networks; distributed algorithms; imperfect sensing

I. INTRODUCTION

Target tracking is a representative and important application of wireless sensor networks [1, 2]. One of the fundamental studies of target tracking focuses on a binary sensing that provides just one bit of information about the target, indicating whether it is present within the sensing range or not. There are two kinds of binary sensing models for binary sensor networks. In the ideal binary sensing model, each node can detect exactly if the target falls within its sensing range R (as shown in Fig. 1(a)). In an imperfect binary sensing model, the target is always detected within an inner disk of radius R_{in} but it is detected only with some nonzero probability in an annulus between this inner disk and the outer disk of radius R_{out}. Targets outside the outer disk are never detected (as shown in Fig. 1(b)).

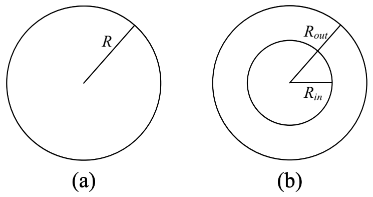


Figure 1. Ideal (a) and imperfect (b) binary sensing models

A number of approaches target tracking using binary sensor networks have been proposed in recent years. In the algorithms presented in [3, 4], sensors detecting the target presence first route their binary information to a central node and then the central node applies particle filters on information gathered from all sensors to update the target's location. Yet, particle filters are expensive to compute and transmitting data from each node to a central one is very costly in terms of the energy needed for communication for any non-trivial size network. In [5], each point on the target's path is estimated by the weighted average of the detecting sensors' locations. Then, a line that fits best the new point and the points on the trajectory established in the recent past is used as the target trajectory. In [6], the weight calculation is improved by using the estimated velocity to narrow the estimation of target location. However, these two methods require time synchronization of the entire network and assume that the target moves at a constant speed on a linear trajectory. Furthermore, they only use positions of the sensor nodes that detected the target. Actually, the absence of detection can be used to improve the tracking accuracy. In [7], both the presence and absence of the target within the node's sensing range were used to form local regions that the target had to pass. These regions are bounded by the intersecting arcs of the circles defined by the sensing ranges of the relevant nodes. The trajectory is estimated as a piecewise linear path with the fewest linear segments that traverses all the regions in the order in which the target passed them. However, the algorithm is centralized and complex to compute. It also requires a designated tracker node to fuse data. This designated node has to accumulate information from tracking sensors to form all regions needed to compute the estimated trajectory, which means that the tracking is not real-time but delayed.

In our previous work [8], we proposed a distributed target tracking algorithm for the ideal binary sensing model. In it, each active node computes the target's location locally but uses cooperation to collect the sensing bits of its neighbors. Furthermore, the algorithm tracks the target in real–time, does not require time synchronization between sensor nodes and can be applied to targets moving in random directions and with varied velocities.

In this paper, we extend our previous work and propose a distributed target tracking algorithm that can be used for an imperfect binary sensing model while keeping all the other properties of its predecessor.

II. DISTRIBUTED TARGET TRACKING ALGORITHM

A. An Overview of The Ideal Binary Sensing Algorithm

1) Basic Idea

To illustrate our basic idea, we use an example from Fig. 2, which shows a target moving through an area covered by three nodes with the ideal binary sensing. Each node will generate a bit "1" when it first senses the target's presence, and later a bit "0" when it first stops sensing its presence. Those are the times at which the target enters and then exits sensing range of that node. Consequently, at the transition time t_j, the target must be on arc A_j which is a part of the border circle of the sensing range of the node reporting the bit information. This arc can be determined cooperatively from presence and absence bits of neighbors of that node. Let's consider arc A_2 defined at time t_2 as an example. At time t_2, node Y senses the target within its sensing range for the first time, so the arc is a part of the sensing range border circle of node Y. At that time, node Y knows that the target is within the sensing range of node X, so the target must be on arc "abc". Node Y knows also that the target is not within the sensing range of node Z, so the target can not be on arc "bcd". Hence, node Y concludes that the target must be on arc denoted as A_2. It is important to observe that, by using this method, the two-dimensional uncertainty of the target's location on the plane is reduced to a one-dimensional uncertainty within the circle section. The shorter this circle section is, the smaller the uncertainty becomes.

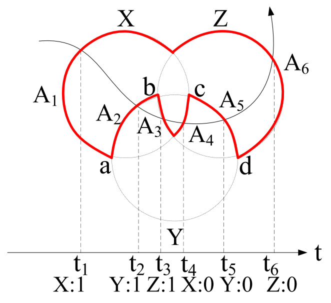


Figure 2. An illustration of the basic idea behind the algorithm

2) Target Tracking Algorithm

At the network deployment stage, each node establishes a list of its neighbors defined as nodes whose sensing ranges intersect the sensing range of that node. When the node discovers the change in the target's presence within its sensing range, it identifies the smallest arc of its sensing range border circle that the target is known to be crossing. The target location is estimated at the middle point of that arc.

We combine all angles corresponding to arcs defined by the neighbor list to determine such smallest arc. The four instances of this process are shown in Fig. 3. If both neighbors generated bits equal to "1", the corresponding central angles are combined by "&" operation that returns the intersection of the two angles. As shown in Fig. 3(a), the common angle of $\angle 1o3$ and $\angle 2o4$ is $\angle 2o3$, so the node Y estimates the target location as the middle point of arc "23". Sometimes, as shown in Fig. 3(b), the common angle of the two angles is equal to one of them. If status of one neighbor is "1" while status of the other is "0", then the corresponding central angles are combined with "-" operation that returns the angle formed by excluding the second angle from the first one. For example, in Fig. 3(c) $\angle 1o3$ - $\angle 2o4$ is equal to $\angle 1o2$. In a special case shown in Fig. 3(d), the result may consist of two angles, $\angle 1o2$ and $\angle 3o4$. The correct angle in this case is chosen by considering the recent estimate of the target location.

Let FA be the sought arc's central angle initialized to 2π (the entire circle of the sensing border of a node). Let IN be the set of neighbor nodes with status set to "1" and let OUT be the set of neighbor nodes with status set to "0". Then, the smallest angle whose corresponding arc the target must be crossing can be expressed as:

$$FA = FA \underset{i \in IN}{\&} angle_i \underset{j \in OUT}{-} angle_j \tag{1}$$

where $angle_i$ is the central angle corresponding to neighbor i.

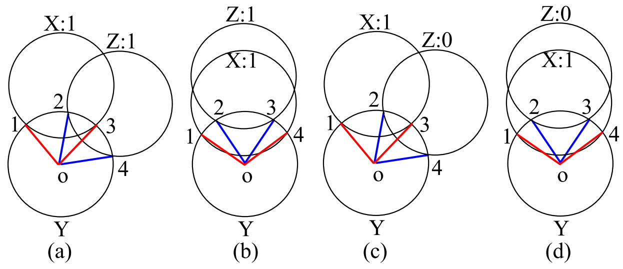


Figure 3. Instances of angle combinations

B. Tracking Algorithm for Imperfect Binary Sensing Model

To make our algorithm robust, as in [7], we take a worst-case approach to the information provided by the imperfect binary sensing model: if a sensor output is "1", then we assume that the target is somewhere inside the large disk of radius R_{out}; if a sensor output is "0", then we assume that the target is somewhere outside the small disk of radius R_{in}. The main influence of the imperfect binary sensing model is that we can no longer identify circular arcs that the target crosses (as was possible in the ideal binary sensing model) when there is a change in the status of the target sensed by a node. Instead, we can only conclude that the target must be within the ring determined by R_{in} and R_{out}. However, we can use a thin ring section which is determined by the neighbor outputs to approximate the circular arcs and then estimate the position of the target. Although this expands the one-dimensional uncertainty of the target's location to a two-dimensional uncertainty, if the resulting ring section is short and thin, the error still will be small.

1) Initialization

In the initialization procedure, each node establishes a list of its neighbors and calculates the exact angle corresponding to a neighbor depending on the output and the relative position of that neighbor. The three instances for neighbor (node Y) that outputs bit "1" are shown in Fig. 4. As described previously, if node Y outputs bit "1", we can only be sure that the target is within sensing range R_{out}. When node X senses there is a change in the status of the target, it knows that the target is within the ring determined by R_{in} and R_{out}. Depending on the relative position of node Y to node X, there could be up to two angles corresponding to node Y resulting from the intersection of R_{in} and R_{out} circles of node X and R_{out} circle of node Y. If two angles for node Y exist, we choose the angle that ensures that the target is within this angle; for example we choose $\angle b_1ob_2$ in Fig. 4 (a) and $\angle a_1oa_2$ in Fig. (b) as angles corresponding to neighbor Y. If only one angle exists for node Y, then it is the corresponding angle, as shown in Fig. 4 (c).

Three cases need to be considered when neighbor (node Y) outputs bit "0" are shown in Fig. 5. As described previously, if node Y outputs bit "0", we are only certain that the target is outside of sensing range *Rin*. Depending on the relative position of node Y to node X, there could be up to two angles corresponding to node Y resulting from the intersection of *Rin* and *Rout* circles of node X and *Rin* circle of node Y. If two angles exist for node Y, we choose the one that ensures that the target is outside of it; for example we choose $\angle a_1 o a_2$ in Fig. 5 (a) and $\angle b_1 o b_2$ in Fig. 5 (b) as the angles corresponding to neighbor Y. In case illustrated in Fig. 5(c), we can not determine that the target is outside $\angle a_1 o a_2$ because the target could be within $\angle a_1 o a_2$ regardless of node X output. So, if node Y outputs bit "0", it will be considered a neighbor of node X only if its *Rin* circle intersects with the *Rin* circle of node X.

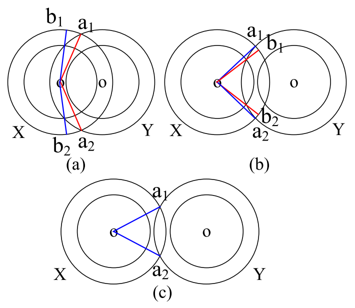


Figure 4. An angle corresponding to neighbor's output "1"

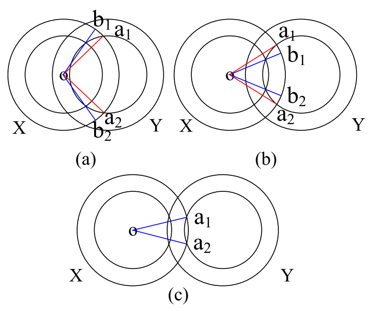


Figure 5. An angle corresponding to neighbor's output "0"

2) *Location Estimate*

At the moment at which the node discovers the change in the target's presence, it calculates the final angle corresponding to the ring section that the target is crossing using the same angle combination method as was used in the ideal binary sensing model. Then, the thickness of the ring section is recalculated to improve the estimation of target position.

We calculate the intersection points of sensing range circles of each pair of node X's neighbors that output bit "1". The intersection point that falls into the final angle and is farthest away from the center of node X defines one of the boundaries of the ring section. It also determines the thickness of the ring section making it as thin as possible. More precisely, a new thinner ring section is determined by this intersection point, *Rout* circle and final angle. For example, ring section "abcd" shown in Fig. 6 (a) ends up with thickness "ab". Please note that the neighbor node that outputs bit "0" contributes only to the angle combination but not to the thickness calculation. As shown in Fig. 6 (b), the recalculated ring section may exclude some area into which the target may fall, although with small probability because this area is near *Rout* circle of node X. Moreover, when the final angle is small, this area will be negligible in size. The target position is estimated to be the center of this ring section.

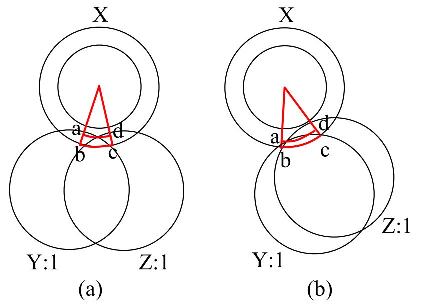


Figure 6. Ring section thickness calculation

3) *Velocity Estimate*

We use a distributed, asynchronous algorithm to estimate the target velocity. As shown in Fig. 7, three nodes X, Y and Z work in asynchronous time. At time t_{Y1} on node Y's local clock, node Y senses target's presence for the first time and generates a bit "1" message. The estimated location of the target is also included in this message to save energy and bandwidth. Since the elapsed time of radio transmission is negligible, node Z receives this message at time t_{Z1} on its local clock. Node Z will also receive the message from node X at time t_{Z2}. Then, node Z can use the time difference t_{Z1}-t_{Z2} and the difference of locations reported in these two messages to estimate the target velocity. To estimate velocity accurately, only location estimates with relatively high accuracy are used (those are locations at the centers of the short and thin ring sections).

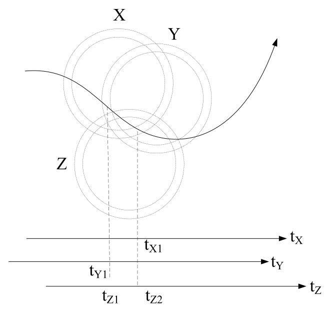


Figure 7. Velocity estimation

4) *Trajectory Estimate*

A weighted line fitting method is used to get the target trajectory and the weight of each estimate is defined as:

$$w = \frac{1}{|\text{ring section}|/|\text{ring}|} \tag{2}$$

where |ring section| is the area of corresponding ring section whose center point is the estimated target location and |ring| is the area of the ring determined by circles *Rin* and *Rout*. Each node finds the line (or two or more line sections if the target turns around) that best fits these weighted estimated locations. This line can be expressed as $y = a \cdot x + b$ allowing us to define the corresponding minimization metric Q as:

$$Q = \sum_{i \in E} w_i (y_i - a \cdot x_i - b)^2 \qquad (3)$$

where $E=[(y_0,x_0),\ldots(y_i,x_i)\ldots(y_k,x_k)]$ is a list of the estimated target locations for which the line is fitted.

III. SIMULATION

We have designed a QT based simulator and used data exchange between multi-threads to simulate wireless communication between sensor nodes. We assume that there is some MAC (Media Access Control) protocol supporting ideal wireless communication, so simulations have not modeled collisions or dropped packets.

A. *Location Estimate*

The first metric that we consider is the location estimation error, measured as the distance between the estimated and real target locations relative to the sensing range R_{out} to reflect the angular error independent of the scale. Hence, this metrics should decrease with the increase of network density.

1) *Simulation Setup*

When evaluating the impact of network density on the location estimation accuracy, we kept the number of nodes fixed at 800 within 800 by 800 area and varied the sensing range R_{out} from 40 to 150 units. The velocity of the target was adjusted proportionally to the sensing range, making it constant if measured in sensing range units. Several types of trajectories have been considered, including linear, circular, and a piece-wise linear trajectory with random turns. In order to exclude the boundary effect, all the trajectories are confined within the square area with length of 800-R_{max} in the middle, where R_{max} is the maximum sensing range (150 units) in the simulation. For the random trajectory, the length of the trajectory is proportional to the sensing range R_{out} . As in [5], we set R_{in} =0.9*R_{out}.

2) *Detection Probability*

Two kinds of detection probabilities for imperfect binary sensing models are used. The first one is a constant distribution as defined in formula (4), where d is the distance between the sensing node and the target.

$$\begin{cases} \dfrac{R_{out} - d}{R_{out} - R_{in}} & R_{in} \le d \le R_{out} \\ 1 & d \le R_{in} \\ 0 & R_{out} \le d \end{cases} \qquad (4)$$

The second one is an exponential distribution defined by Eq. (5), where α is its exponent parameter. To make the detection probability approximately 0 when d=R_{out}, we also let $e^{-\alpha(R_{out}-R_{in})} = 0.01\%$, yielding $\alpha = \ln(0.01\%)/(R_{in} - R_{out})$.

$$\begin{cases} e^{-\alpha(d-R_{in})} & R_{in} \le d \le R_{out} \\ 1 & d \le R_{in} \\ 0 & d \ge R_{out} \end{cases} \qquad (5)$$

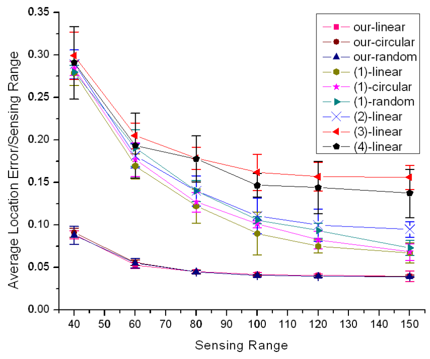
3) *Algorithms to be Compared*

We compare our algorithm with the following other four algorithms introduced in [5] and [6]: (1) Equal Weight: target's position is estimated as the average of the detecting sensors' positions. (2) Distance Weight: target's position is estimated as the weighted average of the detecting sensors' positions. The weight for each node is set at $1/\sqrt{R_{out}^2 - 0.25(v \cdot t)^2}$, where v is the target velocity and t is the time expired since the target has been detected. (3) Duration Weight: target's position is estimated as the weighted average of the detecting sensors' positions. Given the time t that expired since the node has detected the target, the weight for each node is ln(1+t). (4) Line Fit: the initial estimate of the target position is made as in algorithm (2), and then a line that fits the history target position point is found and the current target position is refined using this line and the target velocity. Model based target tracking methods use totally different approaches, hence comparison with them was judged to be beyond the scope of this paper

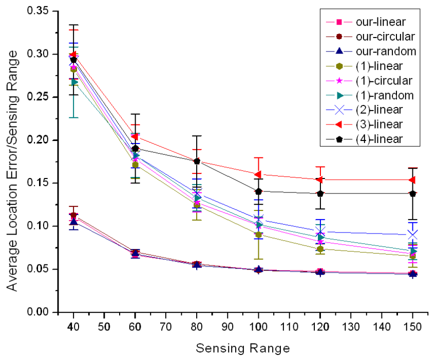
Algorithms (2), (3) and (4) are designed for a linear trajectory and constant target velocity, so we compare them with our algorithm using only this kind of target movements.

4) *Simulation Results and Discussion*

We ran the simulation for ten times and computed the average and confidence interval of the results under confidence level of 95%. Fig. 8 shows the location estimate accuracy under the two considered detection probabilities. In both cases, our algorithm bests all four other algorithms. Even for the sparse network with sensing range R_{out} = 40, in which each node has on average only five neighbors, the algorithm still performs well. Additionally, the location estimates of our algorithm have nearly the same accuracies for all three trajectories, indicating independence of our algorithm from the trajectory type.



(a) The results under the first detection probability

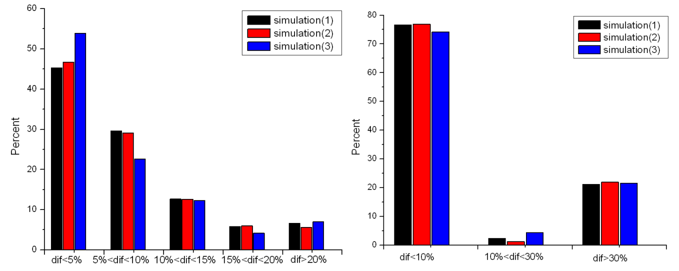


(b) The results under the second detection probability

Figure 8. The location estimate accuracy

B. Velocity Estimate

We tested the performance of velocity estimation under the configuration of 800 nodes with R_{out}=40 unit and R_{in}=0.9*R_{out} unit sensing ranges using the first detection probability in two scenarios in which the target moves along a linear trajectory. In the first scenario, the target moves at a constant velocity which is R_{out}/15 unit/second. In the second scenario, the velocity of the target changes suddenly several times during simulation by a random value that is a multiple of R_{out}/15 unit/second.

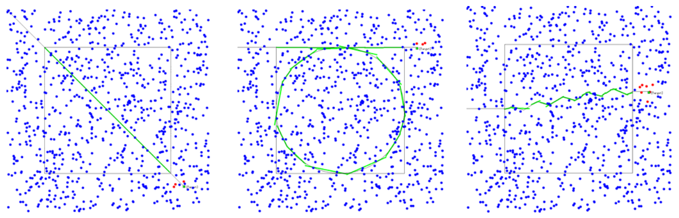


(a) Constant velocity (b) Random velocity

Figure 9. A histogram of differences between estimated and real velocities

We ran three simulations of each of the two scenarios and got a histogram of percentages of differences between estimated and real velocities shown in Fig. 9. It is clear that most of the time (around 80%) the difference between real and estimate velocity is within 10% in both scenarios. And less than 10% of time the difference is above 20% in the first scenario. The difference is a little larger in the second scenario because there is some delay before the change of real velocity is reflected in its estimate. As a result, there are large deviations in the brief moments immediately after the velocity change.

C. Trajectory Estimate



(a) Linear trajectory (b) Circular trajectory (c) Random trajectory

Figure 10. Examples of trajectory estimation (red dots indicate sensor currently tracking the target)

Fig. 10 shows typical trajectory estimations for three trajectories with the configuration of 800 nodes, R_{out}=40 units, and R_{in}=0.9*R_{out}, using the first detection probability. We measure the accuracy of estimated trajectory using the average difference between the estimated and real trajectories. It is calculated using the area of a polygon formed by these two trajectories divided by the length of the real target trajectory. The average accuracies are 0.287, 1.811 and 2.873 units, for linear, circular and piece-wise linear trajectories with random turns, respectively.

IV. CONCLUSION

In this paper, we extend our study of target tracking problem under the ideal binary sensing model and introduce a real-time distributed target tracking algorithm without time synchronization for imperfect binary sensing. Extensive simulations of this algorithm performed under different configurations and scenarios are reported. We observe that our algorithm yields good performance and outperform other algorithms by estimating accurately the target location, velocity and trajectory. Our target tracking method can be adapted for multi target tracking when the targets are sufficiently far from each other to be independently detected. In our future work we will consider the case of tracking target close to each other by assigning target id to the newly detected location based on each target's past location estimates.

ACKNOWLEDGEMENT

Research was sponsored by US Army Research Laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defence, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora and M. Miyashita, "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.

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

[3] P. M. Djuric, M. Vemula, and M. F. Bugallo, "Signal processing by particle filtering for binary sensor networks," Proc. 11th IEEE Digital Signal Processing Workshop & IEEE Signal Processing Education Workshop, pp. 263-267, 2004.

[4] T. Jing, S. Hichem, and R. Cedric, "Binary variational filtering for target tracking in sensor networks," Proc. IEEE/SP 14th Workshop on Statistical Signal Processing, pp. 685-689, 2007.

[5] K. Mechitov, S. Sundresh, Y. Kwon, and G. Agha, "Cooperative tracking with binary-detection sensor networks," Technical Report UIUCDCS-R-2003-2379, University of Illinois at Urbana-Champaign, September 2003.

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

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

[8] Z. Wang, E. Bulut, and B. K. Szymanski, "A distributed cooperative target tracking with binary sensor networks," Proc. IEEE International Conference on Communication Workshops, May 2008, Beijing, China, pp. 306-310.