Energy-Efficient Mechanisms for Coverage Recovery in WSNs

Chih-Yung Chang, Sheng-Wen Chang, Ming-Hsien Li, and Yu-Chieh Chen
Dept. Computer Science and Information Engineering
Tamkang University, Taipei, 251, Taiwan
E-mail: cychang@mail.tku.edu.tw, swchang@wireless.cs.tku.edu.tw, 695410505@s95.tku.edu.tw, ycchen@wireless.cs.tku.edu.tw,

***Abstract*—In wireless sensor networks(WSNs), coverage of the monitoring area represents the quality of service (QoS) related to the surveillance. In literature, a number of studies developed robot deployment and patrol algorithms. However, the efficiency of existing repair algorithms can be further improved in terms of time and energy consumption. Moreover, existing repair algorithms did not consider the existence of obstacles and the constraint of limited energy of the robot. This paper presents novel tracking mechanism and robot repairing algorithm for maintaining the coverage quality for a given WSN. Without the support of location information, the tracking mechanism leaves the robot's footmarks such that sensors that are nearby the failure region can learn better routes for sending repairing requests to the robot. Upon receiving several repairing request messages, the robot applies the proposed repairing algorithm to establish an optimal route that passes through all failure regions with minimal overhead in terms of the required time and power consumption. In addition, the proposed repairing algorithm also considers the remaining energy of the robot so that the robot can be back to home for recharging energy and overcome the unpredicted obstacles. Performance study reveals that the developed protocol can efficiently maintain the coverage quality while the required time and energy consumption are significantly reduced.**

***Keywords*—Deployment, Coverage Quality, Repair, Robot.**

I. Introduction

Wireless Sensor Networks are composed of many sensor nodes embedded with simple process, fewer memory, tiny sensing material, and energy-limited battery. The accuracy of sensing information depends on the coverage quality in the monitoring region. In literature, previous works [1][2][4] use robot to deploy sensors in a dangerous region that is unsuitable for human deployment. Some researches [1][2][3][4] exploit the robot to execute important tasks such as sensor deployment, patrol, or hole repair. In [1], the robot deploys the sensors according to the predefined direction priority of south, west, north, and east. Each sensor counts the time interval that the robot does not visit for each direction. The deployed sensors may guide the robot's movement by suggesting a suitable direction with maximal time interval to the robot. However, the approach can not guarantee full coverage and may cause too much sensing redundancy if the robot encounters obstacles. Furthermore, the developed robot movement policy did not take into account the energy constraint of the robot.

Some other research [4] proposes an obstacle-free robot deployment mechanism that deploys the monitoring region with near-minimal number of sensors and likely achieves the full coverage purpose. As shown in Figs. 1 and 2, the proposed deployment algorithm deploys minimal number of sensors in the environments with and without obstacles, respectively. However, since sensor nodes are battery powered and deployed in the outdoor environment, they might be failure due to energy exhaustion or environmental influence, and hence result in the WSN coverage-loss. In [4], no algorithm is for the robot to cope with the network maintaining problem.

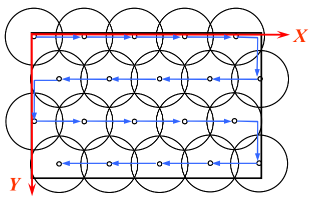


Figure 1. The snake-like movement deployment proposed in [4].

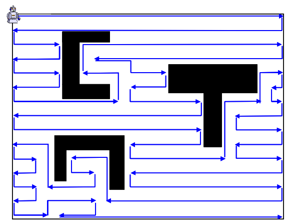


Figure 2. The OFRD deployment algorithm [4] overcomes the different shapes of obstacles.

Previous research [1] proposed a patrol algorithm that is able to cope with the network-maintaining task. It assumes that the robot equips with a compass and is able to detect an obstacle. To guide the robot's movement, each deployed sensor maintains the time interval that the robot does not visit for each direction of south, east, north, and west. When the robot intends to make a decision of movement direction, it communicates with the closest deployed sensor, and then moves toward the direction that has the largest value of time interval maintained in that sensor. Although the coverage quality can be maintained by the robot, however, the occurrence of failure sensors might not be related to the time interval. For example, frequent events occurred at a region might cause the sensors of that region energy exhaustion. As a result, the failure region should passively wait for the robot's visiting. In addition, the robot did not leave the trajectory on the deployed sensors and hence those sensors that are nearby the failure regions are unable to send the repairing request to the robot. Consequently, the robot might visit the failure regions after a long period of time. The patrolling mechanism proposed in [1] also employs a Home algorithm that enables the robot going home for the energy recharge. However, the remaining energy of the robot and the existence of obstacles are not taken into consideration. Consequently, the robot might exhaust its energy during the execution of repairing task.

In proportion to the abovementioned drawbacks, we seek to propose novel tracking mechanism and robot repairing algorithm. With the extension of the existing robot deployment algorithm [4], the tracking mechanism is proposed to enable the robot leaving the trajectory information on each deployed sensor. According to this information, sensors nearby the failure region can learn a better path to efficiently notify the robot for repairing. Then the robot repairing algorithm constructs a power-conservation repairing path after the robot collecting the request notifications during a predefined period of time. The proposed repairing algorithm considers the existence of unpredicted obstacles and the constraint in robot's remaining energy. As a result, the proposed repairing algorithm efficiently takes minimal time and consumes minimal energy to recover the failure regions.

II. Tracking Mechanism

This paper considers a single robot that carries limited static sensor nodes and is embedded with a compass through which the robot is aware of its moving direction. We assume that a number of sensors have been deployed in the monitoring region by applying the obstacle-free robot deployment mechanism proposed in [4].

A deployed WSN might be coverage-loss due to the failure of sensors in some region. Let the sensors that are nearby the failure region be request initiators which are responsible for sending the repairing requests to the robot. Since the robot moves for executing the repairing task, its location is not known by sensors. Therefore, a tracking mechanism is essential for those sensors that are nearby the failure region to track where the robot is and send the repairing requests to the robot. This section proposes the tracking mechanism which enhances the deployment algorithm OFRD [4] and aims at helping all sensors learn better routes to deliver messages from themselves to the robot. The basic idea behind the proposed tracking mechanism is that the robot leaves footmarks on the deployed sensors when it executes the deployment task. Two types of footmarks are set up on each deployed sensor by the robot. The first type of the footmark is a two-dimensional coordinates which are constructed during the execution of network deployment task. This type of footmark aims to create a coordinate system which represents the physical location of the monitoring region. A virtual two-dimensional coordinate system will be established by the robot which gives each sensor node a unique Virtual Coordinates (or VC in short). According to the VC of each failure region, the robot can move to the location for executing repairing task. Since the robot is aware of its moving direction, the coordinates of the first deployed sensor node are given by (0, 0) and the x-axis and y-axis of the deployed sensor will be increased by one when the robot moves and deploys a sensor in right and down directions, respectively. Figure 3 shows the VC of the deployed sensors based on the snake-like deployment algorithm proposed in [4].

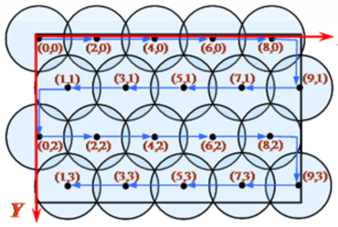


Figure 3. The robot leaves a footmark on each sensor. It assigns each deployed sensor with coordinates and thus the VC system of the WSN is constructed.

In addition to the VC, each sensor should additionally maintain the second type of footmark, called Counter Value, which provides tracking information and helps itself learn a better route to the robot. The counter value is represented as <Broadcast ID, Hops> where the Broadcast ID maintains the number of broadcasts received by this sensor and the Hops maintains the number of hops between the sensor and robot. The initial value of the Broadcast ID is zero. Whenever the robot deploys a sensor during the execution of the network deployment task, the value of Hops of the deployed sensor is increased by one. Furthermore, when the robot completes the deployment task and moves around the monitoring region, it should update the counter value of the visited sensor by increasing the Hops value by one. As a result, in case that the Broadcast ID is zero, a sensor with a larger Hops value represents the robot is closer to that sensor node. Each sensor should maintain its Counter Values and the neighboring sensor with the largest Counter Value.

Each request initiator which detects the failure neighboring sensor will utilize the Counter Value to trace the current robot location and send the repairing request to the robot. The repairing request packet consists of the VC of the failure sensor and the ID of neighbor sensor node which has the largest Counter Value. Upon receiving the repairing request packet, the node with the ID defined in repairing packet will be responsible for forwarding the packet. It simply forwards the request packet to the neighbor that has the largest Counter Value. Since a sensor closer to the robot maintains a larger Hops value, the request packet will be delivered to the robot step by step. The robot will wait for a predefined constant time period to receive several request packets transmitted by different request initiators. Then the robot applies the proposed repairing algorithm to construct a repairing path that passes through all failure regions and repairs the failure regions along the constructed path.

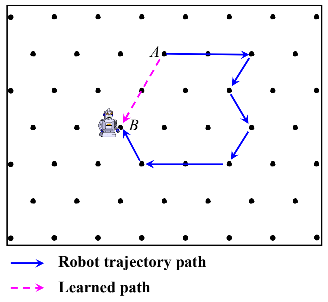


Figure 4. The route constructed according to the trajectory path (solid path) of the robot is rugged and inefficient. The proposed X-correction mechanism enables the robot learns a better route (dotted path) from itself to the robot.

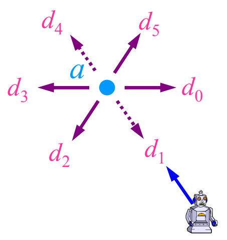


Figure 5. The basic concept of X-correction mechanism.

2.1 X-correction Mechanism

Although the Counter Value can be used by all sensors to trace the location of the robot, however, the tracking path may be rugged and inefficient, as shown in Fig. 4. To help the request initiators learn a better route, an X-correction mechanism is proposed herein. After the robot completes the deployment task, it will periodically transmit the X-correction packet to adjust the Counter Value. Although the flooding mechanism is the simplest way to enable all sensors to maintain the up-to-date location of the robot, however, it raises significant control overheads. The proposed X-correction mechanism corrects the Counter Value along two cross-paths and hence significantly reduces the control overhead. The control packet broadcasted in the X-correction mechanism is called X-correction packet which aims to reduce the inefficient and rugged tracking path and raise fewer control overhead. Let k be a predefined constant *threshold*. The robot will transmit an X-correction packet to update the Counter Value when it moves every k steps. Since each sensor node has six neighbors, the robot will select four directions to broadcast the X-correction packet and hence the packet will be forwarded along X-shape paths. According to snake-like deployment proposed in [4], each deployed sensor has six neighboring sensors that are located in six different directions d_i, for all $0 \le i \le 6$. Here, we assume that the robot moves along straight line. Therefore, when a robot intends to move to a static sensor, say a, it will be close to sensor a along direction d_i and then leave sensor a along direction $d_{(i+3) \bmod 6}$. The basic concept of the X-correction mechanism is that the robot broadcasts the X-correction packet along the four directions other than the current entering and leaving directions d_i and $d_{(i+3) \bmod 6}$. Figure 5 depicts the basic concept of the X-correction mechanism. The robot is close to and leave sensor a along directions d_1 and d_4, respectively. The broadcasted X-correction packet will be transmitted along directions d_0, d_2, d_3, and d_5. As a result, the trajectory of the X-correction packet is like an X-shape.

As shown Fig. 6, the X-correction packet consists of four fields, including the current location (expressed by VC) of the robot, Broadcast ID, Hops, and the moving direction. The Broadcast ID represents the version of the X-correction packet transmitted by the robot. The value of version is initialized by zero and will be automatically increased by one whenever the robot broadcasts an X-correction packet. The value of Hops is initialized by k-1. Upon receiving the X-correction packet, a forwarding sensor that applies the X-transmission Rule and determines to rebroadcast the packet will replace its own Broadcast ID and Hops values according to the corresponding fields in the X-correction packet. Then the forwarding sensor decreases the value of Hops in the X-correction packet and subsequently rebroadcasts the revised packet.

$<x, y>$: Counter value (x, y) : VCL

X-correction Packet Format	Robot Location	Broadcast ID	Hops	Moving Direction

Figure 6. The X-correction packet format.

The following presents how each sensor that receives an X-correction packet implements the X-correction mechanism. Let the robot be nearby the sensor A whose VC value and counter value are (s, t) and (0, k), respectively. The robot broadcasts an X-correction packet. Upon receiving the X-correction packet, any sensor s with VC=(x, y) will make decision whether or not the packet should be forwarded. We classify the moving directions of the robot into three cases to discuss which sensors should rebroadcast the received X-correction packet. In case that sensor s should broadcast the packet, it updates its own counter value and the value of Hops field of the packet and then rebroadcasts the updated packet. The following presents the X-correction mechanism.

X-correction Mechanism

```
Let the robot broadcasts an X-correction packet near by the sensor $A$
whose VC value and counter value are $(s, t)$ and $(0, k)$, respectively.
Let node $s$ receive X-correction packet = $((s, t)$, B ID, Hops, $d)$
Case $d$ of
    $d_o,d_3$ : if $|x-s| == |y-t|$ then flag = true
    $d_1,d_4$ : if $(x-s == -(y-t)$ or $y == t)$ then flag = true
    $d_2,d_5$ : if $(x-s == y-t$ or $y == t)$ then flag = true
If (flag = true)
{
    replace VC of nodes with (B ID, Hops)
    update X-correction packet : Hop = Hop – 1
    transmit the packet
}
```

Figure 7 gives an example to illustrate the proposed X-correction mechanism. As shown in Fig. 7(a), each deployed sensor maintains its original VC and counter values which are set up during the network deployment phase. For example, the VC value and counter value of sensor a are (9, 3) and (0, 63), respectively. We assume that that the robot moves close to sensor a from the direction d_1 and transmits an X-correction packet at that location. The X-correction packet contains location=(9, 3), Broadcast ID=1, Hops=62 and Directions=d_1. Upon receiving the packet, the neighboring sensor d does nothing since it applies the X-correction mechanism and results flag=false. On the other hand, the sensor b plays a forwarder and replaces its counter value with (1, 62). Then sensor b updates the value of Hops in X-correction packet by 61 and rebroadcasts the packet. Upon receiving the packet, sensor c replaces its counter value with (1, 61). Figure 7(b) depicts the resultant counter values of all sensors.

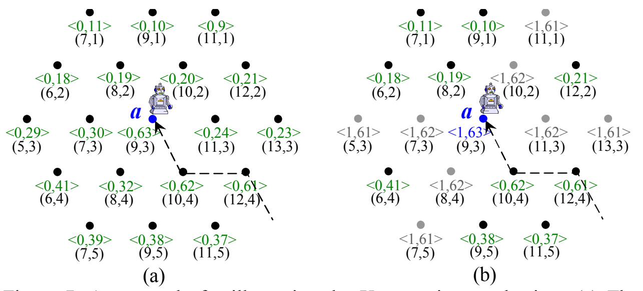
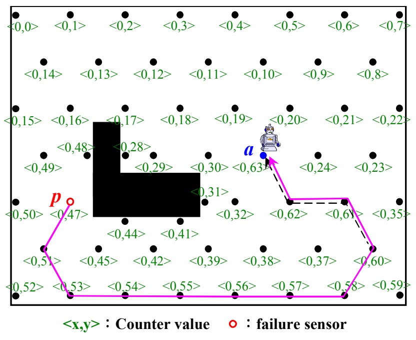
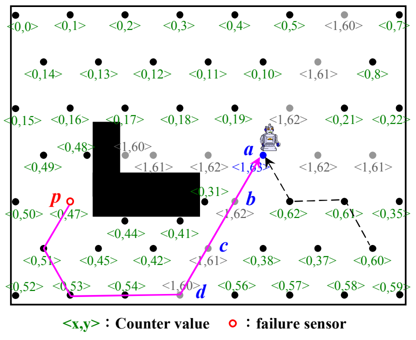


Figure 7. An example for illustrating the X-correction mechanism. (a) The original VC value and counter value maintained by each deployed sensor. The robot broadcasts an X-correction packet at location (9, 3). (b) Sensors apply the X-transmission Rule and update their own counter values.



(a)



(b)

Figure 8. An example that illustrates the benefit obtained by applying the X-correction mechanism. (a) Without applying the X-correction mechanism, the RR packet is forwarded to the robot along the trajectory of the robot, which is rugged and inefficient path. (b) Applying the X-correction mechanism to adjust the Counter Value of sensors, the RR packet is delivered to the robot along an efficient route.

2.2 Tracking the Robot

In the previous subsection, an X-correction mechanism has been presented to update the counter values of all sensors located on the X-shape lines. This subsection introduces how request initiators learn better routes for sending the repairing request message to the robot.

To maintain the coverage quality, whenever a sensor detects the absence of its neighboring sensor, it plays the request initiator and intends to request the robot, asking for executing the redeployment task. The request initiator will initiate a repairing request packet or RR packet in short. The RR packet contains the location of failure sensor in a format of VC. To deliver the RR packet to the robot along a better route, the request initiator forwards the RR packet to the neighboring sensor that has the largest counter value. Upon receiving the RR packet, the forwarder simply forwards the packet to the neighbor that has the largest counter value. We notice that the WSN applied the X-correction mechanism has the characteristic that the robot is always located at the sensor with largest counter value. Consequently, the RR packet can be delivered to the robot along a better route.

Figure 8 shows an example that illustrates how a request initiator learns a better route after the X-correction mechanism has been applied on the WSN. In Fig. 8(a), the trajectory of the robot's movement is represented by the dotted line and the robot broadcasts an X-correction packet nearby the sensor *a*. We assume that sensor *p* detects the absence of its neighbor and plays the role of request initiator. In Fig. 8(a), the solid line denotes the route traversed by the RR packet in case that the X-correction mechanism is not applied to the WSN. The RR packet will be delivered to the robot by the forwarders that have the largest counter value. In fact, the packet is forwarded along the robot's trajectory. As a result, the length of route traversed by the RR packet is 11. However, by applying the proposed X-correction mechanism in Figure 8(b), sensors *b*, *c*, and *d* have updated their counter values. Therefore, sensor *d* will forward the received RR packet to sensor *c*. Similarly, sensor *c* subsequently forwards the RR packet to sensor *b*. As a result, the length of route traversed by RR packet is 7. In comparison, applying the X-correction mechanism saves 4 hops of route for delivering the RR packet to the robot.

III. Power-Conservation Repair Mechanism

Upon collecting several repairing request packets from different request initiators during a predefined time period, the robot executes the redeployment task for repairing the failure regions so that the monitoring quality can be maintained. According to the virtual cooperates of the failure regions, a repairing algorithm is proposed herein for the robot to construct an efficient route that passes through all failure regions and takes the obstacles and going home for energy supply into consideration. The details of the designed repairing algorithm executed in non-obstacle and obstacle environments are presented in the following two subsections.

3.1 No obstacle environment

In the non-obstacle environment, the robot receives repairing request packets and knows the distance from it's location to each failure region and the Home. A repairing algorithm that applies dynamic programming scheme is employed herein for the robot to construct the shortest movement path. Figure 9 shows the algorithm. Lines 1-19 are the operations that enable the robot to construct the shortest movement path. If the robot has enough energy to pass through all failure regions, it will repair the failure regions along the constructed path. As shown in Fig. 10, the shortest repairing path that passes through failure regions *A*, *B*, *C*, *D* and *E* is constructed and is denoted by the solid line. In case that the remaining energy of the robot is not enough to pass through all failure regions, the home location should be visited before the robot exhausts its energy. One simple method is to insert the home location at the constructed shortest path. However, the resultant path might be inefficient since the home location might be far away the failure regions visited before and after the home location. For example, inserting the home location between failure regions *B* and *D*, the resulting path is *C*→*A*→*B*→Home→*D*→*E*, as shown in Fig. 11(a), which results in the inefficient segments *B*→Home and Home→*D*. Lines 20-33 in Fig. 9 cope with the energy exhaustion problem and construct an optimal path that prevent the robot from energy exhaustion. Applying the proposed repairing algorithm, an optimal movement path *A*→Home→*B*→*D*→*E*→*C* is constructed, as shown in Fig. 11(b).

Path Construction Algorithm_Without Obstacle

```
     Void travel ( int n, const number W[ ], minlength, Energy(R) )
     {
1      /* n: the number of nodes ( including Home )
2        W[] : adjacency matrix
3        R: the location of Robot
4        V: set of all the nodes ( including Home and R )
5        U=all subsets of V which includes Home
6        D[vi][A] =length of a shortest path from v_i to v_1
                  passing through each vertex in A exactly once
7        Energy(R): the energy of Robot at R
8      */
9      index i, j, k
10     number D[1..n][subset of V-{R}]
11     for (i=2;i<=n,i++)
12       D[i][ϕ]=W[i][1]
13     for (k=1;k<=n-1;k++)
14       for (all subset A ⊆ V-{R} containing k vertices)
15         for (i such that i≠R and v_i is not in A)
16         {
17           D[i][A]=minimum( W[i][j]+D[j][A-{v_j}])
                      j:V_j∈A
18         }
19     minlength=minimum(D[j][V-R-Home])
                    j∈A,j≠R
20     if (Energy(R) <= minlength)
21     {
22       minlength=minimum(D[j][V-R])
23       if (Energy(R) <= minlength)
24       {
25         for (k= n-1;k>=1;k--)
26           for (all subset A ⊆ U-{R} containing k-1 vertices )
27             if (Energy(R) >= minlength)
28             {
29               minlength=minimum(D[j][V-R])
30               break
31             }
32       }
33     }
     }
```

Figure 9. Robot Repairing algorithm that considers the remaining energy of the robot but does not considers the existence of obstacles.

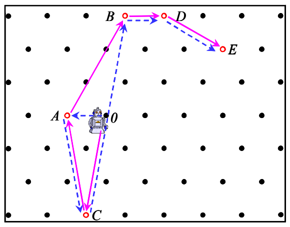


Figure 10. The comparison of the constructed paths by applying the greedy algorithm and the proposed repairing algorithm. The proposed repairing algorithm constructs a shortest path marked by the solid line.

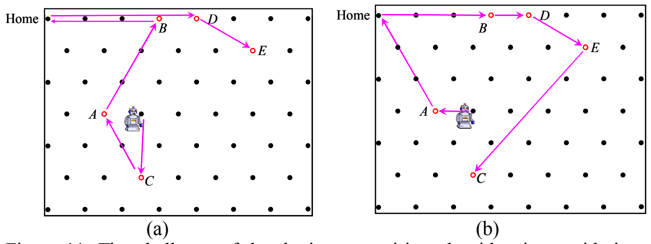


Figure 11. The challenge of developing a repairing algorithm in considering the remaining energy. The developed repairing algorithm constructs a route that prevents the robot from energy exhaustion. (a) Inserting home location in a shortest route is not a feasible solution to prevent robot from energy exhaustion. (b) The repairing algorithm takes into consideration the remaining energy of robot. The robot can timely recharge the energy before the energy is exhausted.

3.2 Obstacle environment

During the execution of deployment task [4], the robot can detect the obstacles. However, the robot is not aware the obstacles' locations and shapes. This subsection presents the path construction algorithm in considering the obstacle environment. The challenge of path construction is that the robot can not estimate the distance between any two failure regions since there might exist obstacle between them. Moreover, an improper estimation of distance might lead to the energy exhaustion of the robot. Figure 12 proposes a path construction algorithm that considers the existence of obstacles. Firstly, an optimal path is constructed by applying the algorithm designed for the non-obstacle environment. However, the distance between any two failure regions might be incorrect because of the existence of obstacle. To prevent the robot from energy exhaustion, the robot should try to estimate the distance of the two successive failure regions scheduled on the constructed route. To accomplish this, the robot initiates a probe packet before its movement, aiming at to estimate the distance of the route for avoiding the energy exhaustion problem. The probe packet will be transmitted along the constructed path and pass through all failure regions. When the probe packet arrives each request initiator, it collects information including the number of hops from the previous request initiator to the current initiator and the distance from the current initiator to the home location. Regarding the distance from the Home's location to the request initiator, it can be simply known by flooding another packet from home location over the WSN. As soon as the probe packet returns to the robot, it analyzes the collected distance information and determines the actual route according to the remaining energy. Figure 12 depicts the algorithm of probe packet transmission. If the remaining energy of the robot can only reach failure region v_i, the robot goes back Home for recharging and the segment $v_i \rightarrow$Home will be inserted in the constructed path.

Obstacle-Free Path Construction Algorithm

Input：repair path(v_0 ,v_1,v_2,…,v_{n-1}) , v_0=*location of robot*

```
if( obstacle_flag = true )
{
    transmit a probe packet to simulate the repair path
    simulate_energy = Energy($R$)
    $i$=0
    for( $i$=0 ; temp < 0 ; $i$++ )
    {
        simulate_energy = simulate_energy − energy of $v_i$ to $v_{i+1}$
        temp = simulate_energy − energy of $v_{i+1}$ to Home
    }
    insert Home behind $v_i$
    repair path = $v_0$,$v_1$,…,$v_i$, Home,$v_{i+1}$,…,$v_{n-1}$
}
```

Figure 12. Algorithm of constructing a repairing path by initiating a probe packet along the previously constructed optimal path for handling the obstacle problem.

IV. Performance Study

This section examines the performance study of the developed tracking and repairing mechanisms. The proposed tracking and robot repairing algorithm, called TRR in short, is compared with previous work in [1] which is referred to CED. Table I lists the parameters values which refer to the typical parameters in Berkeley motes.

The robot is assumed to be equipped with a compass and a constant number of Berkeley motes. The total energy and the speed of the robot are 64800J and 3m/s, respectively. The mobility cost is set by 8.267J/m which refers to previous work[5]. The experimental environment is described in below. The network size is 400*400m^2. The home is located at the left-top corner of the monitoring region and the start location of the robot is home location. Each simulation result is obtained from the average of 10 independent runs.

Table I：Simulation parameters

Parameter	**Value**	**Parameter**	**Value**
Communication range	40m	Idle cost	0.025J/s
Sensing range	20m	Total initial energy	32400J (100hr)
Packet transmission cost	0.075J/s	Maximum energy consumption in motes	324J/hr
Packet reception cost	0.030J/s		

The request initiator which is nearby the failure sensor will send a repairing request to the robot. Upon receiving the request message, the robot will perform the proposed repairing algorithm to construct a repairing path if it collects at least 5 different repair request messages in 10 minutes. In case that the robot collects fewer than 5 failure messages for 20 minutes, the robot also executes the repairing mechanism in order to prevent the request initiator from waiting for a long time.

To evaluate the benefit obtained from the proposed X-correction, two straightforward mechanisms including Flooding-correction mechanism and Line-correction mechanism are compared with the proposed X-correction mechanism in terms of the control packet overhead and the length of tracking path. In the flooding mechanism, the robot floods a correction packet over the entire WSN and all sensor nodes update their counter values according to the correction packet. In the line-correction mechanism, the y-axis of the robot's location is considered to be the path traversed by the correction packet. Nodes that lie on the y-axis will correct their counter value according to the correction packet.

We assume that the robot broadcasts the correction packet for every movement of 400m. The failure nodes are randomly selected and 5 failure nodes will be generated every 10 minutes. As shown in Fig. 13, line-correction and flooding mechanisms have smallest and largest control overheads, respectively. The control packet overhead of X-correction mechanism is approximately 2.5 times of the line-correction mechanism while the control overhead of flooding mechanism is about 37 times of X-correction mechanism. In other words, the control overheads of the line-correction and X-correction are similar and they are significantly smaller than that of the flooding mechanism.

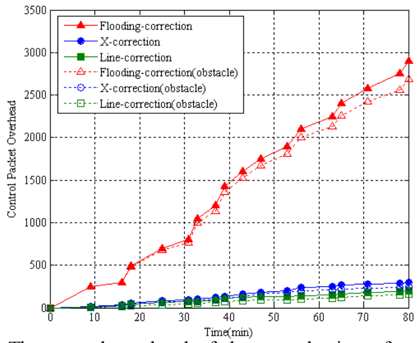


Figure 13. The control overhead of three mechanisms for correcting the counter values.

In order to evaluate the impact of applying the three correction mechanisms on the tracking length, the average length of tracking path is calculated and compared. Figure 14 examines the average distance between sensor nodes and robot. The repairing request packet sent by any sensor node can always reach the robot by traveling the shortest path by applying the flooding-correction mechanism. Although the length of tracking path by applying the X-correction mechanism is not as short as the one created by applying the flooding mechanism, however, the X-correction mechanism efficiently reduces the average length and keeps the path length smaller than 520m. As a result, the average path lengths by applying X-correction and Line-correction mechanisms are 280m and 372m, respectively. We evaluate which mechanism is the most cost-effective by using a correction efficient index which is defined by the reduced routing path divided by the number of correction packets. As shown in Fig. 14, the correction efficient indexes of X-correction and Line-correction are 17.26 and 15.75, respectively. Thus, the X-correction mechanism outperforms the Line-correction and flooding mechanisms in terms of the correction efficient index.

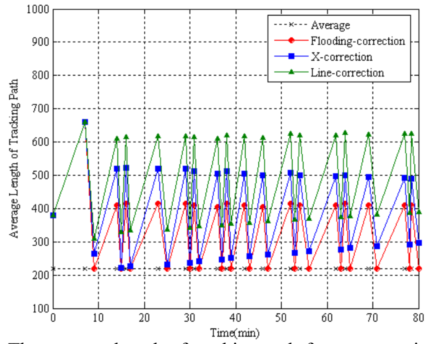


Figure 14. The average length of tracking path from request initiators to the robot by applying the three correction mechanisms.

The threshold of robot movement distance that initiates the X-correction mechanism is ranging from 200m to 500m. As shown in Fig. 15, the value of threshold set by 200m and 300m can significantly reduce the average length of tracking path. When the threshold values are 400m and 500m, the improvement of path length is not significant. Therefore, the X-correction mechanism will use 400m as the threshold value in the repairing task to compare with the CED mechanism [1].

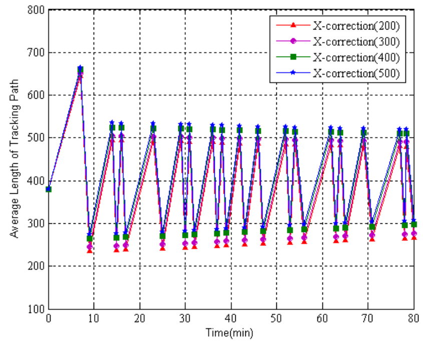


Figure 15. The average length of tracking path by varying the threshold value of movement distance.

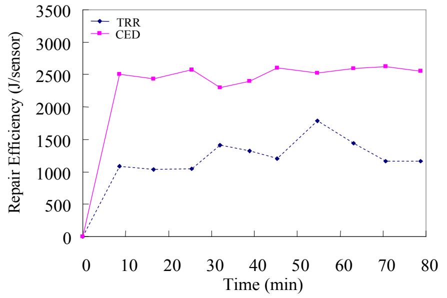


Figure 16. Comparison of TRR and CED in terms of the repairing efficiency.

TRR additionally reduces the energy consumption of the robot. When the robot receives multiple repairing request messages, it constructs a shortest route that passes through all failure regions. Let repairing efficiency be measured by the average energy consumption required for robot to repair a failure region. Since the robot moves along the shortest route, the TRR saves robot's energy and has a better repairing efficiency than CED, as shown in Fig. 16.

Finally, we compare the TRR and CED in terms of the coverage ratio. The proposed TRR enables request initiators actively notify the robot to repair the failure region, resulting the failure region can be rapidly redeployment within 10 minutes. Therefore, TRR maintains 98% of coverage radio. However, CED passively waits the robot to repair the failure region, and hence some existing holes can not be repaired in time. As shown in Fig. 17, TRR has a better coverage ratio than CED.

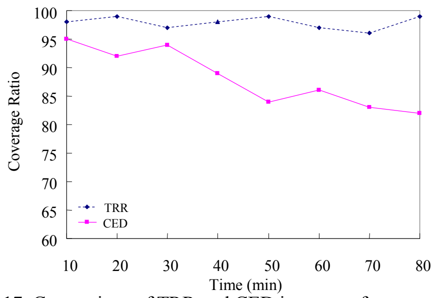


Figure 17. Comparison of TRR and CED in terms of coverage ratio.

V. Conclusions

This paper proposes a tracking mechanism and a robot repairing mechanism for maintaining coverage quality of a given WSN. With the extension of the existing robot deployment algorithm [4], the robot leaves virtual coordinates and counter value on each deployed sensor as a footmark for tracking the robot. After the robot completes its deployment task, it moves for handling the event and repairing the failure regions. Since the robot changes its location, an X-correction mechanism is proposed for the robot to update the counter values of some sensors in a cost-effective manner. Consequently, the request initiator learns a better route to notify robot for repairing. Upon receiving the location of failure regions from different request initiators, the robot dynamically constructs a shortest route that passes through each failure region for executing the redeployment task so that the coverage quality of the given WSN can be maintained. The route construction also takes obstacle and constraint of robot's remaining energy into consideration. Performance results reveal that the proposed tracking and repairing algorithms outperform existing mechanism in terms of repair efficiency and coverage ratio.

References

[1] M. A. Batalin and G. S. Sukhatme, "Efficient Exploration without Localization," *Intl. Conference on Robotics and Automation* (*ICRA*), pp. 2714–2719, Taipei, Tanwan, May 2003.

[2] M. A. Batalin and G. S. Sukhatme, "Coverage, Exploration and Deployment by a Mobile Robot and Communication Network," *Proceedings of the International Workshop on Information Processing in Sensor Networks*, pp. 376-391, Palo Alto Research Center (PARC), Palo Alto, Apr 2003.

[3] Y. Zou and K. Chakrabarty, "Sensor Deployment and Target Localization in Distributed Sensor Networks," *ACM Transations on Embedded Computing Systems*, vol. 3, pp.61-91, 2004.

[4] C. Y. Chang, H. R. Chang, C. C. Hsieh and C. T. Chang, "OFRD: Obstacle-Free Robot Deployment Algorithms for Wireless Sensor Networks," *IEEE WCNC*, Hongkong, March 2007.

[5] S. Ganeriwal, A. Kansal and M. B. Srivastava, "Self Aware Actuation for Fault Repair in Sensor Neworks," *IEEE International Conference on Robotics and Automation* (ICRA), Apirl 2004.