Probabilistic Support Vector Machine Localization in Wireless Sensor Networks

Reza Samadian and Seyed Majid Noorhosseini

Sensor networks play an important role in making the dream of ubiquitous computing a reality. With a variety of applications, sensor networks have the potential to influence everyone's life in the near future. However, there are a number of issues in deployment and exploitation of these networks that must be dealt with for sensor network applications to realize such potential. Localization of the sensor nodes, which is the subject of this paper, is one of the basic problems that must be solved for sensor networks to be effectively used. This paper proposes a probabilistic support vector machine (SVM)-based method to gain a fairly accurate localization of sensor nodes. As opposed to many existing methods, our method assumes almost no extra equipment on the sensor nodes. Our experiments demonstrate that the probabilistic SVM method (PSVM) provides a significant improvement over existing localization methods, particularly in sparse networks and rough environments. In addition, a post processing step for PSVM, called attractive/repulsive potential field localization, is proposed, which provides even more improvement on the accuracy of the sensor node locations.

Keywords: Wireless sensor networks, support vector machine, localization, probabilistic SVM, machine learning, neural networks.

Manuscript received Nov. 17, 2010; revised May 13, 2011; accepted May 20, 2011.
This work is supported by Iran Telecommunication Research Center (ITRC).
Reza Samadian (phone: +98 21 64542740, email: pe.samadian@gmail.com) and Seyed Majid Noorhosseini (email: majidnh@aut.ac.ir) are with the Department of Computer and Information Technology, Amirkabir University, Tehran, Iran.
http://dx.doi.org/10.4218/etrij.11.0110.0692

I. Introduction

Recent advances in technology and especially electronics, have allowed sensor nodes to become cheaper, smaller, and thus, more applicable. In fact, sensor networks have made the concept of pervasive computing become more realistic. There are many applications for wireless sensor networks (WSNs). Some examples of the use of deployed sensor networks are wild animal behavior observation, glacier monitoring, ocean water monitoring, rescue of avalanche victims, tracking military vehicles, sniper localization and many more, as mentioned in [1].

However, for the WSNs to be sufficiently useful, there is a vital need for the nodes to be localized. Suppose that in a forest fire detection application, a sensor node reports of a temperature increase in its temperature sensor. Unless we know where the report is coming from, the information is almost useless. There are similar issues in many other applications.

In WSNs, the problem of localization is the ability to determine a node's relative or absolute position [2]. While designing a localization procedure for the WSNs, there are some challenges to face. First of all, a localization algorithm has to be energy-aware. That is, we prefer localization procedures that impose less communication as well as less computation. On the other hand, as we need a single sensor node to be really cheap, a localization algorithm has to be low in costs. That is, we prefer algorithms which require fewer types of equipments on the sensor nodes.

The first idea that comes to mind when talking about localization is using GPS on each node. However, the problem with the GPS is that it needs an extra piece of equipment on the sensor node, which is an expensive one. Furthermore, a GPS device on a sensor node imposes space issues, which is another

challenge in WSNs.

Considerable efforts have been made in the field of localization in WSNs. Proposed methods can be categorized mainly into two groups: range-based and range-free localization algorithms. Range-based algorithms (that is, one-hop approaches) are those which use range measurement equipment. The methods used for measuring range in these devices is usually a received signal strength indicator (RSSI), time of arrival (ToA), time difference of arrival (TDoA), or angle of arrival (AoA)/direction of arrival (DoA) [2]. Each node then uses a method like triangulation to localize itself. Range-free algorithms, in contrast, do not hold any assumption of existence of any range measurement devices on sensor nodes. Therefore, range-free techniques, although more complicated, are better choices if they achieve an acceptable accuracy.

In this paper, support vector machines (SVMs) will be used to develop a range-free localization algorithm. Previously, [3] and [4] proposed to use SVMs for the localization of the sensor nodes. However, in this study, we will try to improve the accuracy of localization by means of probabilistic SVMs. As it will be observed in the following sections, a probabilistic SVM is more powerful than the simple SVM in this application. Furthermore, the optional modified mass spring optimization (MMSO) phase [4] will be replaced with the innovative attractive/repulsive potential field localization (ARPoFiL) method so that more accurate results can be achieved.

The rest of this paper is organized as follows. In section II, SVM localization will be explained and probabilistic SVM for the localization in lieu of SVM will be proposed. In section III, the innovative approach for further modification of nodes' positions will be introduced. Section IV briefly reviews the experimental results. Finally, in section V, the study will be concluded and suggestions will be made for future research.

II. Support Vector Machine Localization

A common problem in machine learning is classifying data in a predefined feature space. To be more precise, assume a feature space of k dimensions and some samples like (X_i, Y_i) are at hand, where X_i and Y_i are the feature vector and class label of the i-th learning sample, respectively. Using this learning set, a model is to be developed by which, given a new unlabeled sample (x) from the same feature space, the class that the sample belongs to can be determined. One of the empirically best solutions to the classification problem is SVM. SVMs were first introduced by Cortes and Vapnik [5] with the main idea to find the equation of a hyper-plane that divides the feature space such that the margin of the hyper-plane to each class is maximal. For samples that are not linearly separable, kernel functions which map samples to a higher dimensional space are used. Consequently, the classifier becomes nonlinear and thus, more powerful.

We now briefly review how to obtain SVM function. For details, refer to [5]. It is assumed that a two-class problem with labels 1 and −1 and k training samples exist. The class label of a new sample X is obtained by

$$sign[f(X,\alpha^*,b^*)] = sign[(w^* X + b^*)] = sign[(\sum_{i\in SV} \alpha_i y_i K(x_i^T, X) + b^*)]. \quad (1)$$

In (1), SV is the set of support vectors. Support vectors of each class are training samples that are closest to the samples of the other class in the feature space. The feature vector and class label for the i-th support vector are x_i and y_i, respectively. K is a kernel function. A linear kernel is actually the inner product of its parameters in a higher dimensional space. Also, α_is and b^* are parameters that can be obtained by solving (2) using quadratic programming techniques and (3), respectively.

$$\text{Maximize} W(\alpha) = \sum_{i=1}^{k} \alpha_i - \frac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{k} \alpha_i \alpha_j y_i y_j K(x_i^T, x_j), \quad (2)$$

$$\text{subject to} \sum_{i=1}^{k} \alpha_i y_i = 0,\ 0 \le \alpha_i \le C,$$

$$y_i(\sum_{i\in SV} \alpha_i y_i K(x_i^T, X) + b^*) = 1,\ 0 \le i \le k. \quad (3)$$

Generally, SVMs have the nice property of being robust against noise. They are also powerful in generalization, based on training sets. In the remainder of this section, the method of [6], [7], which uses SVMs to localize nodes in a WSN, will be reviewed briefly and probabilistic SVM will be discussed.

The main procedure of localization using SVMs as is mentioned in [6], [7] is as follows. Having a wireless sensor network consisting of N nodes, S_1, S_2, …, S_N, out of which first k nodes are beacon nodes, deployed in a 2D area $[0, D]^2$, first, beacon nodes start by broadcasting "Hello" messages across the network. Then, as each beacon node receives other beacon nodes' "Hello" messages, it can determine its hop-count. Next, each beacon node constitutes a vector of hop-counts to other beacon nodes, ordered by beacon node IDs. Each beacon node then sends this vector to the base station, as well as its position (which is assumed to be known). Then, the base station is all set up to start the training phase.

Actually, each hop-count vector gathered at the base station is a training sample with each hop-count being a feature and the location corresponding to each vector being a class. So, the base station starts the training phase with k samples of k dimensions. But as mentioned before, the SVM is used to classify the samples into two classes, while the value of location through dimensions, x and y, are continuous.

To overcome this problem, the authors proposed to consider $M-1$ classes for each dimension, where $M=2^m$. Then, for instance in x dimension, a sample belongs to the i-th class if $x \geq iD/M$. This way, a binary tree can be built for each dimension such that each tree node is an x dimension class. The classes will be assigned to the tree nodes such that if the tree is traversed like left child → parent → right child, the ordered list of the classes will be obtained: $cx_1, cx_2, \ldots, cx_{M-1}$. A test sample can be classified starting with the root node of the tree. If the test sample belongs to the class, it will be traversed to the right child, else, to the left child. The rest can be done in a divide and conquer manner.

Note that SVM decisions for a test sample are crisp. That is, each time the SVM decides whether a test sample belongs to a class or not, it never comes back to that decision. If the decision made was wrong, there will be an amount of error in the localization, especially when this wrong decision is made at one of the very top nodes of the tree. If the way SVM decides could be altered such that more smooth decisions could be made, the error could be reduced. This can be done by making the SVM define a degree of certainty in its decisions.

The idea of probabilistic SVMs was put forth first in [8], [9]. Returning to the definition of SVM, a sign function to convert the output of the SVM function, $f(X, \alpha^*, b^*)$, from a real number to classes 1 or −1 was used. Definitely, f is the equation of the hyper-plane dividing two classes in the feature space. So it can be said that the more a test sample is away from the hyper-plane, the more certain our decision becomes.

The idea behind the probabilistic SVM is to use some function in lieu of sign function to convert the output of the SVM function from an unbounded real number to a real number bounded between 0 and 1. Sigmoid function (that is, logistic function) has this property. The ordinary sigmoid function is given by

$$P(x) = \frac{1}{1+e^{-x}}. \tag{4}$$

The advantage of the sigmoid function is that while the input approaches ∞, the output of sigmoid function approaches 1, and while the input approaches $-\infty$, the output of sigmoid function approaches 0.

Properties of sigmoid function especially fit our demands for smoothing SVM decisions. If the output of the SVM function is passed to a sigmoid function, the output of the sigmoid function will therefore show the certainty of the SVM decision that the test sample belongs to class 1. Using this fact, all leaf nodes of the tree can be traversed and the location of a sensor node based on a weighted sum can be estimated. The modified recursive localization algorithm through x dimension using probabilistic SVM is as follows.

Algorithm. X-dimension localization (S, CurrentRoot):
Estimate the X coordinates of sensor S:
1. c = certainty of SVM prediction that S in class $cx_{\text{CurrentRoot}}$.
2. If ($cx_{\text{CurrentRoot}}$ is leaf node)
   -Return $x'(S) = (1-c) \times (\text{CurrentRoot} - 1/2) \times D/M + c \times (\text{CurrentRoot} + 1/2)\ D/M$.
3. Else
   - Left-child location = X-dimension localization (S, left-child of currentRoot).
   - Right-child location = X-dimension localization (S, right-child of currentRoot).
   - Return $x'(S) = (1-c) \times$ Left-child location $+ c \times$ Right-child location.

Probabilistic SVM can be further strengthened by defining variables A and B in the sigmoid function as

$$P[f(X)] = \frac{1}{1+e^{Af(X)+B}}. \tag{5}$$

An iterative procedure, which is based on maximum likelihood estimation, to find the optimum values for A and B has been proposed in [9], [10]. However, during experimental testing, we noticed that using an artificial neural network (ANN) for this purpose would generate more accurate values for A and B in a shorter time. In other words, it converges more rapidly. ANNs are one of the most reliable techniques used when it comes to a problem of optimization. The general procedure is as follows.

Having a set of n training samples labeled $\{\underline{x}_1, \ldots, \underline{x}_n\}$, with target outputs labeled $\{y_1, \ldots, y_n\}$, where $y_i \in \{1, -1\}$ and $i=1,\ldots,n$, the network is initialized with random values (usually 0) for the parameters to be optimized (here A and B). These parameters are considered as the weights of the network inputs. Each network input is simply an element of the input vector. Next, inputs are multiplied by their corresponding weights, being summed up in the destination perceptron, generating a weighted sum of the inputs. The activation function for the perceptron is selected such that its output is the target function for the optimization.

Using the above procedure, the output is calculated for each training sample as system output. Then according to the margin between system output and target output, that is, error, the weights are tuned such that the margin gets smaller. This is done using "gradient descent" rule which tries to lower error by altering weights, moving along the steepest descent of the error. To be more precise, the partial derivative of error against all weights is set to 0, and the optimal value for each weight is obtained. A factor, $0<\mu<1$, called "learning rate" usually impacts the amount of weight tuning during each iteration.

In our case, we used a simple single-perceptron neural network architecture which is depicted in Fig. 1. We also used weights W_1 and W_2 in place of parameters A and B and the

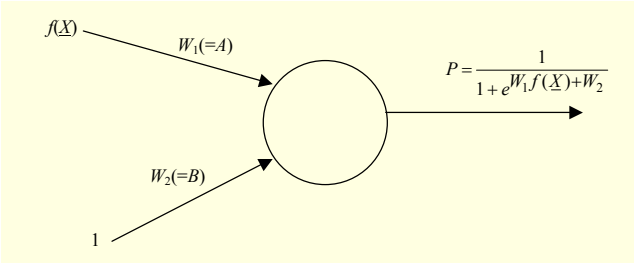

Fig. 1. ANN architecture for finding parameters of probabilistic SVM, A and B.

sigmoid function was used for the perceptron's activation function. By applying $f(\underline{X})$ and the constant integer 1 as the inputs to the network, the final system output P is calculated as is shown in the figure. Note that the system output is analogous to the probabilistic SVM's sigmoid function except that the parameters A and B are replaced by weights W_1 and W_2. The sigmoid function has the nice property that $P'(f(\underline{X}))=P(f(\underline{X}))(1- P(f(\underline{X}))$. The expected output for a training sample is 1 (a probability of 100%) when it belongs to class 1, and is 0 (a probability of 0%) when it belongs to class −1. The gradient descent rule used for altering weights after calculating system output for each sample is

$$\begin{cases} W_1 = W_1 + \mu O_s(1-O_s)(O_s - O_t) f(\underline{X}), \\ W_2 = W_2 + \mu O_s(1-O_s)(O_s - O_t), \end{cases} \quad (6)$$

where W_1 and W_2 are the weights to be learned (or A and B, respectively), μ is the learning rate (=0.9 in our case), and O_s and O_t are system output and target output, respectively.

Using the training samples for the SVM, this neural network was trained during the training phase of the network and the optimized parameters A and B were obtained. Results showed that with the use of A and B obtained by the ANN approach, the final $P[f(\underline{X})]$ for training samples belonging to class 1 was nearly 1 (for example, 99.98%) and for training samples belonging to class −1 was nearly 0 (for example, 0.01%) which means that almost the optimum A and B were in hand. In comparison, although the maximum likelihood method converges in each iteration, it finally returns parameters A and B which in practice, leads to less accurate results (for example, 0.8 to 0.9 and 0.1 to 0.2 for samples belonging to class 1 and −1, respectively). Therefore, we preferred to use ANN instead of the maximum likelihood approach.

III. Improving Accuracy: ARPoFiL Method

After the localization phase in each node, there is still room for improving the location estimates. For this purpose, each node sends its estimated position as well as its neighbor nodes' IDs to the base station. Then, the base station can gently modify sensor locations such that in the estimated locations model, neighborhood information still matches.

For this purpose, [11] has proposed the use of mass spring optimization (MSO) phase in the base station. The main idea behind MSO is that each node is considered a mass and each link to its neighbors is considered a spring. So, the node has to be stopped at a place where the resultant force of the springs is equal to zero. The energy of the system to be minimized is the sum of the energies in all nodes. So, the minimization of the nodes' energies is the goal, which is given by

$$E(S_i) = \sum_{\text{neighbor } S_j} [dist_{\text{est}}(S_i, S_j) - dist_{\text{true}}(S_i, S_j)]^2. \quad (7)$$

In (7), $dist_{\text{est}}(S_i, S_j)$ and $dist_{\text{true}}(S_i, S_j)$ are the estimated distance and true distance between nodes S_i and S_j, respectively. The resultant force on each node is then the vector sum of the force vectors of its neighbors which is given by

$$\overrightarrow{f(S_i, S_j)} = [dist_{\text{est}}(S_i, S_j) - dist_{\text{true}}(S_i, S_j)] \times \overrightarrow{u(S_i, S_j)}, \quad (8)$$

where the vector $u(S_i, S_j)$ is the unit vector from node S_i to S_j. Therefore, if a node's estimated distance to its neighbor is more than the true distance, a positive force towards that neighbor is impressed. The resultant force on node S_i which is the vector sum of the force vectors of neighbors is given by

$$\overrightarrow{F(S_i)} = \sum_{\text{neighbor } S_j} \overrightarrow{f(S_i, S_j)}. \quad (9)$$

Using the definitions above, a number of iterations are performed on all of the nodes. In each iteration and on each node, the energy and the resultant force of the node are calculated according to (7) and (9), respectively. Then the node is moved gently towards the resultant force vector by $\alpha_i F(S_i)$. α_i is recommended to be set to $1/2m_i$ where m_i is the number of neighbors of the node S_i [11]. If the energy in this configuration is lower than the one in the previous configuration, the new position for the node is accepted.

To modify this approach, the authors of [6] proposed to replace $dist_{\text{true}}$ with r which is the transmission range of any node, assuming all the nodes have the same transmission range. MMSO works fine and reduces the amount of error of SVM as shown in experimental results of [6], as well as our experiments (section V). However, we are going to use another fact about the nodes as well. MMSO successfully uses neighborhood information of the nodes. We are proposing to additionally use non-neighborhood information. In other words, not only does a node have to be in the transmission range of its neighbors, it has to be outside the transmission range of its non-neighbors.

Consider the problem of navigation in the robots. The problem is that a robot is placed somewhere in a field with obstacles, aiming to reach a goal point without hitting the obstacles. A method called attractive/repulsive potential field

has been proposed for this problem [12], [13]. The idea of this solution is simply to consider the robot and obstacles as positive potential fields and the goal point as a negative potential field. The robot is attracted by the goal and repulsed by the obstacles. The more the robot is near an obstacle, the more force repulses it from that obstacle. We used this idea for our sensor localization problem. In fact, ARPoFiL stands for attractive/repulsive potential field localization.

To implement this, the equations of the MMSO are modified to use non-neighborhood information as well as neighborhood information. Note that energy in (7) is a measure of how accurate the current location of a node is. To modify it, an expression should be added such that if a node is in the neighborhood of one of its real non-neighbors, the energy increases. The energy turned out to be somehow the amount of error. So (7) was modified in a way that the energy of a node is the sum of the energy for neighbors located outside the transmission range, as well as non-neighbors located inside the transmission range of the node itself. The modified energy of node S_i is

$$E(S_i)=\sum_{\text{neighbor } S_j : dist_{\text{est}}(S_i,S_j)>r}[dist_{\text{est}}(S_i,S_j)-r]^2 + \sum_{\text{non-neighbor } S_k : dist_{\text{est}}(S_i,S_k)<r}[dist_{\text{est}}(S_i,S_k)-r]^2. \quad (10)$$

Equation (8) does not necessarily have to be changed. Since the force of a non-neighbor node located inside the transmission range of the node will be negative, it will cause the node to be repulsed from that non-neighbor node. Equation (9) will also be modified such that the resultant force on a node will be the vector sum of the force caused by neighbors located outside the transmission range (attraction) and non-neighbors located inside the transmission range of the node (repulsion):

$$\overrightarrow{F(S_i)}=\sum_{\text{non-neighbor } S_k : dist_{\text{est}}(S_i,S_k)<r}\overrightarrow{f(S_i,S_k)} + \sum_{\text{neighbor } S_j : dist_{\text{est}}(S_i,S_j)>r}\overrightarrow{f(S_i,S_j)}. \quad (11)$$

In many respects, the idea of ARPoFiL is similar to that of robot navigation. If you consider a node as a robot, then non-neighbor nodes which are inside the transmission range of that node are obstacles (generating repulsive potential field), and neighbor nodes outside that node's transmission range are goal points (generating attractive potential field). In the next section, the performance of our method versus current SVM and MMSO method will be experimentally compared.

IV. Experimental Results and Simulation Study

To further show that our proposed method outperforms the current method, results were double checked by simulating both methods in dense and sparse networks. We simulated WSN on java platform using the Lib-SVM package for SVM. Note that authors in [6], [7] used RBF kernel for SVM because of its empirical effectiveness. The linear kernel was tried for SVM and as will be seen, works better than the RBF kernel. The results of the simulation study show this in comparison to the results in [6], [7].

The dense network was a 100 m × 100 m field with 1,000 sensor nodes, while a 50 m × 50 m network with 100 sensor nodes represented the sparse network in our simulation. The performance of both approaches in different populations of beacon nodes was also checked at 5%, 10%, 15%, 20%, and 25%. Furthermore, the transmission range is shown to be effective in the amount of error [6], [7]. So both 7 m and 10 m transmission ranges for the nodes were tested.

One of the challenges in real environments while deploying sensor networks are coverage holes, which cause most localization algorithms to suffer from inaccuracy. The proposed algorithm and the existing one were tested to compare their ability in modeling the coverage holes. Two configurations for the network with circle-shaped coverage holes were assumed: one with a coverage hole at the center of the field and radius $D/6$ and one with 5 coverage holes. One was at the center, the same as the previous case, and 4 at corners of the field with radius $D/12$ and a margin of $D/5$ to the nearest edges of the field. A similar configuration is assumed in [6].

Because the number of SVs and classification accuracy do not differ in SVM and probabilistic SVM, they will be omitted here. The total number of SVs in [6] is not very high, which is also true in our study. The small number of support vectors shows that there will be a small amount of computations needed at the base station as well as a small amount of information for the learnt model being transferred. On the other hand, classification accuracy as is shown in [6] increases with the beacon population. Parameter M, which is the number of classes in each dimension, was set to 128 for the dense network, and half the value, that is, 64, for the sparse network. These values were directly obtained from what the authors proposed in [6].

Mean location error, or the average displacement of the nodes from their true location, is shown in Table 1, in the dense network and 0 coverage holes under various configurations and different approaches. Also, Tables 2 and 3 show the location errors with 1 and 5 coverage holes in the dense network, respectively. As a general rule, when beacon population increases, the average location error reduces as was expected.

The other important point is that both of the methods perform better in longer transmission ranges. This is because SVM generally works better in dense networks.

Figures 2, 3, and 4 show the average location error for all of the methods in a dense network under different beacon

Table 1. Average location error (min/avg./max) for dense network with 0 coverage holes in different configurations.

(m)

Average location error		Beacon population				
Parameter settings		5%	10%	15%	20%	25%
$r = 7$ m	LSVM	0.125/4.813/26.294	0.095/3.132/15.414	0.087/2.674/13.422	0.081/2.31/11.507	0.056/2.234/10.255
	P-SVM	0.127/4.657/25.497	0.102/3.116/14.974	0.079/2.663/13.253	0.069/2.3/11.478	0.066/2.234/10.255
	LSVM-MMSO	0.075/3.054/14.52	0.05/2.132/10.915	0.056/1.88/10.862	0.057/1.701/8.738	0.061/1.65/8.362
	PSVM-ARPoFiL	0.05/2.712/15.572	0.065/1.859/9.151	0.062/1.632/9.499	0.038/1.453/7.858	0.039/1.403/6.274
$r = 10$ m	LSVM	0.16/4.62/22.469	0.071/3.171/15.332	0.085/2.481/10.969	0.078/2.269/9.305	0.072/2.087/9.214
	PSVM	0.174/4.445/20.855	0.081/3.128/15.231	0.083/2.463/10.908	0.071/2.262/8.98	0.069/2.091/9.125
	LSVM-MMSO	0.057/2.72/13.978	0.047/1.979/11.275	0.049/1.641/8.002	0.041/1.603/7.466	0.04/1.51/7.261
	PSVM-ARPoFiL	0.076/2.34/9.678	0.048/1.593/7.883	0.037/1.295/5.41	0.056/1.218/5.163	0.038/1.148/4.647

Table 2. Average location error (min/avg./max) for dense network with 1 coverage hole in different configurations.

(m)

Average location error		Beacon population				
Parameter settings		5%	10%	15%	20%	25%
$r = 7$ m	LSVM	0.131/4.619/19.17	0.095/3.512/15.586	0.07/2.868/12.78	0.053/2.464/9.453	0.077/2.302/9.815
	P-SVM	0.109/4.47/18.619	0.094/3.481/15.211	0.068/2.86/12.615	0.053/2.453/9.874	0.079/2.301/9.839
	LSVM-MMSO	0.04/3.061/13.424	0.064/2.41/11.295	0.052/2.016/9.62	0.05/1.768/7.7	0.063/1.712/8.794
	P-SVM-ARPoFiL	0.082/2.586/10.127	0.077/2.032/9.692	0.059/1.688/9.035	0.046/1.471/6.027	0.052/1.418/6.829
$r = 10$ m	LSVM	0.104/4.932/23.03	0.086/3.291/14.646	0.094/2.654/9.864	0.047/2.433/9.439	0.068/2.22/8.864
	P-SVM	0.14/4.758/21.467	0.093/3.261/13.983	0.109/2.641/9.722	0.044/2.425/9.491	0.068/2.218/8.776
	LSVM-MMSO	0.045/2.848/13.066	0.054/2.093/9.964	0.061/1.769/7.412	0.034/1.662/7.677	0.048/1.59/7.248
	P-SVM-ARPoFiL	0.053/2.299/10.034	0.051/1.581/6.153	0.045/1.306/4.821	0.041/1.234/4.662	0.039/1.17/4.689

Table 3. Average location error (min/avg./max) for dense network with 5 coverage holes in different configurations.

(m)

Average location error		Beacon population				
Parameter settings		5%	10%	15%	20%	25%
$r = 7$ m	LSVM	0.123/5.064/23.169	0.084/3.544/16.317	0.088/2.914/12.499	0.07/2.567/10.596	0.059/2.407/10.159
	P-SVM	0.096/4.907/21.913	0.084/3.524/16.269	0.087/2.904/12.525	0.068/2.566/10.64	0.059/2.403/9.596
	LSVM-MMSO	0.062/3.338/16.602	0.087/2.47/11.131	0.063/2.068/10.353	0.054/1.859/8.69	0.054/1.791/8.386
	P-SVM-ARPoFiL	0.059/2.935/14.797	0.059/2.06/9.419	0.041/1.698/8.515	0.048/1.507/7.905	0.047/1.454/7.13
$r = 10$ m	LSVM	0.118/5.277/25.706	0.117/3.345/13.378	0.082/2.829/10.77	0.092/2.484/9.666	0.072/2.301/8.732
	P-SVM	0.141/5.079/23.739	0.127/3.321/13.242	0.062/2.831/10.731	0.078/2.488/9.707	0.071/2.302/8.624
	LSVM-MMSO	0.089/3.36/18.116	0.055/2.196/9.523	0.049/1.936/7.337	0.046/1.782/7.569	0.049/1.71/7.022
	P-SVM-ARPoFiL	0.076/2.739/14.603	0.051/1.694/7.146	0.055/1.445/5.546	0.044/1.292/5.01	0.056/1.247/5.268

populations and transmission ranges with 0, 1, and 5 coverage holes, respectively. According to the figures, it is obvious that PSVM-ARPoFiL completely outperforms SVM-MMSO. However, SVM and probabilistic SVM are almost the same. We will see that PSVM works more accurately in sparse networks.

Note that when a coverage hole appears, the existing method encounters incremental rise in error, while the proposed method experiences almost no changes in error. This matter however, when 5 coverage holes are added, influences both approaches. Nevertheless, the proposed approach suffers from a smaller amount of error.

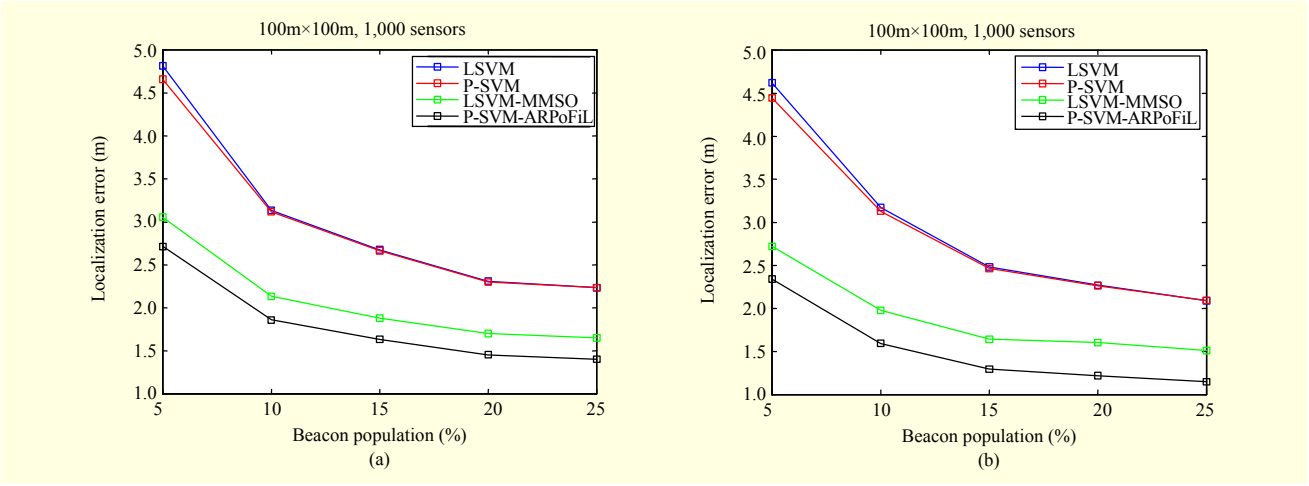

Fig. 2. Average location error for dense network with 0 coverage holes: (a) $r = 7$ m and (b) $r = 10$ m.

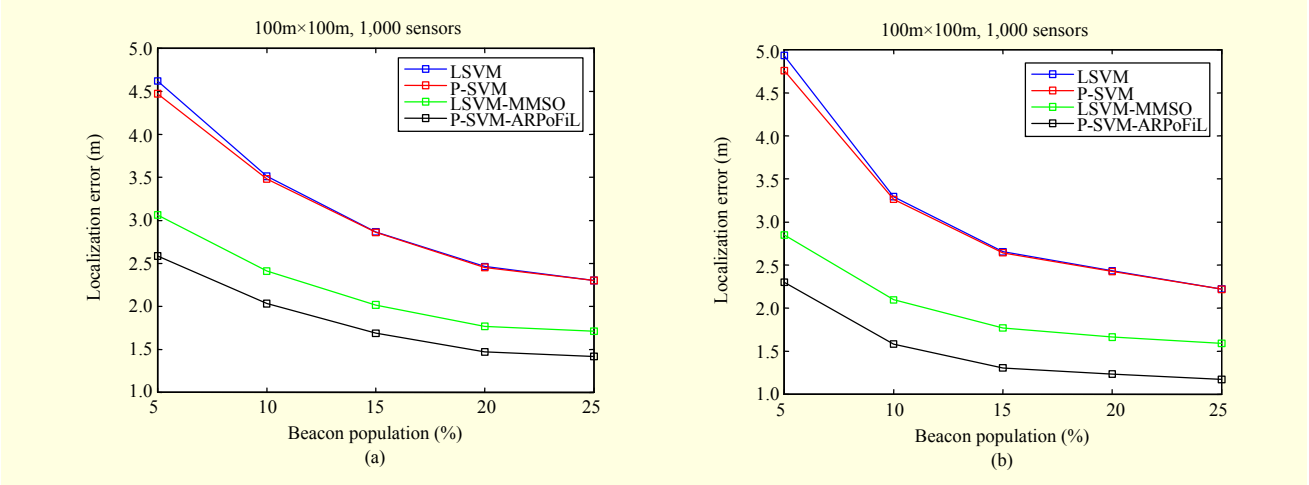

Fig. 3. Average location error for dense network with 1 coverage hole: (a) $r = 7$ m and (b) $r = 10$ m.

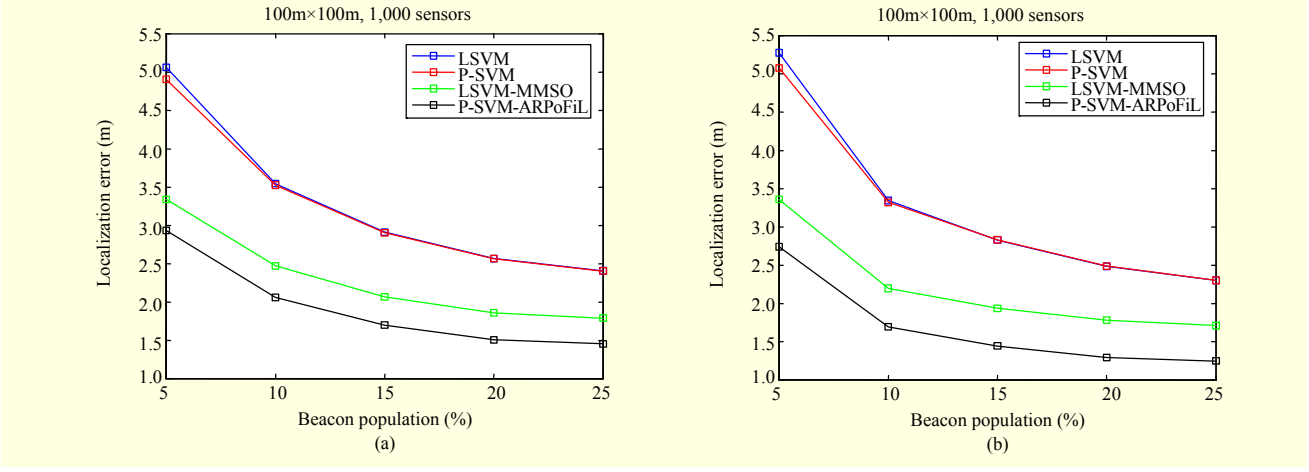

Fig. 4. Average location error for dense network with 5 coverage holes: (a) $r = 7$ m and (b) $r = 10$ m.

From the discussion above, it can be said that the proposed method works better in rough environments where there exist coverage holes. Actually, the method successfully models the coverage holes in the learning phase. This matter is of a major

Table 4. Average location error (min/avg./max) for sparse network with 0 coverage holes in different configurations.

(m)

Average location error		Beacon population				
Parameter settings		5%	10%	15%	20%	25%
$r = 7$ m	LSVM	1.112/13.671/38.175	0.928/9.452/33.118	0.552/6.308/22.634	0.512/6.022/21.632	0.535/5.089/22.076
	P-SVM	0.656/12.309/36.109	0.658/8.566/31.483	0.509/5.654/20.041	0.421/5.398/19.138	0.476/4.701/20.914
	LSVM-MMSO	0.871/12.227/36.445	0.561/8.11/27.9	0.398/5.224/19.978	0.355/5.039/19.024	0.35/3.908/13.1
	P-SVM-ARPoFiL	0.675/10.375/32.315	0.377/6.952/28.971	0.398/4.394/20.978	0.308/3.989/18.434	0.246/3.103/13.043
$r = 10$ m	LSVM	1.093/11.802/33.445	0.49/7.002/20.976	0.423/6.036/20.103	0.444/5.566/20.107	0.551/4.518/13.82
	P-SVM	0.683/10.508/29.907	0.603/5.768/18.606	0.334/5.418/18.212	0.342/4.922/18.013	0.465/4.19/12.418
	LSVM-MMSO	0.838/9.7/28.853	0.348/5.359/18.19	0.212/4.869/18.513	0.256/4.214/15.392	0.354/3.552/11.244
	P-SVM-ARPoFiL	0.483/6.922/23.711	0.255/3.239/12.181	0.199/3.059/13.292	0.194/2.772/13.145	0.238/2.306/7.74

Table 5. Average location error (min/avg./max) for sparse network with 1 coverage hole in different configurations.

(m)

Average location error		Beacon population				
Parameter settings		5%	10%	15%	20%	25%
$r = 7$ m	LSVM	0.881/13.707/40.163	0.659/10.603/30.513	0.717/6.724/21.466	0.507/5.857/24.937	0.498/5.78/21.867
	P-SVM	0.84/12.501/36.032	0.44/9.766/29.234	0.428/6.09/20.93	0.414/5.303/23.455	0.318/5.536/21.089
	LSVM-MMSO	0.754/11.921/35.211	0.479/9.493/29.427	0.468/5.488/18.304	0.238/4.814/21.009	0.501/4.757/19.554
	P-SVM-ARPoFiL	0.779/11.047/36.525	0.416/8.632/29.305	0.284/4.702/20.405	0.248/3.976/21.968	0.296/3.924/19.208
$r = 10$ m	LSVM	0.64/11.609/31.09	0.748/7.679/22.682	0.708/6.213/21.382	0.451/5.189/18.785	0.531/4.706/15.514
	P-SVM	0.61/10.276/27.905	0.523/6.221/19.402	0.467/5.143/18.93	0.524/4.626/16.661	0.469/4.327/14.82
	LSVM-MMSO	0.582/9.393/24.455	0.523/5.875/19.025	0.397/4.807/17.376	0.364/4.154/14.805	0.41/3.683/12.885
	P-SVM-ARPoFiL	0.655/6.97/24.432	0.329/3.66/13.355	0.299/2.907/12.212	0.222/2.834/13.87	0.281/2.423/11.453

Table 6. Average location error (min/avg./max) for sparse network with 5 coverage holes in different configurations.

(m)

Average location error		Beacon population				
Parameter settings		5%	10%	15%	20%	25%
$r = 7$ m	LSVM	1.402/14.963/35.887	0.674/9.377/29.186	0.534/7.013/26.749	0.56/6.178/23.084	0.367/5.787/27.44
	P-SVM	1.205/13.543/33.28	0.564/8.258/25.736	0.576/6.424/25.758	0.523/5.695/22.529	0.273/5.476/27.052
	LSVM-MMSO	0.952/13.409/34.648	0.557/8.236/28.251	0.467/5.945/21.836	0.468/5.248/20.443	0.294/4.706/22.876
	P-SVM-ARPoFiL	0.714/11.912/34.373	0.436/6.775/26.193	0.437/5.205/23.146	0.263/4.686/21.262	0.3/4.322/23.852
$r = 10$ m	LSVM	0.908/12.33/32.296	0.78/7.74/22.805	0.677/6.265/20.492	0.447/5.158/18.225	0.375/4.53/14.798
	P-SVM	0.799/10.7/29.77	0.564/6.449/20.24	0.489/5.237/18.527	0.361/4.525/16.626	0.388/4.272/15.121
	LSVM-MMSO	0.696/9.842/27.209	0.349/5.744/18.692	0.502/4.72/16.198	0.335/3.929/14.319	0.335/3.489/12.231
	P-SVM-ARPoFiL	0.682/8.165/27.386	0.339/3.894/14.83	0.329/3.105/13.04	0.268/2.588/11.613	0.24/2.478/10.08

importance to us because sensor networks are mostly deployed in rough areas that cause coverage holes to appear in the architecture of the network.

On the other hand, Tables 4, 5, and 6 show the location error for the sparse network with 0, 1, and 5 coverage holes, respectively. Note that the total error in comparison with the dense network is increased.

As previously mentioned, this increase is due to the fact that SVM generally works better in dense networks. We may consider this as a draw-back of the SVM based method. Furthermore, when the network is dense, more neighbors are available, and that means more neighborhood and non-

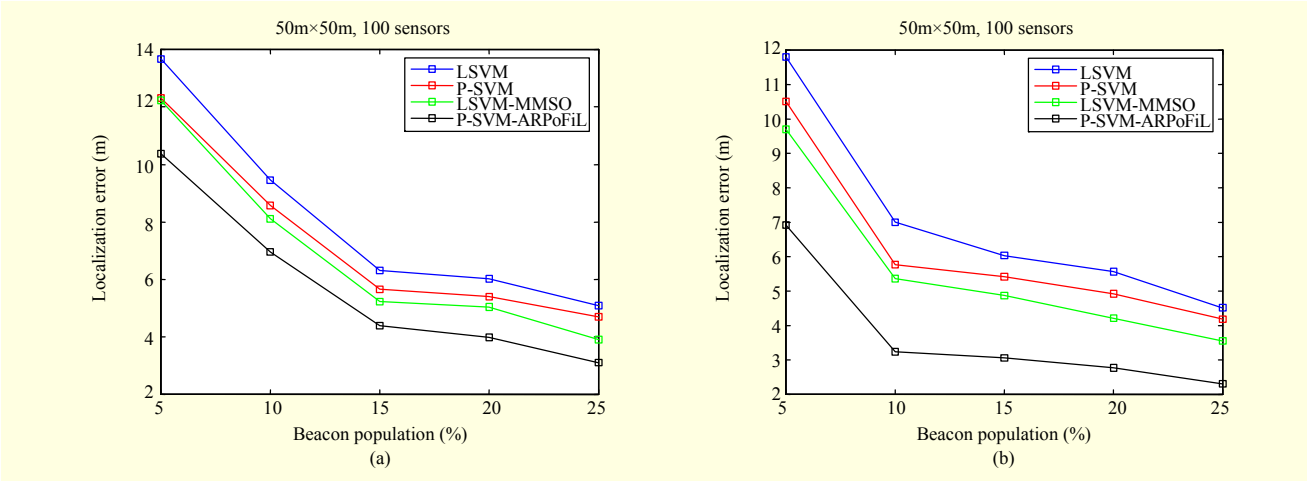

Fig. 5. Average location error for sparse network with 0 coverage holes: (a) $r = 7$ m and (b) $r = 10$ m.

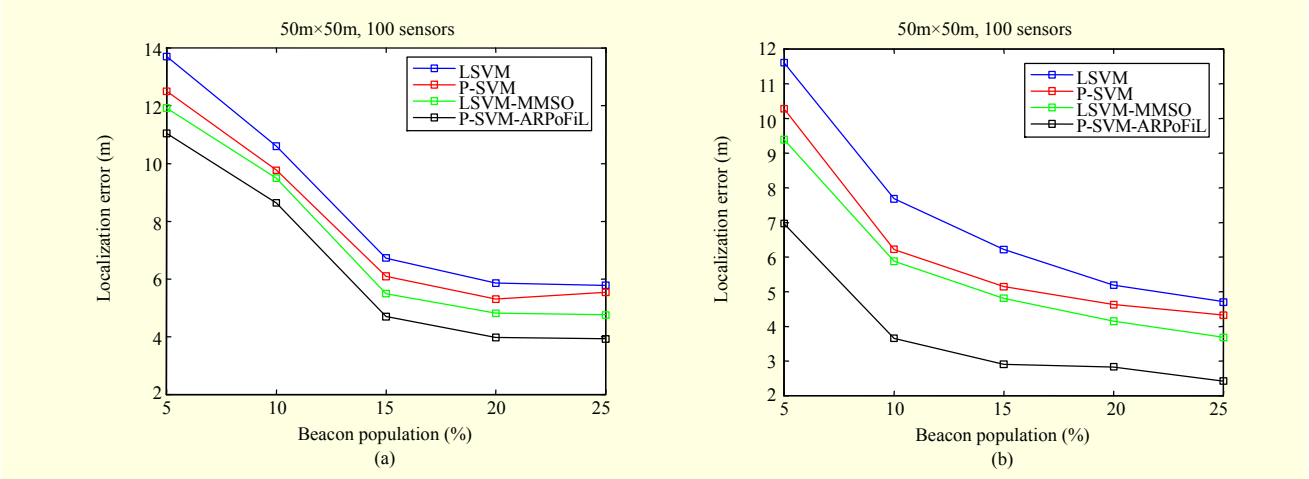

Fig. 6. Average location error for sparse network with 1 coverage hole: (a) $r = 7$ m and (b) $r = 10$ m.

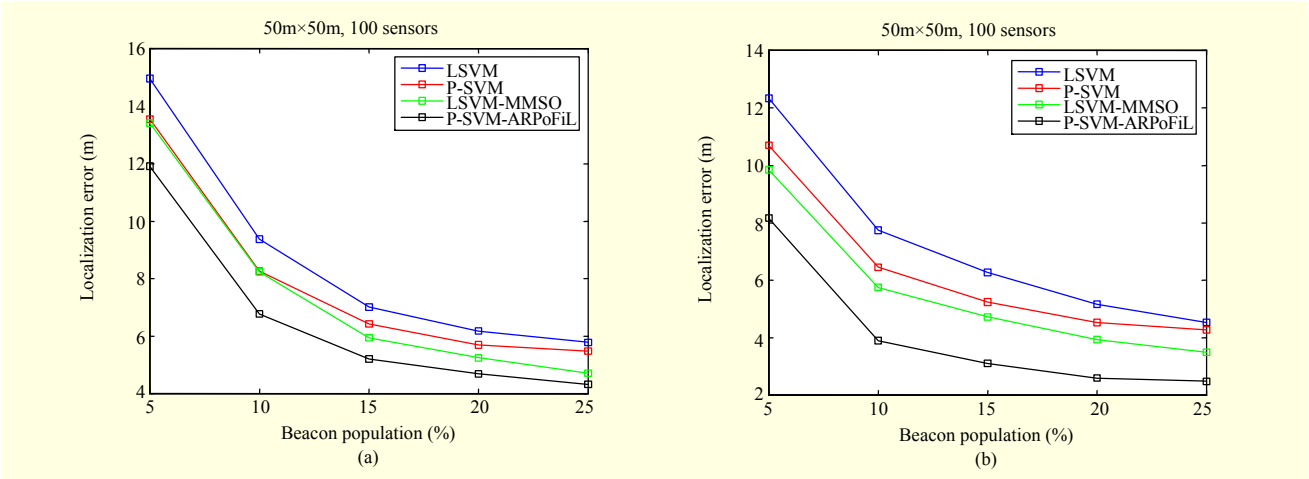

Fig. 7. Average location error for sparse network with 5 coverage holes: (a) $r = 7$ m and (b) $r = 10$ m

neighborhood information, which is why both MMSO and ARPoFiL perform better in dense networks.

Figures 5, 6, and 7 show the location error for the sparse network under different beacon populations and different

transmission ranges for all of the methods for 0, 1, and 5 coverage holes, respectively. Here, the point is that PSVM works much better than SVM, and in some cases, it is comparable to SVM-MMSO (r = 7 m, 5% beacon).

For the dense network and from Figs. 3 to 5, it can be said that increasing the beacon node population results in reduction in the mean location error. However, this error reduction is significant, up to a beacon population of 10% to 15%.

As the beacon nodes are more expensive than normal sensor nodes and impose more restrictions, a small beacon population in a localization algorithm is preferred. However, a trade-off exists between the beacon population and the amount of error. So it seems that a beacon population of 10% to 15% is adequate both in error and cost for a dense network.

On the other hand, for all of the sparse networks except the case when there are no coverage holes and r = 10 m, the above statement is true. In the case of no coverage holes and r = 10 m, it seems the adequate beacon population is about 10%.

V. Conclusion

We have shown that probabilistic SVM works more accurately than the ordinary SVM in the problem of localization in sensor networks. Furthermore, the initiated method of ARPoFiL modifies the location of the sensors with more accurate results than MMSO. Both algorithms work in a centralized manner.

One can argue that although the proposed approach works more accurately than the existing one, it has more computational costs. For example, take the SVM phase. In LSVM, each node localizes itself using a binary tree and with an order of log(M). However, in probabilistic SVM, each node has to go through all nodes of the tree, using all models learnt to localize itself. As a solution, the procedure of localization can be modified so that this computational cost incurs on the base station. Instead of sensor nodes sending their neighborhood information to the base station after the localization phase, they can send this information as well as their hop-counts to beacon nodes, before the localization phase. Then, the base station can do the math itself. This way, there is no need for the base station to broadcast the learnt models.

Another issue that impacts the computational cost is that the ARPoFiL method modifies each node's location based on more nodes (non-neighbors as well as neighbors) in comparison to MMSO. However, that should not be a real concern because these computations, as said before, are done at the base station, which we assume has an unlimited power source and is powerful enough.

The proposed method may be improved by changing the learning mechanism. An ANN instead of an SVM may improve the localization. Furthermore, classes can be defined in other ways. Currently, each dimension has its own class definitions, while the feature vector used, hop-counts to beacon nodes, is based on both directions. The localization may be improved by defining classes in a way that considers an area, for instance, coin-shaped classes, as [14], [15] defined. However, to cover the whole sensor field, many of these classes are needed, and this leads to more computational costs in the learning phase. Also, classes can be defined such that the sensor field is divided into four similar squares each time. However, this way, SVM needs to work with 4 classes, which has its own difficulties. We plan to cope with this in the future.

From this study, it can be concluded that our method is more suitable for localization than the existing one in all networks, especially sparse ones and those deployed in rough environments leading to coverage holes. Not only was the localization improved with our proposed method, our results for LSVM in comparison to results in [6] and [7], show that it was also improved with the use of linear kernels instead of RBF kernel for the SVM.

Generally, the SVM-based method is suitable when restrictions on the costs and the equipments needed to deploy the sensor network exist. In fact, the SVM-based method, and especially our proposed method, makes use of a small amount of information to localize nodes, and this is done successfully and with the minimal amount of error.

References

[1] K. Roemer and F. Mattern, "The Design Space of Wireless Sensor Networks," *IEEE Wireless Commun.*, vol. 11, no. 6, Dec. 2004, pp. 54-61.

[2] L.M. Pestana Leao de Brito and L.M. Rodríguez Peralta, "An Analysis of Localization Problems and Solutions in Wireless Sensor Networks," *Polytechnical Studies Rev.*, vol. 6, no. 9, 2008.

[3] V. Ramadurai and M.L. Sichitiu, "Localization in Wireless Sensor Networks: A Probabilistic Approach," *Proc. Int. Conf. Wireless Netw.*, Las Vegas, NV, June 2003, pp. 275-281.

[4] A.A. Kannan, G. Mao, and B. Vucetic, "Simulated Annealing Based Wireless Sensor Network Localization," *J. Comput.*, vol. 1, no. 2, 2006, pp. 15-22.

[5] C. Cortes and V. Vapnik, "Support-Vector Networks," *Mach. Learning*, vol. 20, no. 3, Sept. 1995, pp. 273-297.

[6] D.A. Tran and T. Nguyen, "Localization in Wireless Sensor Networks Based on Support Vector Machines," *IEEE Trans. Parallel Distributed Syst.*, vol. 19, no. 7, July 2008, pp. 981-994.

[7] D.A. Tran and T. Nguyen, "Support Vector Classification Strategies for Localization in Sensor Networks," *Proc. IEEE Int. Conf. Commun. Electron.*, 2006.

[8] A. Madevska-Bogdanova, D. Nikolik, and L.M.G. Curfs,

"Probabilistic SVM Outputs for Pattern Recognition Using Analytical Geometry," *Neurocomput.*, vol. 62, Dec. 2004, pp. 293-303.

[9] J.C. Platt, "Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods," *Advances in Large Margin Classifiers*, A.J. Smola et al., Eds., MIT Press, 1999, pp. 61-74.

[10] H. Lin, C. Lin, and R.C. Weng, "A Note on Platt's Probabilistic Outputs for Support Vector Machines," *Mach. Learning*, vol. 68, no. 3, Oct. 2007, pp. 267-276.

[11] N.B. Priyantha et al., "Anchor-Free Distributed Localization in Sensor Networks," *Proc. 1st Int. Conf. Embedded Netw. Sensor Syst.* LA, CA, 2003, pp. 340-341.

[12] Y. Koren and J. Borenstein, "Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation," *IEEE Int. Conf. Robot. Autom.*, vol. 2, Sacramento, CA, 1991, pp. 1398-1404.

[13] G. Luh and W. Liu, "Dynamic Mobile Robot Navigation Using Potential Field Based Immune Network,"*10th World Multi-Conf. Syst., Cybern., Inf.*, vol. 2, July 2006, pp. 246-251.

[14] X. Nguyen, M.I. Jordan, and B. Sinopoli, "A Kernel-Based Learning Approach to Ad Hoc Sensor Network Localization," *ACM Trans. Sensor Netw.*, vol. 1, no. 1, Aug. 2005, pp. 134-152.

[15] D.A. Tran, X. Nguyen, and T. Nguyen, "Learning Techniques for Localization in Wireless Sensor Networks," *Localization Algorithms and Strategies for Wireless Sensor Networks: Monitoring and Surveillance Techniques for Target Tracking*, 2008.

Reza Samadian received his BSc in software engineering from Qazvin Azad University, Iran, in 2007. In 2010, he received his MSc in artificial intelligence from Amirkabir University of Technology, Iran. He has been working as a researcher in the Real Time Intelligent Systems Laboratory, at the Department of Computer Engineering and IT, Amirkabir University of Technology. His fields of interests include machine learning, evolutionary computing, and pattern recognition.

Seyed Majid Noorhosseini received his BSc and MSc from Amirkabir University of Technology, Iran, in 1986 and 1989, respectively. He received his PhD from McGill University in Montreal, Canada, in 1994. He was a senior scientist at Nortel Networks, in Canada and the U.S during 1996 to 2005, working in different areas of network management. He holds US patent 6707795 in alarm correlation methods and systems. He is now with the Department of Computer Engineering and Information Technology at Amirkabir University of Technology. He is also serving as the deputy researcher at Iran Telecommunication Research Center (ITRC).