Impasse-Aware Node Placement Mechanism for Wireless Sensor Networks

Chih-Yung Chang, *Member, IEEE*, Yu-Ting Chin, Cheng-Chang Chen, and Chao-Tsun Chang, *Member, IEEE*

***Abstract*—In wireless sensor networks, sensor deployment is an important issue in which sensors are deployed in a specific monitoring region using low-cost hardware but achieving high coverage quality. In recent years, several mechanisms have been developed for the efficient robotic deployment of sensors. Their performance levels are highly dependent on unknown obstacles that can give rise to the dead-end problem. A key challenge when developing a robot deployment mechanism is to overcome the dead-end problem and deliver full coverage with a minimal number of sensors. This paper proposes an impasse-aware robot deployment (IAD) algorithm. The proposed IAD mainly consists of basic deployment rules and dead-end handling rules. The basic deployment rules are intended to achieve full coverage with a minimal number of sensors; the proposed dead-end handling rules can efficiently resolve the dead-end problem. Extensive experimental studies have demonstrated that our proposed IAD achieves superior performance to that of existing robot deployment mechanisms with respect to coverage ratio, energy efficiency, deployment path length, and required stack space.**

***Index Terms*—Coverage, dead-end, impasse, deployment, obstacle, robot, wireless sensor network (WSN).**

I. Introduction

WIRELESS sensor networks (WSNs) and robot control have been used in numerous applications, including in trajectory tracking [1], healthcare [2], road safety [3], environmental monitoring [4], military surveillance (i.e., demining robots) [5], adaptive control [6], and sensor deployment. The sensor deployment issue has received much attention in recent years. The proposed deployment mechanisms can be classified into random deployment, mobile sensor dispatching, and robotic deployment. Random deployment [7], [8] are simple and can be implemented easily, but they are not efficient because the sensor densities vary in different regions. A region with a large number of deployed sensors usually suffers from high overhead costs, such as hardware costs, computing costs, and communication costs. By contrast, a region with low sensor density usually contains coverage holes. To achieve full sensor coverage, a large number of sensors may be required, incurring a high hardware cost. Some related mechanisms involve analyzing the deployment of mobile sensors (i.e., mobile sensor dispatching) before deployment begins in centralized [9], [10] and distributed ways [11], [12]. First, a set of locations in a coverage hole can be calculated. The mobile sensors then cooperatively move to the locations of coverage holes. However, this incurs overhead costs such as hardware costs and the energy consumption costs of mobile sensors.

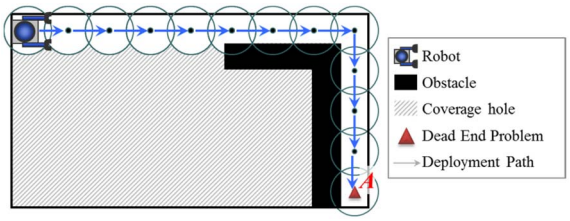

Fig. 1. Robot encounters a Dead-End problem.

Other studies have used robots to address the deployment problem. In each round, a robot moved a certain number of steps, stopped, and deployed one static sensor, thereby readily achieving full coverage using only the appropriate numbers of sensors. In addition to sensor deployment, a robot can perform other tasks such as detecting coverage holes, redeployment, and surveillance. However, diverse unknown obstacles can have deleterious effects on deployment efficiency. When a robot deploys sensors, it may become trapped by obstacles, boundaries, or deployed sensors, leading to a situation in which the robot has no available direction for further movement; this is called the *Dead-End* problem. The robot depicted in Fig. 1 encounters the *Dead-End* problem when it moves to location *A*. In this scenario, the robot is surrounded by obstacles or deployed sensors and it is unable to advance. Consequently, the deployment process is terminated, leading to the existence of coverage holes, which are marked in gray.

To overcome the *Dead-End* problem, in [13], a robot deployment mechanism, called LRV, was presented. When the robot visited a deployed sensor, the sensor broadcast the least recently visited direction with the aim of guiding the robot to explore the region to be monitored. However, unpredicted

Manuscript received April 7, 2016; revised October 25, 2016 and January 19, 2017; accepted January 23, 2017. Date of publication February 13, 2017; date of current version July 17, 2018. This work was supported by the Ministry of Science and Technology of the Republic of China under Contract MOST 103-2221-E-032-020-MY3 and Contract MOST 105-2218-E-007-005. This paper was recommended by Associate Editor W. Shen. *(Corresponding author: Chih-Yung Chang.)*

C.-Y. Chang, Y.-T. Chin, and C.-C. Chen are with the Department of Computer Science and Information Engineering, Tamkang University, New Taipei City 25137, Taiwan (e-mail: cychang@mail.tku.edu.tw; 143221@mail.tku.edu.tw; ccchen@mail.tku.edu.tw).

C.-T. Chang is with the Department of Information Management, Hsiuping University of Science and Technology, Taichung 41280, Taiwan (e-mail: cctas@mail.hust.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2017.2659803

TABLE I
COMPARISON OF THE MAIN CHARACTERISTICS OF THE PROPOSED IAD WITH EXISTING RELATED WORKS

Studies	Static Sensor	Mobile Sensor	Mobile Robot	Deployed Density	Full Coverage	# of Sensors	Obstacle Consideration	Obstacle Information	Dead-End Consideration	Path Efficiency
[7-8]	Used	No	No	Nonuniform	No	Large	No	N/A	N/A	N/A
[9-12]	No	Used	No	Nonuniform	No	Medium	No	N/A	N/A	N/A
[13]	Used	No	Used	Nonuniform	No	Medium	Yes	Unknown	No	Lower
[14-15]	Used	No	Used	Uniform	No	Fewer	Yes	Unknown	No	Medium
[16-17]	Used	No	Used	Uniform	No	Fewer	Yes	Unknown	No	Medium
[18]	Used	No	Used	Uniform	Yes	Fewer	Yes	Unknown	Yes	Lower
IAD	Used	No	Used	Uniform	Yes	Fewer	Yes	Unknown	Yes	Higher

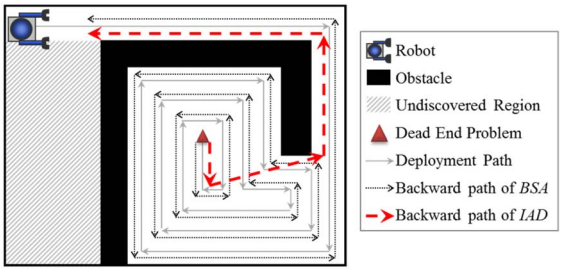

Fig. 2. Two backward paths for a *Dead-End* problem, as produced by IAD and BSA mechanisms.

obstacles severely degraded the deployment efficiency of LRV. In [14] and [15], the robot applies right-hand policy for reducing the impact of obstacles. However, the trajectory path of the robot is inefficient since the robot might move a long path, but did not deploy any sensor.

To reduce the inefficiency caused by obstacles, and to improve deployment efficiency, the efficient obstacle-resistant robot deployment (ORRD) [16] and obstacle-free and power-efficient deployment algorithm (OFPD) [17] mechanisms were proposed. However, those mechanisms left coverage holes in some monitoring areas that contained obstacles with certain shapes. When the ORRD and OFPD algorithms halted, they left coverage holes.

A robot deployment mechanism [18] called backtracking spiral algorithm (BSA) employs a last in first out stack to record all visited locations and turns back to an unexplored location whenever it encounters a *Dead-End* problem. As shown in Fig. 2, BSA constructs a long return path, which is marked with a black dotted line. In addition, BSA needs a large stack. By contrast, the proposed impasse-aware robot deployment (IAD) mechanism records very few locations. As a result, the robot can return to an unexplored region along an efficient path, which is marked with a red dotted line in Fig. 2.

This paper presents an IAD algorithm which consists of basic deployment rules and *Dead-End* handling rules. The basic deployment rules are composed of *Moving Direction* and *Horizontal and Vertical Movement* rules in a 2-D simulated environment. Guided by these rules, the robot can deploy a minimal number of sensors but achieve full coverage. The *Dead-End* handling rules are composed of *Normal*, *Narrow-Lane*, *Dead-End*, and *GoBack* rules which assist the robot to overcome the *Dead-End* problem efficiently.

Table I summarizes the characteristics of previous related studies and the proposed IAD. The proposed IAD additionally considers the *Dead-End* problem and improves the path efficiency.

The rest of this paper is organized as follows. Section II describes the network environment and problem statement of the robot deployment. Section III presents the basic deployment rules, which are mainly applied in environments without obstacles. Section IV presents the *Dead-End* handling rules which aim to solve the problems of unpredicted obstacles. Section V analyzes and compares the deployment efficiency levels of the proposed IAD algorithm and the earlier BSA algorithm in terms of the length of traversed paths. Section VI presents the physical implementation of the proposed IAD. In Section VII, extensive simulation experiments are presented to compare the performance levels of the proposed IAD and of earlier methods. Section VIII presents the conclusion.

II. NETWORK ENVIRONMENT

A. System Model

In this paper, it was assumed that the robot carries a set of static sensors and is aware of the boundaries of the region to be monitored. These sensors can be deployed in the simulated region to be monitored, M, which is modeled by an arbitrary polygon on a 2-D plane. The obstacles that exist inside M are also modeled as polygons and do not partition M. The robot is equipped with GPS, and an E-compass sensor to provide its current location information and moving direction information, respectively. Moreover, the robot is also equipped with an ultrasonic sensor and radio system for detecting obstacles and communicating with deployed sensors wirelessly, respectively. The communication range of a static sensor is r_c which is at least $\sqrt{3}$ times its sensing range r_s. This guarantees that full coverage can be achieved with a minimal number of sensors [16], [17], [19]. The robot is able to calculate its moving distance by using GPS information, localization mechanisms [20], [21], and the rotation speed of its wheels.

B. Problem Formulation

This paper aimed to investigate the robot deployment problem in environments containing unknown obstacles. The robot moves and deploys sensors, aiming to minimize its path length

and to cover the region to be monitored fully with deployed sensors, despite the region to be monitored containing unpredicted obstacles. The proposed IAD algorithm copes with the *Dead-End* problem and guides the robot back to the unexplored region along an efficient path, reducing the time and energy required for robot deployment

$$I_s = \frac{\bigcup_{1\le i\le n} \mathrm{cov}(s_i)}{|M| - \bigcup_{1\le j\le m} \mathrm{cov}(o_j)}. \quad (1)$$

Let $S = \{s_1, s_2, \ldots, s_n\}$ denote the set of sensors sequentially deployed by the robot, where n denotes the number of deployed sensors. Let $O = \{o_1, o_2, \ldots, o_m\}$ denote the set of obstacles in the region to be monitored, M. When applying the deployment algorithm in the presence of obstacles, the robot might encounter the *Dead-End* problem because of the obstacles. Assume the robot encounters *Dead-End* problems at $|D|$ locations, denoted by $D = \{l_1^{\mathrm{DeadEnd}}, l_2^{\mathrm{DeadEnd}}, \ldots, l_{|D|}^{\mathrm{DeadEnd}}\}$, where l_i^{DeadEnd} represents the location of the ith *Dead-End* encountered by the robot. When the robot encounters a *Dead-End* problem, the robot should move back to the unexplored region without deploying any sensors. Let $P_{\mathrm{Deployment}}$ and $P_{\mathrm{Backtracking}}$ denote the length of the robot's forward path, and the length traversed when it turns back to the nearest location adjacent to an unexplored region. Because any coverage hole would reduce the surveillance quality of the WSN, the deployment algorithm strives to maximize the size of the sensing area covered by the deployed sensors. Let the notation $|M|$ denote the size of the relevant region to be monitored M. Let $\mathrm{cov}(s_i)$ and $\mathrm{cov}(o_j)$ denote the coverage areas of the sensor s_i and obstacle o_j, respectively. The surveillance index I_s given by (1) is used to estimate the degree of surveillance. The surveillance index I_s is defined as the ratio of the area covered by the deployed sensors to the area of the feasible deployment region.

In addition, let l_i^{robot} denote the location of the robot when the robot stops and deploys sensors$_i$. Let $T = \{t_1, t_2, \ldots, t_{|T|}\}$ denote a set of points in time when the robot executes the deployment task.

The following presents the objective function and constraints intended to maximize the efficiency of the proposed robot deployment algorithm:

Objective Function

$$\textbf{Maximize } I_s \text{ with minimal}\left(P_{\mathrm{Deployment}} + P_{\mathrm{Backtracking}}\right) \quad (2)$$

Subject to:

$$\mathrm{dist}(s_i, s_j) = \sqrt{3}r_s, \quad \forall s_i, s_j \in S,\ s_j \text{ neighboring to } s_i \quad (3)$$

$$\sum_{l_j \in M} w_{i,j} = 1, \quad \forall s_i \in S, \text{ where}$$

$$w_{i,j} = \begin{cases} 1, & \text{if } s_i \text{ is located at location } l_j \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

$$R_k^{\mathrm{deploy}} + R_k^{\mathrm{move}} \le 1, \quad \forall t_k \in T, \text{ where}$$

$$R_k^{\mathrm{deploy}} = \begin{cases} 1, & \text{if the robot stops and} \\ & \quad \text{deploys sensor at time } t_k \text{ and} \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

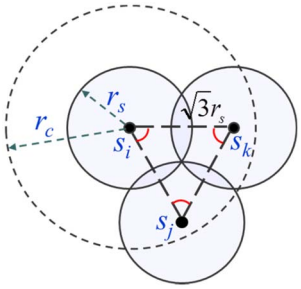

Fig. 3. Optimal sensor deployment.

$$R_k^{\mathrm{move}} = \begin{cases} 1, & \text{if the robot is moving at time } t_k \\ 0, & \text{otherwise} \end{cases}$$

$$l_i^{\mathrm{robot}} \ne l_j^{\mathrm{robot}}, \quad \forall j \ne i. \quad (6)$$

The efficiency criterion of a robot deployment algorithm is specified in (2), namely, to minimize the robot's path (which tends to minimize the deployment time and energy consumption of the robot) and to maximize the degree of surveillance I_s.

Constraint (3) is intended to limit the hardware cost of deployed sensors. To minimize the number of deployed sensors, two neighboring sensors should have a minimal overlapped sensing region under the constraint of full coverage. Assume that sensors s_i, s_j, and s_k neighbor each other. As illustrated in Fig. 3, previous studies [16], [17], [19] have proven that the optimal deployment can be reached if the coverage areas of three sensors intersect exactly at one point. That is, the Euclidean distance of any pair of s_i, s_j, and s_k is equal to $\sqrt{3}r_s$, where r_s denotes the sensing range of each sensor.

Equation (4) stipulates that each sensor can be deployed to at most one location. Equation (5) identifies the stop-deploy-move model. That is, the robot only has two states: deployment and moving states. In the deployment state, the robot stops and then deploys one sensor. In the moving state, the robot moves to the next target location. Equation (6) indicates that the robot can exactly deploy one sensor at each stop.

The next section presents the proposed IAD algorithm intended to optimize objective function (2) while satisfying constraints (3)–(6).

III. Optimal Deployment Mechanism Without Considering Obstacles

To simplify the deployment problem, this section assumes a simplified environment in which no *Dead-End* problem occurs during the deployment process. The *Dead-End* problem is considered in the next section. This section presents two major rule sets: the *Moving Direction Rule* and the *Horizontal and Vertical Movement Rules*. The *Moving Direction Rule* helps the robot determine the direction of its movement, whereas the *Horizontal and Vertical Movement Rules* determine the locations of sensor deployment on a 2-D plane.

A. Moving Direction Rule

The deployment task to enable monitoring of a given region consists of a series of basic operations: moving one step and

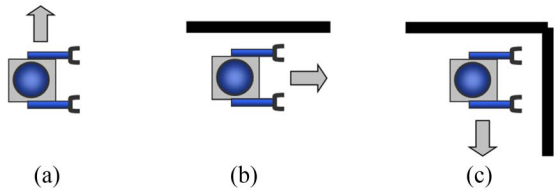

Fig. 4. Priorities of movement directions. (a) Priority 1: left. (b) Priority 2: straight. (c) Priority 3: right.

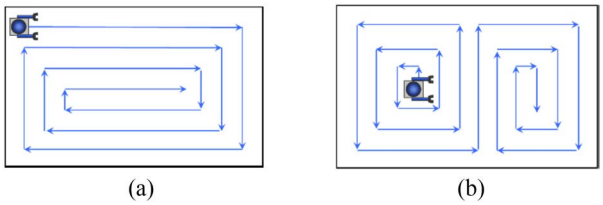

Fig. 5. Examples of deployment trajectories generated by applying the proposed *Moving Direction Rule*. (a) Robot moves along a spiral path, from outside to inside, if it is initially located at the left-top corner. (b) Movement trajectory of a robot that is not initially located at the boundary region.

then placing a sensor at the current location. Conceptually, a spiral movement scheme can be applied to greedily reduce the unexplored portion of the region to be monitored until the robot has deployed enough sensors to implement full coverage. The robot maintains an important valuable, called *direction*, whose value determines the direction of the robot's next movement. More specifically, the values of *direction* might be *left*, *straight*, or *right*, which represent that the movement instructions "turn left," "go straight," or "turn right," respectively. The following *Moving Direction Rule* mainly implements the robot's spiral movement.

Moving Direction Rule: The priority of *direction* for the robot's movement is *left*, *straight*, and then *right*.

Herein, the robot treats the boundaries of the region to be monitored and the sensing regions of the deployed sensors as virtual obstacles. Fig. 4 provides the priorities of the movement directions. As shown in Fig. 4(a), if no obstacle exists at the left side of the robot, the robot turns left with the highest priority. Otherwise, the robot goes straight if no obstacle exists in front of the robot, as shown in Fig. 4(b). If the aforementioned two rules fail, the robot turns right, as shown in Fig. 4(c).

The following discusses two cases in which the *Moving Direction Rule* is applied. In the first case, the robot is initially located at the top-left corner of the region to be monitored; the deployment trajectory is as shown in Fig. 5(a). In the second case, the robot is not initially located at the boundary of the region to be monitored; it moves in a counterclockwise manner, from inside to outside, until it encounters the boundary. After that, it moves in a clockwise manner, from outside to inside, as shown in Fig. 5(b). Consequently, the robot reduces the unexplored region step by step until the region to be monitored has been completely traversed by the robot.

B. Horizontal and Vertical Movement Rules

For satisfying constraints (3)–(6), the basic concept of the *Horizontal and Vertical Movement Rules* is that the robot deploys sensors at intervals of $\sqrt{3}r_s$.

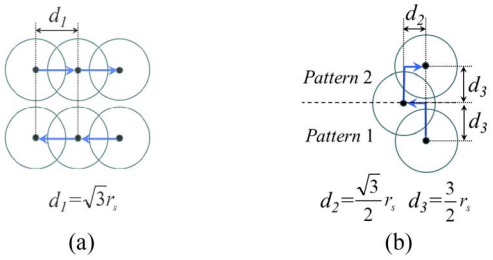

Fig. 6. Horizontal and vertical movement rules. Movement of (a) *H*-mode and (b) *V*-mode consists of two patterns.

To deploy the sensors at the optimal locations, the robot applies either of two movement modes, namely the horizontal-mode (*H*-mode) or vertical-mode (*V*-mode). If the robot stays in *H*-mode, it moves toward and deploys sensors separated by distances of $\sqrt{3}r_s$ as shown in Fig. 6(a). By contrast, if the robot stays in *V*-mode, its movement consists of two patterns. As shown in Fig. 6(b), the first pattern is to move straight for a distance of $(3/2)r_s$, turn left, and then move for a distance of $(\sqrt{3}/2)r_s$, whereas the second pattern is similar to the first except that the robot turns right. Staying in *V*-mode, the robot alternately applies the two patterns so that the optimized deployment can be achieved. The following presents the proposed *Horizontal and Vertical Movement Rules* which implement the optimal placement.

Horizontal and Vertical Movement Rules: The robot initially moves along the horizontal direction and initially stays in *H*-mode. The robot stays in *H*- or *V*-mode if it moves along horizontal or vertical directions, respectively.

By applying the *Moving Direction* and *Horizontal and Vertical Movement Rules*, the robot can achieve full coverage by using a minimal number of deployed sensors.

IV. Impasse-Aware Deployment Algorithm

This section considers obstacles to overcome the *Dead-End* problem efficiently. The following first presents the design of the proposed IAD algorithm. The procedures of the IAD algorithm are then proposed.

A. Detailed Design of the IAD Algorithm

To deal with the *Dead-End* problem, the robot should consider a variety of situations which can be classified into four states, namely the *Normal state*, *Narrow-Lane state*, *Dead-End state*, and *GoBack state*. Let the notation *AvbDir* represent the number of available directions along which the robot can move one step without encountering any obstacle. The robot is said to be in a state of *Normal*, *Narrow-Lane*, or *Dead-End* if the values of *AvbDir* are 2^+, 1, and 0, respectively, where 2^+ denotes a value greater than or equal to 2.

1) Normal State: The robot stays in the *Normal state* if it has at least two available directions for further movement. In this state, the robot only implements spiral movement and deploys sensors according to the *Moving Direction Rule* and *Horizontal and Vertical Movement Rules*. If the value of *AvbDir* changes to 0 or 1, the robot switches to *Dead-End* or *Narrow-Lane states*, respectively.

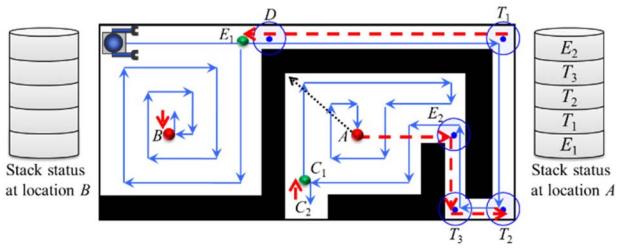

Fig. 7. Example of a robot executing the proposed IAD algorithm.

Fig. 7 provides an example to illustrate the operations executed by robot in the *Normal state*. The vertical lines in the figure indicate movement in *V*-mode; the detailed turns and motions to deploy sensors would be difficult to depict. To facilitate the presentation, the following examples use straight vertical lines to represent *V*-mode movements. As shown in Fig. 7, the robot is initially located at the top-left corner of the region to be monitored and the value of *AvbDir* is initially 2. Therefore, the robot stays in the *Normal state* and goes straight until it arrives at location *D*. At this moment, the robot has only one available direction for the next movement (the value of *AvbDir* changes from 2 to 1); therefore, the robot switches to the *Narrow-Lane state*. When the robot moves from location C_1 to C_2, it has no available direction for its next movement (the value of *AvbDir* changes from 2 to 0); thus, the robot switches to the *Dead-End state*.

2) Narrow-Lane State: The robot stays in the *Narrow-Lane state* if it has only one available direction for its next movement. The *entrance*, *exit*, and *turn points* are important locations that can guide the robot to leave the dead-end state. The robot records the locations of these points in its stack when it stays in the *Narrow-Lane* state.

Definitions (Entrance, Exit, and Turn Points): The *entrance* point of the *Narrow-Lane* is defined as the previous location of the robot at which the robot switches from the *Normal state* to the *Narrow-Lane state*. The location of the robot at which it switches from the *Narrow-Lane state* to the *Normal state* is defined as the *exit* point of the *Narrow-Lane*. A point at which the robot changes direction (i.e., turns) but stays in the *Narrow-Lane state* of a particular region is defined as a *turn* point of the *Narrow-Lane*.

Whenever the robot visits the *entrance* point of a *Narrow-Lane*, it assigns a new ID known as "*nid*" to that particular *Narrow-Lane*. As a result of *Dead-End* problems in the *Narrow-Lane state*, the stack is required to maintain some visited locations for reversing the robot so that it can eventually return to the unexplored regions. In the *Narrow-Lane state*, the robot should push the locations of the *entrance*, *exit*, and *turn* points onto the stack. Each entry of the stack consists of three fields: {*nid*, *type*, (x, y)}, where (x, y) denotes the corresponding location of the robot and *type* stores the point type, which has possible values of *entrance*, *exit*, or *turn*. In the following section, we present the *Narrow-Lane Rule,* which is executed by the robot when it stays in the *Narrow-Lane state*.

Narrow-Lane Rule: If the robot encounters any point of a *Narrow-Lane* like *entrance*, *exit,* or *turn* point then the robot pushes the information of {*nid*, *type*, (x, y)} onto the stack. The robot switches to *Dead-End* or *Normal states* if the value of *AvbDir* changes to 0 or 2^+. Otherwise, the robot remains in *Narrow-Lane state*.

Fig. 7 provides an example to illustrate the operations executed by the robot in the *Narrow-Lane state*. As shown in Fig. 7, when the robot reaches the point *D*, where the value of *AvbDir* changes from 2 to 1, it pushes the preceding location E_1, which is an *entrance* point, onto the stack and then stays in the *Narrow-Lane state*. Afterward, the robot makes turns at T_1, T_2, and T_3, which are *Narrow-Lane* turn points. It pushes the corresponding location information of T_1, T_2, and T_3 onto the stack. Lastly, when the robot arrives at location E_2, where the value of *AvbDir* changes from 1 to 2, it pushes the current location information onto the stack because the location is an *exit* point of the *Narrow-Lane*. The robot then switches to the *Normal state*.

The following explains why the robot must record the *entrance*, *exit*, and *turn* points of a *Narrow-Lane.* Suppose that the robot only recorded the *entrance* point of the *Narrow-Lane*, and did not record the *turn* and the *exit* points. When the robot arrived at location *A*, it would encounter the *Dead-End* problem. It would pop the *entrance* point from the stack and then would try to move directly from location *A* to the *entrance* point of the *Narrow-Lane*, to visit the unexplored location. However, the robot would be blocked by the obstacle. In Fig. 7, the blocked path is marked by the thin black dotted line. To prevent the robot from encountering obstacles when it goes back to the *entrance* point, it should record the location information of the *entrance*, *exit*, and *turn* points of its *Narrow-Lane*. Then the robot can successfully go back to the *entrance* point along the shortest path, which is marked by the thick red dotted line in Fig. 7.

3) Dead-End State: The robot stays in the *Dead-End state* if it has no available direction for its next movement. In the *Dead-End state*, the robot must perform two processes: 1) the *Pushing Stack Process* and 2) *Termination Check and State Switch Process*. In the *Pushing Stack Process*, the robot checks whether it should push the preceding location to the stack. Note that the value of *AvbDir* might change from 1 to 0 or from 2^+ to 0 when the robot stays in the *Dead-End state*. In the case that the value of *AvbDir* changes from 1 to 0, the robot does nothing and then tries to perform the next process, as described in the following sections. Otherwise, the value of *AvbDir* might change from 2^+ to 0. This indicates that the robot changes from the *Normal state* to the *Dead-End state* directly. Fig. 7 provides an example of this special case. As shown in Fig. 7, the values of *AvbDir* are 2 and 0 at locations C_1 and C_2, respectively. When the robot arrives at location C_2, which is the *Dead-End* point, the robot has no instruction to move back to location C_1 for further deployment of sensors in the unexplored region. Therefore, in this case, the robot should push the preceding location C_1 to the stack if the value of *AvbDir* changes from 2^+ to 0 at location C_2. The *Pushing Stack Process* is then complete and the *Termination Check and State Switch Process* starts.

In the *Termination Check and State Switch Process*, the robot should verify whether it has completed the deployment task. If the stack is nonempty, the region to be monitored still has some unexplored locations. Thus, the robot should switch to the *GoBack state*, aiming to find an unexplored location for further deploying sensors. In the *GoBack state*, the robot can pop a target location from the stack and go straight to that location for discovering the unexplored region. Otherwise, the stack is empty and the deployment task is finished. Fig. 7 provides an example in which the robot enters the *Dead-End state* and checks the stack to determine whether the deployment task is finished. When the robot encounters the *Dead-End state* at location A, the stack is nonempty, which means that the deployment task is not yet finished. As a result, the robot switches to the *GoBack state*, and leaves the *Dead-End state*. By contrast, when the robot enters the *Dead-End state* at location B, the robot can prove that the deployment task is finished because the stack is empty.

The following formally presents a *Dead-End Rule* that is executed by the robot when it is in the *Dead-End state*.

Dead-End Rule: The robot first executes the *Pushing Stack Process*. That is, if the value of *AvbDir* changes from 2^+ to 0, the robot pushes the preceding location to the stack. Otherwise, the robot does nothing. After executing the *Pushing Stack Process*, the robot additionally executes the *Termination Check and State Switch Process*. In this process, the robot checks the stack status. If the stack is empty, the robot terminates the deployment task. Otherwise, the robot switches to the *GoBack state*.

4) GoBack State: In the *GoBack state*, the robot tries to find an unexplored region. In what follows, we present the *GoBack Rule,* which is executed by the robot when it stays in the *GoBack state*.

GoBack Rule: The robot pops a location from the stack and moves straight to that location without deploying any sensors. Afterward, the robot will reset its state according to the *AvbDir* value.

Fig. 7 provides an example of the robot staying in the *GoBack state* until it reaches the *entrance* point of the *Narrow-Lane*. When the robot encounters the *Dead-End state* at location A, it pops location E_2 (the *exit* point of the *Narrow-Lane*) from the stack and moves straight to that location. Because the value of *AvbDir* at location E_2 is 0, the robot switches its status to the *Dead-End state* and then checks the stack status. Because the stack is nonempty, the robot switches its status from the *Dead-End state* to the *GoBack state*. Henceforth, it applies the *GoBack Rule* by popping location T_3 (the *turn* point of the *Narrow-Lane*) and then moves straight to that location. Similarly, the robot pops locations T_2, T_1, and E_1 from the stack and moves straight to the most recently popped location after each pop. Because the *AvbDir* value is 2 at location E_1, the robot switches to the *Normal state*. The trajectory of the robot in the *GoBack state* is marked by the thick red dotted line in Fig. 7.

B. Algorithm of IAD

The following presents the proposed IAD algorithm, which is composed of the *Normal*, *Narrow-Lane*, *Dead-End*, and *GoBack States*. Steps 4–15 present the situation in which the robot stays in the *Normal state* and applies the *Moving Direction Rule* and *Horizontal and Vertical Movement Rules*, aiming to achieve the optimal deployment. Steps 16–33 present the situation in which the robot stays in the *Narrow-Lane state* and pushes the locations of the *entrance*, *exit*, or *turn* points onto the stack. Steps 34–43 present the situation in which the robot stays in the *Dead-End state.* In the *Dead-End state*, steps 37–42 and 43 present the operations designed for the *Pushing Stack Process* and *Termination Check and State Switch Process*, respectively. Steps 44–51 present the situation in which the robot stays in the *GoBack state,* pops a location from the stack and then moves straight to the location, aiming to reach an unexplored location. As shown in step 3, when $AvbDir = 0$ and the stack is empty, the deployment process is finished.

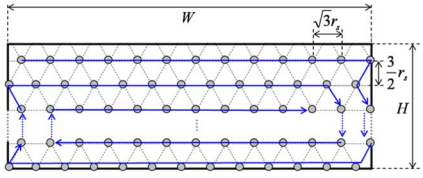

Fig. 8. Examples of WSN deployed by applying the IAD and BSA algorithms.

V. Analysis of Deployment Trajectories

This section analyzes and compares the deployment efficiency of the IAD and the BSA algorithms, in terms of the length of traversed path. Let W and H, respectively, denote the length and width of the given region to be monitored, M. Fig. 8 depicts the resultant WSN deployed by applying the proposed IAD and the existing BSA algorithms. Let notation $n_{\max}$ denote the total number of sensor nodes required for optimal deployment in region M. Equation (7) provides the derivation of $n_{\max}$

$$n_{\max} = \left(\left\lfloor \frac{W}{\sqrt{3}r_s} \right\rfloor + 1\right) \times \left(\left\lfloor \frac{H}{3/2r_s} \right\rfloor + 1\right) \tag{7}$$

$$|P| = (n_{\max} - 1) \times \sqrt{3}r_s. \tag{8}$$

The traveling path length $|P|$ can be derived by (8) because the path should pass through each sensor and the distance between two neighboring sensors is $\sqrt{3}r_s$. Note that the trajectory resulting from the *Vertical Movement Rule* contains many turns, raising unimportant details and diverting the discussion from the proposed algorithm. To facilitate the discussion, the following analyses use straight lines between two successively deployed sensors to represent the robot's movements.

The following discussion considers an environment that contains obstacles. We assume that k obstacles exist in region M. The obstacles constitute the major factor that degrades the efficiency of the deployment. For any obstacle with an irregular shape, we can construct a corresponding regular obstacle such that the impacts of the two obstacles on the traversed path length are equivalent.

Fig. 9(a) and (b) gives examples of an irregular obstacle and a corresponding regular obstacle. This motivates us to

Algorithm 1 Impasse-Aware Deployment Algorithm

Input: A robot in a given monitoring region which may contain obstacles. The robot carries static sensors for executing deployment task.
Output: A region is fully covered by the deployed sensors.
Initialization: Create an empty stack K consisting of three fields: {*nid*, *type*, *location*}

```
deploy the first sensor node
update AvbDir
while AvbDir > 0 or stack ≠ empty do
//Normal State
   If AvbDir = 2 then
      state←Normal
      If PreAvbDir = 1 then
         type←exit
         location←current location
         push {nid, type, location} onto the stack.
      end if
      execute Moving Direction Rule
      execute Horizontal and Vertical Movement Rules
      update AvbDir
   end if
//Narrow-Lane State
   If AvbDir = 1 then
      state←Narrow-Lane
      If PreAvbDir = 2 or PreAvbDir = 0 then
         nid←generate a new ID
         type←entrance
         location←preceding location
         push {nid, type, location} onto the stack.
      end if
      If PreAvbDir = 1 and direction is changed then
         type←turn
         location←current location
         push {nid, type, location} onto the stack.
      end if
      execute Moving Direction Rule
      execute Horizontal and Vertical Movement Rules
      update AvbDir
   end if
//Dead-End State
   If AvbDir = 0 and then
      state = Dead-End
      If PreAvbDir = 2 then
         nid← null
         type←null
         location←preceding location
         push {nid, type, location} onto the stack.
      end if
      If stack ≠ empty then
//GoBack State
         state = GoBack
         do
         j=The number of elements in the stack
         pop out a element K[j] from the stack
         move straightly to K[j].location
         update AvbDir
      end if
   end if
end while
```

concentrate our attention on the regular-obstacle environment. Consider the environment that contains m regular obstacles o_i with sizes $L_i \times \alpha_i L_i, 1 \le i \le m$, as shown in Fig. 9(b). Without loss of generality, we assume that the thickness of the obstacle is $\sqrt{3}r_s$. As a result, the number of sensors covered by these

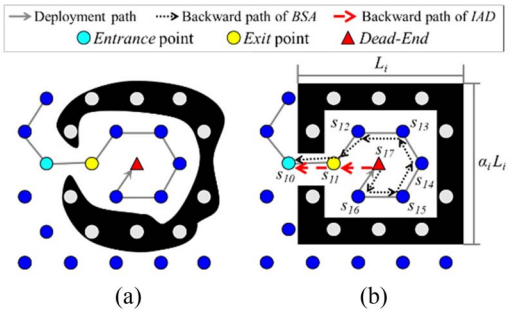

Fig. 9. To simplify the deployment efficiency analysis, each obstacle is treated as a rectangular region. (a) Irregular obstacle. (b) Corresponding regular obstacle.

obstacles can be evaluated by (9), if we put these obstacles in the WSN shown in Fig. 8

$$\sum_{i=1}^{m}\left(2\times\left(\frac{L_i}{\sqrt{3}r_s}+\frac{\alpha_i L_i}{3/2r_s}\right)\right). \tag{9}$$

Equation (10) evaluates the number of sensors deployed by applying both IAD and BSA algorithms because no sensors should be deployed inside the obstacles

$$n_{\text{req}} = n_{\max} - \sum_{i=1}^{m}\left(2\times\left(\frac{L_i}{\sqrt{3}r_s}+\frac{\alpha_i L_i}{3/2r_s}\right)\right). \tag{10}$$

The BSA mechanism overcomes the *Dead-End* problem by pushing all visited locations onto the stack. If the robot encounters a *Dead-End* problem, it turns back to the *entrance* point along the original path, as marked by the thin black dotted line in Fig. 9(b). Thus, the robot must return to the *entrance* point by revisiting all the sensors deployed on the path. Equation (11) evaluates the total length of backward paths $P_{\text{BSA}}^{\text{Backward}}$ for the BSA algorithm, where $\sqrt{3}r_s$ is the distance between two neighboring sensors

$$\left|P_{\text{BSA}}^{\text{Backward}}\right| = \sum_{i=1}^{m}\left(\frac{L_i}{\sqrt{3}r_s}\times\frac{\alpha_i L_i}{3/2r_s}\right)\times\sqrt{3}r_s. \tag{11}$$

Consequently, the total traveling path length, as given in (12), of the BSA algorithm can be calculated by the sum of the forward paths and the backward paths

$$|P_{\text{BSA}}| = \left[\left(n_{\text{req}}-1\right)+\sum_{i=1}^{m}\left(\frac{L_i}{\sqrt{3}r_s}\times\frac{\alpha_i L_i}{3/2r_s}\right)\right]\times\sqrt{3}r_s. \tag{12}$$

However, the proposed IAD turns back along the shortest path from the *Dead-End* point to the *entrance* point. For example, as shown in Fig. 9, the robot encounters the *Dead-End* at s_{17} and then moves backward to the *entrance* point along the thick red dotted path, with a length of $2\times\sqrt{3}r_s$. Because the *Dead-End* problem may occur at any location, the evaluation of the length of the backward path is complicated. For simplicity, we assume that the *Dead-End* occurs at the farthest location from the *entrance* point. Therefore, the path length for turning back to the *entrance* point of obstacle o_i is

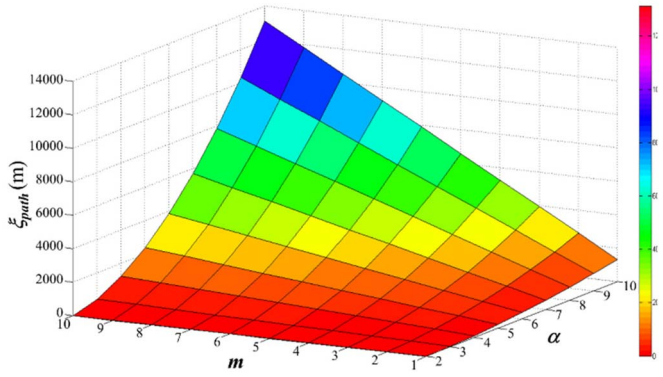

Fig. 10. Analysis of the performance improvement of the proposed IAD approach against the existing BSA approach in terms of ξ_{path}.

$\sqrt{(L_i)^2 + (\alpha_i L_i)^2}$. Consequently, the total traveling path length of the proposed IAD can be derived by

$$|P_{\text{IAD}}| \leq \left|P_{\text{IAD\_MAX}}\right| = \left(n_{\text{req}} - 1\right) \times \sqrt{3} r_s + \sum_{i=1}^{m} \left(\sqrt{(L_i)^2 + (\alpha_i L_i)^2} \right). \quad (13)$$

As a result, the improvement, denoted by x_{path}, of IAD to BSA in terms of traveling path length, can be derived by

$$\xi_{\text{path}} = |P_{\text{BSA}}| - |P_{\text{IAD}}| \geq |P_{\text{BSA}}| - \left|P_{\text{IAD\_MAX}}\right| = \sum_{i=1}^{m} \left[\left(\frac{\alpha_i (L_i)^2}{3\sqrt{3}/2 \times (r_s)^2} \right) \times \sqrt{3} r_s - \sqrt{(L_i)^2 + (\alpha_i L_i)^2} \right]. \quad (14)$$

Let $L_i = a_i \times r_s$ and $\alpha_i L_i = b_i \times r_s$, where a_i and b_i represent the widths and heights of the obstacles o_i, respectively.

Because the thickness of the obstacle is $\sqrt{3} r_s$, the obstacle o_i is solid, without any holes, when the size of o_i is less than or equal to $2\sqrt{3} r_s \times 2\sqrt{3} r_s$. Therefore, the value of ξ_{path} can be derived as

$$\xi_{\text{path}} = |P_{\text{BSA}}| - |P_{\text{IAD}}| \geq \sum_{i=1}^{m} \begin{cases} \left(\frac{2}{3} a_i b_i - \sqrt{a_i^2 + b_i^2} \right) \times r_s, & \text{if } a_i > 2\sqrt{3} \text{ and } b_{\text{i}} > 2\sqrt{3} \\ 0, & \text{otherwise.} \end{cases} \quad (15)$$

Because the size of each obstacle is larger than $2\sqrt{3} r_s \times 2\sqrt{3} r_s$, the value of ξ_{path} is larger than 0. This also implies that the traveling path length of IAD is shorter than that of BSA. Equation (15) indicates that IAD shortens the traveling path length in proportion to the size and number of obstacles. The larger and more numerous the obstacles are, the more IAD shortens the path.

According to (15), Fig. 10 depicts the improvement of the traveling path length ξ_{path} from IAD, as compared with the BSA scheme under $r_s = 25$. Consider an environment that contains m regular obstacles, each of which measures $\alpha r_s \times \alpha r_s$. The parameters m and α vary, ranging from 1 to 10 and 2 to 10, respectively. This figure shows that IAD significantly outperforms BSA in terms of traveling path length.

(a) (b) (c)

Fig. 11. Experimental environment of the (a) Lego robot. (b) Scenario 1. (c) Scenario 2.

VI. Physical Experiments

In this section, we present the physical implementation of the proposed approach in a Lego Mindstorms robot system, as shown in Fig. 11(a). The hardware of our Lego Mindstorms robot included a 32-bit ARM7 microcontroller, 256kB Flash, 64kB RAM, and 6 AA batteries as a power supply. All operations of the robot were controlled by a notebook through a Bluetooth wireless communication link (Bluetooth Class II V2.0 compliant). The robot was equipped with an ultrasonic sensor, which enabled the robot to see and detect obstacles, and to measure the distance from obstacles to itself. Three servomotors enabled the movements of the robot. Each motor had a built-in rotation sensor, which enabled the robot to record its movements and rotation precisely.

The proposed IAD was implemented on a notebook in Java. The implemented IAD sent commands (API calls within the iCommand module) through the Bluetooth wireless channel to control the robot's operations, such as spiral movements, obstacle detection, and rotation. For example, the API calls moveForward(int n) and turnLeft(d) commanded the robot to move forward for n milliseconds and turn left d degrees, respectively.

Fig. 11(b) depicts scenario 1, in which the monitoring area contained an obstacle. The area in scenario 2, shown in Fig. 11(c), contained two connected obstacles. The robot was initially placed in the top-left corner of the region. This experiment tested the robot's movement trajectory; therefore, the robot did not actually deploy sensors. Instead, the robot only detected the obstacles using its ultrasonic sensor and moved according to the *Moving Direction Rules*, as applied to the *Normal state*, *Narrow-Lane state*, and *Dead-End state*. Since scenarios 1 and 2 are indoor, the GPS cannot be used. Instead, we use the rotation speed of the robot's wheels to measure the distance moved by robot. We implemented the key components of IAD and BSA in the Lego robot and compared them in terms of path length. We define the following efficiency ratio to compare the performance levels of the IAD and BSA:

$$\text{Efficiency Ratio} = \frac{\text{IAD Path Length}}{\text{BSA Path Length}}. \quad (16)$$

As shown in Table II, the proposed IAD outperformed BSA in terms of path length and energy conservation.

In the physical environment, we used rotation speed of the robot's wheels to measure the distance moved by robot and

TABLE II
EXPERIMENTAL RESULTS

	Scenario 1	Scenario 2
Efficient ratio of robot deployment path	93.71%	97.5%

TABLE III
COMPARISON WITH DIFFERENT EXPERIMENTAL ENVIRONMENT

Localization Method	GPS	Rotation Speed of Robot's Wheels	
Environment	Dry	Dry	Slippery
Average Error Ratio	–	3.82 %	17.5%

TABLE IV
SIMULATION PARAMETERS

Parameter	**Value**
Simulation Tool	TOSSIM/TinyViz
Dimension of the Topography	800m × 800m
Communication Range	50m
Sensing Range	25m
Robot Speed	2 m/s
Mobility Cost	8.267J/m
Simulation Repetitions	500 times

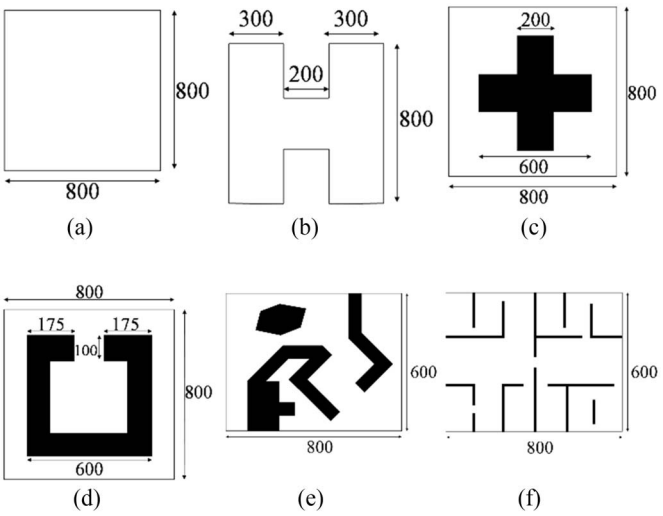

Fig. 12. Shapes of regular obstacles considered in the simulation. (a) Square-shaped region to be monitored. (b) *H*-shaped region to be monitored. (c) *X*-shaped obstacle. (d) *C*-shaped obstacle. (e) Complex obstacles. (f) Exhibition center.

hence the location information can be obtained. In this way, the location error can cause the robot deploy at the locations which are different to the expected locations. Let d_{basic} denote the distance of basic movement and d_{actual} denote the actual distance by using the rotation speed of robot wheels. To measure the impact of the inaccurate location, we measure the distance between actual location and expected location and calculate the average error ratio of each movement by applying

$$\text{Average Error Ratio} = \frac{|d_{\text{basic}} - d_{\text{actual}}|}{d_{\text{basic}}}. \tag{17}$$

As shown in Table III, the average error ratio is significant if the ground is slippery. However, in case of dry ground, the average error ratio is acceptable. We believe that the average error ratio can be further improved if some other complex localization mechanisms [20], [21] are applied.

VII. SIMULATION STUDY

In this section, we use simulator TOSSIM/TinyViz [22] to compare the performance levels of the proposed IAD mechanism and previously published works [16]–[18], in terms of the coverage ratio, energy efficiency, deployment path length, and required stack space.

This section presents the report results of simulations coded in TOSSIM/TinyViz [22], which has been widely used in simulating realistic and expandable networks. The proposed IAD mechanism is compared with the previous works including ORRD, [16], OFPD, [17], and BSA, [18]. Table IV summarizes the simulation environment. The parameters values of the communication range and the sensing range are the typical parameters of Berkeley motes [23]. The robot is assumed to be equipped with a compass and a constant number of Berkeley motes. The parameters values of the robot speed and the mobility cost refer to the typical parameters of the Robomote [24]. The experimental environment is described below.

The shape of obstacle can have notable effects on the efficiency of the robot deployment. Two types of obstacles were considered in the experiments: 1) regular obstacles and 2) randomly generated irregular obstacles. As shown in Fig. 12, regular obstacles with six different shapes were generated.

In the environments with irregular obstacles, the numbers of generated obstacles were 0, 3, 6, 9, 12, or 15. The process for generating an irregular obstacle is more complicated than the process for generating a fixed obstacle. In the experiments, each irregular obstacle was composed of k unit squares, with each unit square having an edge length of r_s, where r_s denotes the sensing range.

The robot was initially placed at the top-left corner of the region to be monitored. Each simulation result was collected based on the average of 200 independent runs.

A. Coverage Ratio

The coverage ratio of a WSN represents its surveillance quality. It denotes the ratio of the covered area to the area of whole region to be monitored. Fig. 13 compares the coverage ratios of the IAD, ORRD, OFPD, and BSA mechanisms for different scenarios. In Fig. 13(a), the region to be monitored contains regular obstacles of various shapes. Although the ORRD and OFPD mechanisms applied rules to cope with the obstacle problem, they did not handle the *Dead-End* problem and thus caused coverage holes. The BSA and the proposed IAD achieved a 100% coverage ratio because the *Dead-End* problem was overcome by different forms of *Dead-End* rules. Fig. 13(b) simulates an environment that contains irregular obstacles. The number of obstacles ranged from 0 to 15. Similar to the fixed obstacle environment, the BSA and the proposed IAD outperformed the other two mechanisms because they considered the *Dead-End* problem. The coverage ratios of the proposed IAD and BSA slightly dropped when irregular obstacles existed, as compared with regular obstacle environments. The major reason was that the arc edges

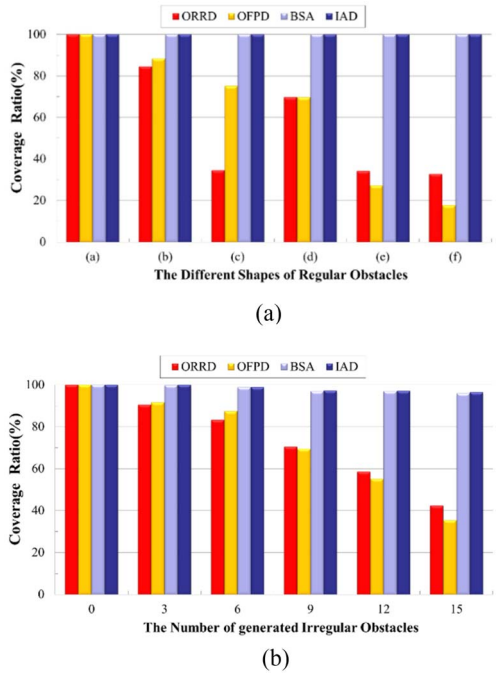

Fig. 13. Coverage ratios of different mechanisms. Results for experimental environments containing (a) regular obstacles of different shapes and (b) irregular obstacles.

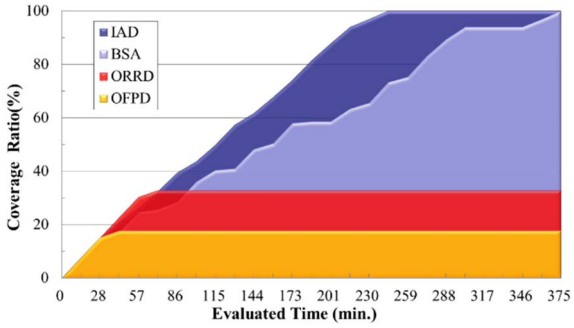

Fig. 14. Coverage ratios of different mechanisms at different evaluation times. The obstacle environment given in Fig. 12(f) is considered.

of the irregular obstacles caused small holes at some obstacle boundaries. In general, the proposed IAD and BSA achieved coverage ratios above 96.5% and outperformed ORRD and OFPD in all cases. The BSA mechanism can achieve high coverage ratios. However, it requires a large memory space. In addition, the path traversed by the robot is not efficient when the *Dead-End* situation is encountered.

Fig. 14 considers the regular obstacles environment given in Fig. 12(f) and investigates the achieved coverage ratio at different evaluation time. Since the existing *ORRD* and OFPD mechanisms did not handle the *Dead-End* problem, the coverage ratio is stop growing at 32.8% and 17.6%, respectively. The BSA and the proposed IAD mechanisms achieve full coverage since they can overcome the *Dead-End* problem. As shown in Fig. 14, by applying the proposed IAD mechanism, the coverage ratio grows significantly with the evaluation time. Conversely, the curve of the existing BSA increases slowly with the evaluation time. This simulation result shows that

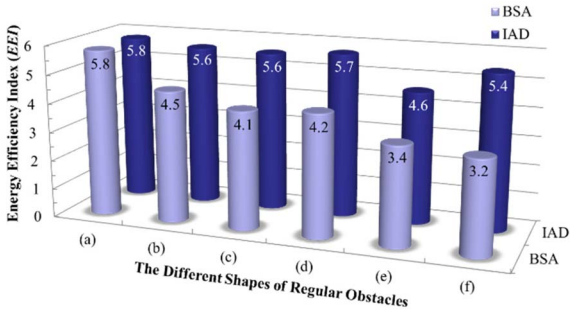

Fig. 15. Comparison of the proposed IAD and BSA mechanisms in terms of EEI. The experimental environment contains regular obstacles of different shapes.

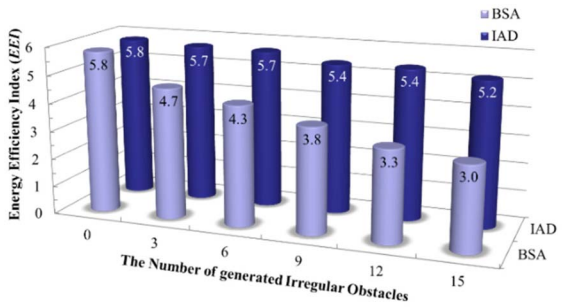

Fig. 16. Comparison of the proposed IAD and published BSA mechanisms in terms of EEI. The experimental environments contained different numbers of irregular obstacles.

the *Dead-End* handling policy proposed in IAD mechanism is more efficient than that of existing BSA mechanism.

B. Energy Efficiency

The ORRD and OFPD are not considered in the following experiments because that they did not address the *Dead-End* problem. Hence, their energy efficiency, deployment path length, and required stack space for full coverage deployment cannot be evaluated. The following experiments compare only the proposed IAD and existing BSA robot deployment mechanisms.

The existence of unpredicted obstacles can increase the path length or even raise the *Dead-End* problem. Furthermore, the *Dead-End* problem additionally increases the path length because the robot must go back to the unexplored region. Figs. 15 and 16 compare the proposed IAD and the BSA deployment mechanisms in terms of the energy efficiency index (EEI), which is measured by (18). A deployment mechanism with a large EEI value indicates that a large coverage area can be achieved by consuming less energy. This also indicates that the robot deploys the sensors along a more efficient path

$$\text{EEI} = \frac{\text{Covered Area } (\text{m}^2)}{\text{Total Energy Consumption (J)}}. \tag{18}$$

In Fig. 15, the results for regular obstacles of various shapes are presented. The BSA mechanism pushes all visited locations in the stack and turns back to the *entrance* point along a long path whenever the robot encounters a *Dead-End* situation.

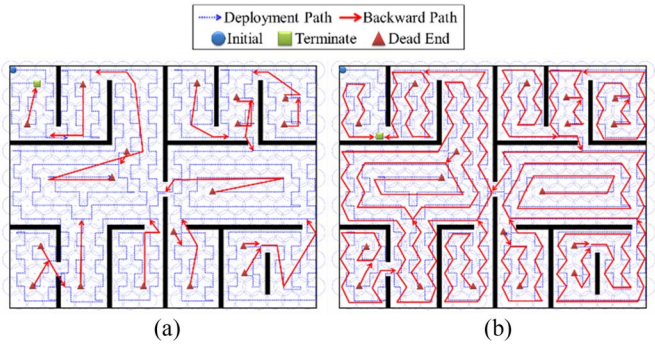

Fig. 17. Snapshots of robots' backtracking paths for IAD and BSA mechanisms using the obstacle environment given in Fig. 12(f). Execution of (a) IAD and (b) BSA.

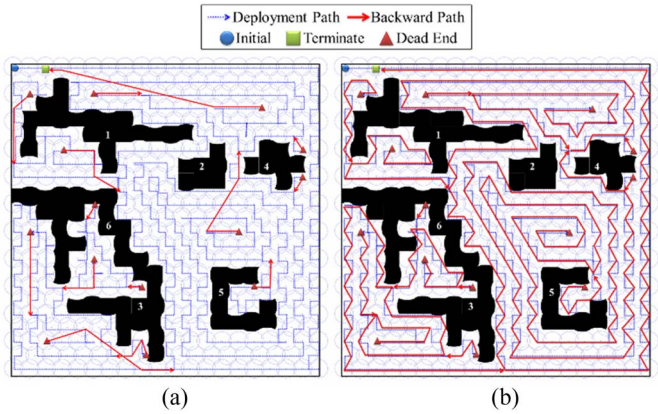

Fig. 18. Snapshots of robot deployments applying IAD and BSA using a scenario containing six irregular obstacles. Execution of (a) IAD and (b) BSA.

This leads to an inefficient backward path which consumes considerable energy. However, the proposed IAD mechanism only pushes the locations of *entrance*, *exit*, or *turn* points in the stack. As a result, the robot can go back to an unexplored location along a more efficient path whenever the robot encounters the *Dead-End* problem. As shown in Fig. 15, the proposed IAD consistently obtained larger EEI values of EEI than did the BSA mechanism. Despite the performance of the proposed IAD slightly dropping when complex obstacles were presented, its deployment efficiency nevertheless exceeded that of the BSA mechanism in all cases.

Fig. 16 shows results for experiments in which a number (ranging from 0 to 15) of randomly-generated irregular obstacles were present in the region to be monitored. The EEI of the BSA mechanism notably decreased as the number of obstacles increased. The proposed IAD mechanism applied the *Narrow-Lane Rule* and *GoBack Rule* to cope with the *Dead-End* problem and consequently outperformed the BSA mechanism in terms of *EEI* in all cases.

C. Trajectory Path

Fig. 17 presents snapshots of the IAD and BSA deployments for the scenario given in Fig. 12(f). The deployment paths are marked by thin blue dotted lines and the backward paths are marked by thick red lines. The BSA path was much longer than the IAD path. This occurred because the robot implementing IAD moved to the nearest unexplored location along an efficient path when it encountered the *Dead-End* problem. Consequently, the backward path in BSA was 5.04 times longer than that in IAD.

Fig. 18 compares IAD and BSA paths for an environment containing six irregular obstacles. The results are similar to those in Fig. 17 because the proposed IAD was designed to cope with the *Dead-End* problem more efficiently.

D. Required Stack Space

Fig. 19 compares the IAD and BSA mechanisms in terms of the *Required Stack Space* by considering a scenario containing regular obstacles. In a region containing small obstacles, the robot must deploy sensors in a large region without obstacles. Thus, the path of the robot decreases as the obstacle size increases. As shown in Fig. 19, the BSA mechanism always

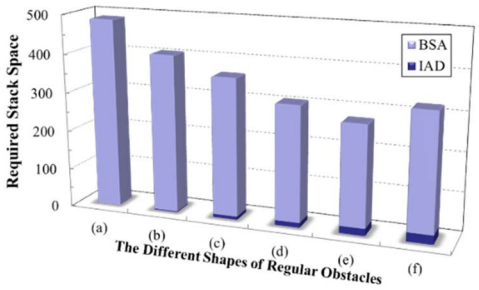

Fig. 19. Comparison of the proposed IAD and published BSA mechanisms in terms of required stack space. The experimental environments contain regular obstacles of different shapes.

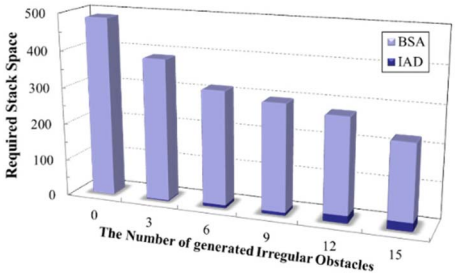

Fig. 20. Comparison of the proposed IAD and published BSA mechanisms in terms of required stack space. The experimental environments contained different numbers of irregular obstacles.

requires an extremely large stack space, as compared with the proposed IAD mechanism. This occurs because that the BSA mechanism pushes all visited locations onto the stack. By contrast, the proposed IAD pushes only the locations of *entrance*, *exit*, or *turn* points onto the stack so that the robot can return to the unexplored locations. As a result, the stack space required by the proposed IAD is notably smaller than that of the BSA.

Fig. 20 further illustrates the required stack space for an environment containing irregular obstacles. The number of irregular obstacles was randomly generated in a range from 0 to 15. The results are similar to those of Fig. 19 because the proposed IAD pushes only the locations of *entrance*, *exit*, and *turn* points onto the stack.

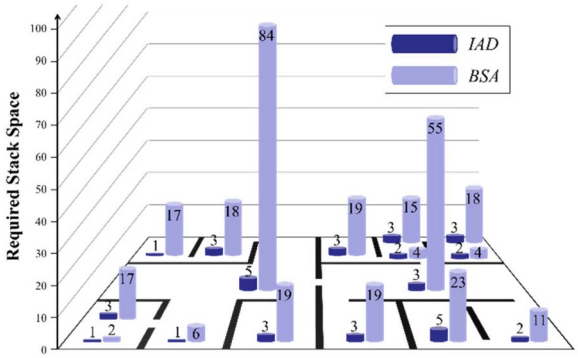

Fig. 21. Amounts of stack space required for handling each *Dead-End* problem when applying IAD and BSA mechanisms to the obstacle environment depicted in Fig. 12(f).

Fig. 21 concerns an environment containing regular obstacles that are arranged as given in Fig. 12(f). The size of monitored region was 800 × 600 m^2. Two instances are depicted at the location where the robot encounters a *Dead-End* situation. The light blue and deep blue bars present the required stack space by applying the BSA and IAD mechanisms, respectively.

As shown in Fig. 21, the BSA mechanism requires a large stack space to record the visited locations to overcome each *Dead-End* situation. However, the robot locates the unexplored location along the original trajectory in a backward direction. Combining Figs. 17 and 21 reveals that the path length for finding the unvisited location increases with the required stack space. When the robot applies the BSA mechanism but encounters a *Dead-End* situation, it turns back to an unexplored location by popping the locations stored in the stack and thus moves using a long path, as shown in Fig. 17(b). By contrast, the proposed IAD mechanism only pushes some proper locations onto the stack. As a result, the robot applying the proposed IAD mechanism requires smaller stack space than that applying the BSA approach, as shown in Fig. 21. The impact of the small required stack space on the short backward path is illustrated in Fig. 17(a).

VIII. Conclusion

This paper presents an efficient algorithm, called IAD, which is intended to overcome unknown obstacles and deploy sensors, thereby achieving full coverage of monitored areas. The *Moving Direction* and *Horizontal and Vertical Movement* rules were proposed to minimize the number of deployed sensors but achieve full coverage of an obstacle-free region. In addition, to overcome the *Dead-End* problem which occurs because of the existence of obstacles, this paper proposed a state management scheme in which the *Normal*, *Narrow-Lane*, *Dead-End*, and *GoBack* states and their corresponding rules were developed. The results of the performance study indicate that the proposed IAD algorithm achieves superior performance to the existing deployment mechanisms in terms of coverage ratio, energy efficiency, deployment path length, and required stack space.

References

[1] Z. Li *et al.*, "Trajectory-tracking control of mobile robot systems incorporating neural-dynamic optimized model predictive approach," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 6, pp. 740–749, Jun. 2016.

[2] C. Ye, S. Hong, X. Qian, and W. Wu, "Co-robotic cane: A new robotic navigation aid for the visually impaired," *IEEE Syst., Man, Cybern. Mag.*, vol. 2, no. 2, pp. 33–42, Apr. 2016.

[3] S. Nahavandi, "Robot-based motion simulators using washout filtering: Dynamic, immersive land, air, and sea vehicle training, vehicle virtual prototyping, and testing," *IEEE Syst., Man, Cybern. Mag.*, vol. 2, no. 3, pp. 6–10, Jul. 2016.

[4] R. V. Kulkarni, A. Förster, and G. K. Venayagamoorthy, "Computational intelligence in wireless sensor networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 13, no. 1, pp. 68–96, 1st Quart., 2011.

[5] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robot. Auton. Syst.*, vol. 61, no. 12, pp. 1258–1276, Dec. 2013.

[6] W. He *et al.*, "Model identification and control design for a humanoid robot," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 47, no. 1, pp. 45–57, Jan. 2017.

[7] L. Liu and H. Ma, "On coverage of wireless sensor networks for rolling terrains," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 1, pp. 118–125, Jan. 2012.

[8] P. Zhang, I. Nevat, G. W. Peters, G. Xiao, and H.-P. Tan, "Event detection in wireless sensor networks in random spatial sensors deployments," *IEEE Trans. Signal Process.*, vol. 63, no. 22, pp. 6122–6135, Nov. 2015.

[9] Z. Liao, J. Wang, S. Zhang, J. Cao, and G. Min, "Minimizing movement for target coverage and network connectivity in mobile sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 7, pp. 1971–1983, Jul. 2015.

[10] T.-Y. Lin, H. A. Santoso, and K.-R. Wu, "Global sensor deployment and local coverage-aware recovery schemes for smart environments," *IEEE Trans. Mobile Comput.*, vol. 14, no. 7, pp. 1382–1396, Jul. 2015.

[11] Y. Song, B. Wang, Z. Shi, K. R. Pattipati, and S. Gupta, "Distributed algorithms for energy-efficient even self-deployment in mobile sensor networks," *IEEE Trans. Mobile Comput.*, vol. 13, no. 5, pp. 1035–1047, May 2014.

[12] H. Mahboubi, K. Moezzi, A. G. Aghdam, and K. Sayrafian-Pour, "Distributed deployment algorithms for efficient coverage in a network of mobile sensors with nonidentical sensing capabilities," *IEEE Trans. Veh. Technol.*, vol. 63, no. 8, pp. 3998–4016, Oct. 2014.

[13] M. A. Batalin and G. S. Sukhatme, "The design and analysis of an efficient local algorithm for coverage and exploration based on sensor network deployment," *IEEE Trans. Robot.*, vol. 23, no. 4, pp. 661–675, Aug. 2007.

[14] C.-H. Ou and W.-L. He, "Path planning algorithm for mobile anchor-based localization in wireless sensor networks," *IEEE Sensors J.*, vol. 13, no. 2, pp. 466–475, Feb. 2013.

[15] J. Rezazadeh, M. Moradi, A. S. Ismail, and E. Dutkiewicz, "Superior path planning mechanism for mobile beacon-assisted localization in wireless sensor networks," *IEEE Sensors J.*, vol. 14, no. 9, pp. 3052–3064, Sep. 2014.

[16] C.-Y. Chang, C.-T. Chang, Y.-C. Chen, and H.-R. Chang, "Obstacle-resistant deployment algorithms for wireless sensor networks," *IEEE Trans. Veh. Technol.*, vol. 58, no. 6, pp. 2925–2941, Jul. 2009.

[17] C.-Y. Chang, J.-P. Sheu, Y.-C. Chen, and S.-W. Chang, "An obstacle-free and power-efficient deployment algorithm for wireless sensor networks," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 39, no. 4, pp. 795–806, Jul. 2009.

[18] E. González, O. Álvarez, Y. Díaz, C. Parra, and C. Bustacara, "BSA: A complete coverage algorithm," in *Proc. IEEE ICRA*, Barcelona, Spain, Apr. 2005, pp. 2040–2044.

[19] K. Sakai, M.-T. Sun, W.-S. Ku, T. H. Lai, and A. V. Vasilakos, "A framework for the optimal *k*-coverage deployment patterns of wireless sensors," *IEEE Sensors J.*, vol. 15, no. 12, pp. 7273–7283, Dec. 2015.

[20] D. Niculescu and B. Nath, "Ad hoc positioning system (APS) using AOA," *IEEE INFOCOM*, San Francisco, CA, USA, Mar./Apr. 2003, pp. 1734–1743.

[21] N. Bulusu, J. Heidemann, and D. Estrin, "GPS-less low-cost outdoor localization for very small devices," *IEEE Pers. Commun.*, vol. 7, no. 5, pp. 28–34, Oct. 2000.

[22] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," in *Proc. ACM SenSys*, Los Angeles, CA, USA, Nov. 2003, pp. 126–137.

[23] J. Hill and D. Culler, "Mica: A wireless platform for deeply embedded networks," *IEEE Micro*, vol. 22, no. 6, pp. 12–24, Nov. 2002.
[24] G. T. Sibley, M. H. Rahimi, and G. S. Sukhatme, "Robomote: A tiny mobile robot platform for large-scale ad-hoc sensor networks," in *Proc. IEEE ICRA*, Washington, DC, USA, May 2002, pp. 1143–1148.

Chih-Yung Chang (M'05) received the Ph.D. degree in computer science and information engineering from National Central University, Zhongli, Taiwan, in 1995.

He is currently a Full Professor with the Department of Computer Science and Information Engineering, Tamkang University, New Taipei City, Taiwan. His current research interests include Internet of Things, wireless sensor networks, *ad hoc* wireless networks, and long-term evolution broadband technologies.

Dr. Chang has served as an Associate Guest Editor for several SCI-indexed journals, including the *International Journal of Ad Hoc and Ubiquitous Computing* from 2011 to 2016, the *International Journal of Distributed Sensor Networks* from 2012 to 2014, *IET Communications* in 2011, *Telecommunication Systems* in 2010, the *Journal of Information Science and Engineering* in 2008, and the *Journal of Internet Technology* from 2004 to 2008.

Yu-Ting Chin received the B.S. and M.S. degrees in computer science and information engineering from Aletheia University, New Taipei City, Taiwan, in 2007 and 2010, respectively, where he is currently pursuing the Ph.D. degree.

His current research interests include Internet of Things, wireless sensor networks, *ad hoc* wireless networks, and software-defined networking and low-power wide-area network technologies.

Cheng-Chang Chen received the B.S. and M.S. degrees in computer science and information engineering from Tamkang University, Taipei, Taiwan, in 2007 and 2009, respectively, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Information Engineering.

His current research interests include Internet of Things, cyber-physical systems, wireless sensor networks, *ad hoc* wireless networks, vehicular ad hoc networks, and worldwide interoperability for microwave access broadband technologies.

Mr. Chen was a recipient of several scholarship grants in Taiwan and has participated in many projects related to wireless sensor networks, vehicular ad hoc networks, Internet of Things, and WiMAX broadband networks.

Chao-Tsun Chang (M'12) received the Ph.D. degree in computer science and information engineering from National Central University, Taoyuan City, Taiwan, in 2006.

He is currently an Associate Professor with the Department of Information Management, Hsiuping University of Science and Technology, Taichung, Taiwan. In the recent ten years, he has directed 12 researching projects, including five National Science Council projects, and seven information system developments. His current research interests include *ad hoc* wireless networks, wireless sensor networks, Bluetooth radio networks, and mobile computing.

Dr. Chang is a member of the IEEE Computer and Communication Societies.