A Reinforcement Learning Method to Improve the Sweeping Efficiency for an Agent

Naoto Ohsaka
Department of Computer Science
The University of Electro-Communications
Chofu, Japan
Email: no911121@jed.uec.ac.jp

Daisuke Kitakoshi and Masato Suzuki
Department of Computer Science
Tokyo National College of Technology
Hachioji, Japan
Email: {kitakosi, suz}@tokyo-ct.ac.jp

***Abstract*—This article proposes a reinforcement learning method aimed at improving the sweeping efficiency of an agent. In the proposed method, an agent attempts to avoid overlapping a swept field by using a combination of distances from the agent to obstacles and the information which expresses whether a field in front of the agent has already been swept. We carried out several computer simulations to evaluate basic characteristics and performance of the proposed method. The empirical results showed that the agent behaves effectively in the field compared to an agent with fixed heuristics.**

***Keywords*-reinforcement learning; sweeping task; virtual robot;**

I. Introduction

In recent years, domestic robots such as cleaning robots and entertainment robots have become familiar objects. In the case of cleaning robots, each robot has a specific task to accomplish. A cleaning robot wipes dust off a floor or removes trash, while moving about a room. We call such behavior "*sweeping*", in which a robot covers a 2-dimensional field (defined as the *working field*). Sweeping can be applied to cleaning a room, painting a floor, mowing a lawn, searching for objects, and so on.

In sweeping tasks, an agent aims to sweep the whole working field once or more. It is desirable that the agent avoid overlapping a swept field. Sweeping tasks is generally classified as off-line and on-line. In off-line sweeping tasks, a map of the environment is assumed to be given to agents, which can also generally use high-accuracy sensors (e.g., GPS). The agent can therefore plan its path in advance. Some path planning algorithms for off-line sweeping tasks have been developed [2], [5], [10]. Kurabayashi et al. [6], [7] proposed an algorithm for off-line sweeping task for multiple robots. However, it would be significantly difficult for the agent to make use of the map in the environment since the map information is actually unknown in many situations. In on-line sweeping tasks, the map is not given to an agent. Therefore, the agent uses sensors to acquire information about the environment and sweeps the working field. Whereas, because agents (e.g., domestic cleaning robots) sometimes have only low-accuracy sensors, they may not decide their location precisely. Various algorithms for on-line sweeping task also have been proposed. Gabriely and Rimon [3] proposed on-line sweeping algorithm known as spanning-tree coverage (STC) for a single robot, and Hazon et al. [4] extended STC for use in a multi-robot environment. In addition, Acar and Choset [1] proposed a robust on-line sweeping algorithm.

In this article, we propose a reinforcement learning method *Learning for Controlling Redundant Sweeping* (*LCRS*) for on-line sweeping task with low-accuracy sensors for a single agent. Reinforcement learning is a kind of machine learning to adapt to an environment. LCRS improves its sweeping efficiency by introducing the following features: 1) an agent takes advantage of a series of its input-output information to precisely make decisions in the environment, and 2) it decides whether it should sweep a field in front of it based on the sweeping condition in the front field. In addition, LCRS may allow agents to adapt to various environments. Several computer simulations are carried out to evaluate the basic characteristics and performance of LCRS.

II. Sweeping Task Planning Problem

In sweeping task, an agent aims to sweep the whole working field (e.g., living room in our house, workplace in industrial establishments, etc.) once or more. A working field has several obstacles. It is also desirable that the task completes in a short period of time. In on-line sweeping task with low-accuracy sensors, the sweeping task which we focus on, it is hard for agents to completely sweep the working field because the agents cannot make planning of its path in advance and cannot make a decision about its own location in the environment precisely.

Sweeping Task Planning (*STP*) Problem is defined as a problem to maximize an area of swept field as quickly as possible. In the STP described in this article, we focus on not hardware performances (i.e., agents' moving speed, and type, performance, and the number of sensors which the agents has) of agents but sweeping efficiency affected by sweeping algorithms incorporated into the agent. In order to evaluate the sweeping efficiency properly, we set some preconditions to exclude the hardware performances of an agent. First, hardware performances described above are constant. Second, a term "sweeping field" is defined to

978-1-4577-0371-3/11/$26.00 ©2011 IEEE

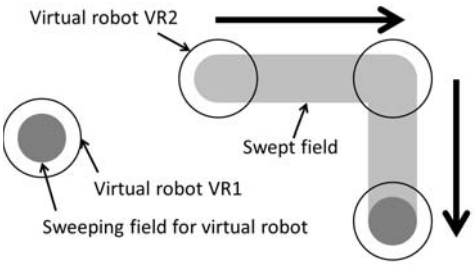

Figure 1. A sweeping field for virtual robot

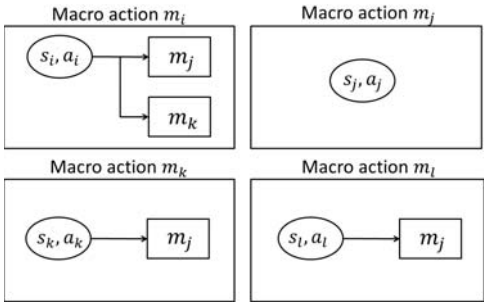

Figure 2. Macro actions

remove the impact of agent's garbage suction performance from this problem. Sweeping field is the field where an agent should suction all of garbage. In Fig. 1, a gray field inside virtual robot VR1 represents sweeping field. In addition as shown in the right half of the figure, a field colored light gray means a virtual robot VR2's moving trajectory and also depicts swept field for VR2. In order to evaluate sweeping efficiency, sweeping rate S_t at t is defined as follows:

$$S_t = \frac{(\text{Area of swept field at } t)}{(\text{Area of working field})} \times 100[\%]. \quad (1)$$

This article assumes that an agent corresponding to a domestic cleaning robot has low accuracy sensors. Such the agent therefore may not be able to acquire complete information (such as its absolute position), and besides in real-world environments, information which the agent acquires is uncertain due to sensory noise. In such environments, the agents may not learn appropriate behavior (perceptual aliasing problem [9]). To overcome a part of this problem, taking advantage of a series of its input-output information regarding an agent's state and action is considered to be an effective way to locate itself in the environment.

III. LCRS

This section describes main components of LCRS. First, we define the agent's state and settings of reward specialized for improving sweeping efficiency. Second, a macro action is introduced as a data structure to represent agent's behavioral sequence. After that, we describe reinforcement learning used as a learning method for agent's behavior.

A. Definition of agent's state and setting of reward

In this article, we introduce a parameter α_t in order to avoid overlapping swept field. α_t is defined as a binary parameter expressing whether a field in front of an agent is already swept ($= 1$) or not ($= 0$). A field in front of the agent is defined as the field where the agent would sweep when it moves forward. The value of α_t is determined by a parameter S'_t expressing sweeping rate in a field in front of the agent. An agent's state at t is defined as a combination of distances from the agent to obstacles and α_t. The agent observes the distances with distance sensors and then determines the value of α_t as follows:

$$\alpha_t = \begin{cases} 1 & S'_t \geq \theta \\ 0 & \text{otherwise} \end{cases}, \quad (2)$$

where θ is a threshold having impact on agent's "activeness". If θ would take small value, the agent tends to behave optimistically, that is, it does not care even if a certain amount of field is not swept yet. On the other hand, if θ would take large value, the agent may be pessimistic, or, it considers having to sweep completely if even a bit of its front field is not swept.

A reward r_t which an agent acquires at t is defined as:

$$r_t = \frac{\Delta S_t - \Delta S_{max}/2}{\Delta S_{max}/2}, \quad (3)$$

where $\Delta S_t = S_t - S_{t-1}$, and ΔS_{max} is the maximum possible increment of S_t during a unit time step. The agent aims at maximizing the total discounted reward. (3) means that an area of field where the agent swept in a unit time step is normalized to $[-1, 1]$. For example, if the agent completely would not overlap already swept field, it would acquire reward 1, and it would acquire reward -1 if the whole field would overlap the swept field.

B. Macro action

In LCRS, a macro action is used as a data structure expressing a series of agent's input-output (state-action) information properly. First, we define macro action, and then application in this article is described. After that, description of how to generate macro actions is given.

1) Definition of macro action: A macro action m is defined as

$$m = ((s, a), M), \quad (4)$$

where (s, a) is a rule, which states "if an agent observes state s, it selects action a" and M is a set of macro actions of which size $|M|$ is equal to or greater than 0. The first and the second components of the rule (s, a) described in (4) are referred to as state and action parts, respectively. The followings are examples of macro action:

$$(a)\ m_i = ((s_i, a_i), \{m_j, m_k\}),\ (b)\ m_j = ((s_j, a_j), \phi), \quad (5)$$

$$(c)\, m_k = ((s_k, a_k), \{m_j\}),\ (d)\, m_l = ((s_l, a_l), \{m_j\}). \quad (6)$$

A macro action can be expressed as a tree structure. The tree structures corresponding to the above 4 macro actions are illustrated in Fig. 2. The length of a macro action $|m|$ is defined as

$$|m| = \max_{m' \in M} |m'| + 1. \quad (7)$$

For example, $|m_i|$, $|m_j|$ and $|m_k|$ in Fig. 2 are equal to 3, 1 and 2, respectively.

2) Application of macro action: In the application of macro actions, t is incremented every time one action is executed. For example, when an agent observes a state s, macro actions are applied according to the following steps:

Step 0. The agent prepares a sufficient number of simple macro actions corresponding to each state at the first stage of learning.

Step 1. It selects one macro action m out of candidates of which state part equals s stochastically.

Step 2. It outputs an action a described in the action part of m. Then the state s changes to s'.

Step 3. If $s' = s$, m is regarded as being selected again, return to step 2.

Step 4. Otherwise, the agent searches macro action of which state part equals s' from a group of macro actions corresponding to the elements of M (the elements of M includes at most one macro action of which action part equals s').

Step 5. If such macro action m' was found as a result of searching, the agent selects m' and then return to step 2. Otherwise, it selects one macro action out of a set of macro action of which state part equals s', $M(s') = \{m | m = ((s', a), M)\}$ stochastically, then return to step 2. The agent selects a macro action m corresponding to s with the following probability:

$$P(m = ((s, a), M)|s) = \frac{\exp(Q(m)/\tau)}{\sum_{m' \in M(s)} \exp(Q(m')/\tau)}, \quad (8)$$

where τ is referred to as temperature parameter, and $Q(m)$ is referred to as the value of macro action m. After the selecting, $Q(m)$ is updated in accordance with the equation described in the next subsection.

Agents having various macro actions may adapt to a variety of environments effectively. However, in unknown environment, they cannot prepare such macro actions in advance. The agent prepares macro actions of which length equals 1 at the first stage of learning, and then generates various macro actions according to the following manner.

3) Macro action generation: For ease of description, time step for macro action u is introduced. The value of u is incremented when an agent updates macro action value, and may correspond to one or more unit time step. An agent's *experience* at u, E_u, is defined as

$$E_u = (m_u, k_u, R_u), \quad (9)$$

Table I
EXAMPLES OF EXPRIENCE

u	t	E_u
0	0,1,2	$(m_i, 3, R_0)$
1	3,4	$(m_l, 2, R_1)$

where m_u is a macro action selected at u, k_u is the number of steps corresponding to u, and R_u is the total discounted reward acquired at u described in the next subsection. An agent's *history* at u, H_u, is defined as

$$H_u = \{E_u, E_{u-1}, \cdots, E_{u-N+1}\}, \quad (10)$$

where N is the limit of the number of experiences. An experience is added to the history every time u is incremented. For example, when the agent having 4 macro actions shown in Fig. 2 observes states s_i, s_k, s_j, s_l and s_j in that order, two experiences are stored as shown in Table I.

In order to generate various macro actions, the following procedure is executed every N macro action time steps.

Step 1. Select two experiences, $E_{u_1} = (m_1, k_1, R_1)$ and $E_{u_2} = (m_2, k_2, R_2)$, from H_u at random, and generate macro action $m_3 = ((s_1, a_1), \{m_2\})$.

Step 2. $Q(m_3)$ is estimated as follows:

$$\begin{aligned} Q(m_3) &= \Sigma(R_j) + \gamma^{\Sigma(k_j)}\Sigma(R_{j+1}) + \\ &\quad \gamma^{\Sigma(k_j + k_{j+1})}\Sigma(Q(m_{j+2})), \end{aligned} \quad (11)$$

$$\Sigma(f(j)) = \frac{\sum_{0 \le j < N-2} n(j) \times f(j)}{\sum_{0 \le j < N-2} n(j)}, \quad (12)$$

$$n(j) = \begin{cases} 1 & m_j = m_1 \ \text{ and } \ m_{j+1} = m_2 \\ 0 & \text{otherwise} \end{cases}, \quad (13)$$

where γ ($\in [0, 1]$) is referred to as discount rate. The agent decides whether m_3 is more effective than m_1 and m_2 in accordance with the following condition:

$$Q(m_3) > \beta_1 Q(m_1) + C_1, \quad (14)$$

$$Q(m_3) > \beta_2 Q(m_2) + C_2, \quad (15)$$

where $\beta_1, \beta_2, C_1, C_2$ are parameters used for macro action generation ($\beta_1, \beta_2, C_1, C_2 > 0$). If these conditions are satisfied, go to step 3, otherwise return to step 1.

Step 3. m_1 is combined with m_2 ($m_1 \leftarrow ((s_1, a_1), M_1 \cup \{m_2\})$) if the following condition is satisfied:

$$\forall((s, a), M) \in M_1 \ \ s \neq s_2, \quad (16)$$

otherwise m_3 is added to $M(s_1)$ ($M(s_1) \leftarrow M(s_1) \cup \{m_3\}$).

In the above procedure, an agent can generate macro actions infinitely. Macro actions, which may not contribute to the appropriate behavior of the agent, accordingly multiply unlimitedly. Consequently, adaptability to an environment may decline. For this reason, we assign the limit length of macto action L_{max} so that the structures of macro

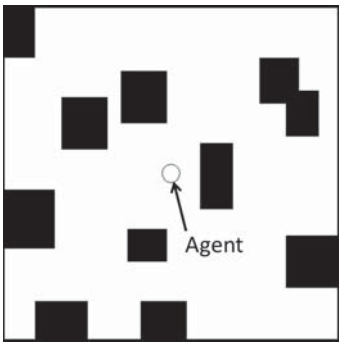

Figure 3. A simulation environment and the initial position of agent

action do not become too complex. When the length of combined or generated macro action is greater than L_{max}, the correspondent macro action is removed.

C. Reinforcement learning

Reinforcement learning (RL) [8] is a kind of machine learning to adapt to an environment based on the concept of dynamic programming. An RL agent (e.g., mobile robot) attempts to acquire an appropriate policy based on its observations and trial-and-error interactions with its environment. A policy characterizes an agent's behavior and is defined as

$$\pi : S \times A \to [0, 1], \tag{17}$$

where S and A denote sets of states and actions, respectively. $\pi(s, a)$ represents a probability that an agent selects a rule (s, a). At each time step t, an agent observes state s_t, and selects action a_t using π. In response to a_t, the agent acquires a reward r_{t+1} and the state is changed to s_{t+1}. The total discounted reward (called return) at t is defined as:

$$R_t = \sum_{k=1} \gamma^{k-1} r_{t+k}. \tag{18}$$

The aim of the agent is to acquire the policy that maximizes the return.

In LCRS, one of the RL methods, sarsa, is employed as the policy learning method. In addition, an agent selects a macro action instead of an action at each time step. When an agent selects a macro action m at t and m' is selected at $t+T$ after the application of m terminates, $Q(m)$ is updated as follows:

$$\begin{aligned} Q(m) \quad \leftarrow \quad & (1-\alpha)Q(m) + \\ & \alpha \left[\sum_{k=1}^{T} \gamma^{k-1} r_{t+k} + \gamma^T Q(m') \right], \end{aligned} \tag{19}$$

where α ($\in [0, 1]$) is referred to as learning rate.

IV. Computer simulations

This section describes computer simulations in STP to evaluate basic characteristics and performance of LCRS.

Table II
PARAMETERS

Parameter	Value	Parameter	Value
α	0.1	C_1	4
γ	0.9	C_2	4
τ	0.5	β_1	1
N	100	β_2	1
L_{max}	8		

Table III
SWEEPING ALGORITHMS

Sweeping algorithm	Definition of state	Macro action
sarsa_s	$\surd$	X
sarsa_m	X	$\surd$
sarsa	X	X

$\surd$: Same as LCRS X : Simplified

Fig. 3 shows an example of simulation environments. The field is square, surrounded by perimeter walls, and besides has 11 random sized obstacles. These obstacles are placed in random positions on the environment. An agent (virtual robot) can detect obstacles and walls within a distance of 60 pixels in three directions (front, left, and right) as well as "close" ones within a distance of 20 pixels, and also can detect whether its front field is already swept or not. Combinations of the pieces of information that the agent detects are therefore characterized as 54 states. The agent takes one of three actions, go forward, rotate left and right. The agent rotates 45° every time it selects the actions rotate left or right. In LCRS, the agent decides one rule (or one macro action) based on (8) when the application of current macro action is terminated, or when no macro actions are applied. It should be noted that the time is incremented every time on rule is executed (not one macro action). Table II shows learning parameters regarding LCRS. An episode is defined as a period from beginning of sweeping task to its end. At the beginning of an episode, all swept field is reset to unswept and an agent is placed at the center of working field as shown in Fig. 3. When S_t reaches 80%, we assume that the agent's sweeping task terminates successfully and one episode ends. In this article, the termination of one episode is called "task completion". The number of episodes is therefore incremented after the task completion.

We carry out three experiments. First, the values of threshold θ having impact on sweeping efficiency are discussed in order to investigate basic characteristics of the proposed method. We fix θ as $0, 5, \ldots, 100$. Second, we prepare two variants of LCRS and then compare those and original sarsa with LCRS to investigate which components in the proposed method affect the sweeping efficiency. These two variants are referred to as sarsa_s (sarsa using sweeping rate), and sarsa_m (sarsa with macro action), respectively. The comparison in characteristics of LCRS with the variants and sarsa is represented in Table III. As shown in this table, the agents with sarsa_m cannot detect whether its front field is already

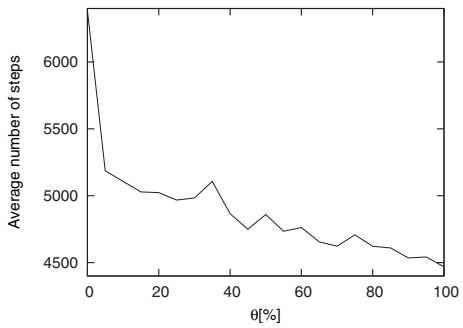

Figure 4. Relationships between the average number of steps and θ

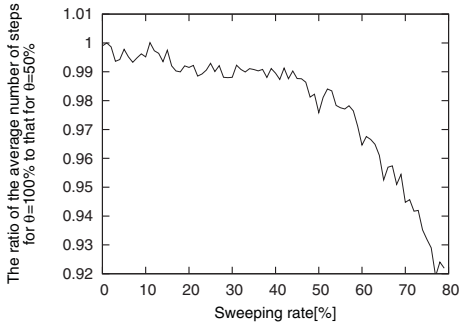

Figure 5. The ratio of the average number of steps for $\theta = 100\%$ to that for $\theta = 50\%$

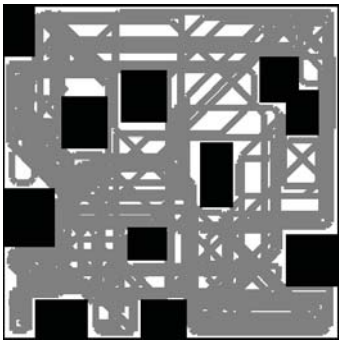

Figure 6. Working field when the task completes in a typical episode

swept or not. The agents with sarsa_s apply sarsa to each rule, without using macro action. Third, we compare the proposed method with another approach (called Heuristics) where the agent behaves according to some kind of behavioral policy based on heuristics, predefined detection-action rules (e.g., if the agent detects an obstacle in its front, it rotates left and otherwise it moves forward).

V. Results and Discussions

Ten trials were carried out for each experiment. We averaged the result for the 10 trials. Fig. 4 shows relationships between the average number of steps for task completion from episode 100 to 199, in which the agent was able to

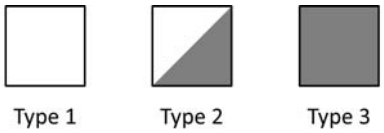

Figure 7. 3 types of an agent's front field

Table IV
Average number of steps for four sweeping algorithms

Sweeping algorithm	Average number of steps
LCRS	4490
sarsa_s	6000
sarsa_m	6360
sarsa	7320

acquire a stable policy, and threshold θ. In this figure, the average number of steps decreased with the value of θ. When θ took maximum value ($= 100\%$), the average number of steps took minimum value ($= 4470$). Now, we discuss a difference between agent's behavioral characteristics with large and small θ. Fig. 5 shows the ratio of the average number of steps to complete the task for $\theta = 100\%$ to that for $\theta = 50\%$ until task completion. In the first half of sweeping task, or if there exist still many unswept field $S_t = 0\% \sim 40\%$, the ratio is almost equal to 1, i.e., the average number of steps for $\theta = 50\%$ and 100% are almost the same. An agent can sweep unswept field even if it behaves randomly in such working field. It seems that θ does not impact on the efficiency in the first half of sweeping task. In the second half of sweeping task, the ratio decreases with S_t. This is due to fragmentation and scattering of unswept field. Fig. 6 shows working field when the task completes in a typical episode. This figure indicates that there are many fragmented unswept fields all over the environment. In the above environment, an agent's front field can be classified roughly into 3 types as shown in Fig. 7. A field of which type equals 1 and 3 can be regarded as unswept and swept, respectively, regardless of the value of θ. At the same time, a field type 2 can be regarded as both swept and unswept, depending on the value of θ. In addition, the ratio for the field of type 2 to whole working field increases with S_t. Consequently, when θ is small, an agent can hardly detect unswept field. In contrast, when θ is large, an agent may be able to sweep a field of type 1 and also type 2. Although the agent with large θ may overlap a lot of already swept field, it can have more opportunities to detect fragmented small unswept field inside the swept ones. Because assigning θ to large value thus leads to the agent's effective behavior, we fixed θ as 100% hereafter.

Second, Table IV shows the average number of steps for task completion from episode 100 to 199 for 4 sweeping algorithms. Both the average number of steps for sarsa_s and sarsa_m were less than that for sarsa. We briefly discuss a difference between the behavior of agent with sarsa and

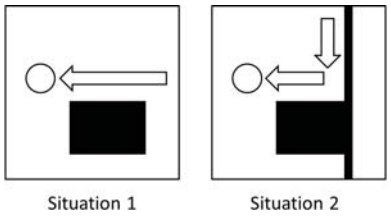

Figure 8. Two situations in the working field

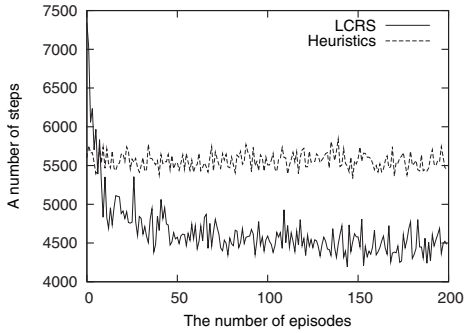

Figure 9. Transitions in the number of steps

sarsa_m. Fig. 8 shows a part of the working field. Assume that the agent follows the paths shown in this figure. States of the agent in situations 1 and 2 are eventually the same. The agent with sarsa may select the same action in both situations. However, rules the agent selected in the situations are different, the agent with sarsa_s therefore can select different action. In the working field in Fig. 3, an agent tends to encounter such situations, and therefore the agent with sarsa_m behaves more effectively than that with sarsa.

Third, Fig. 9 shows transitions in the number of steps. The average number of steps for Heuristics from episode 0 to 199 was 5560; in contrast, that for LCRS was 4490 and moreover decreased with the number of episodes. The standard deviation of the number of steps for LCRS (= 100) was larger than that of Heuristics (= 140). In the episodes such that the number of step was relatively large, the agent with LCRS has swept the same field repeatedly. The reason for this result is that the field around the agent is completely swept. As mentioned above, the agent tends to avoid sweeping a field completely swept. When the field around the agent is completely swept, the agent cannot detect unswept field and therefore it may sweep the same field repeatedly.

VI. Conclusions

This article presents a reinforcement learning method LCRS to improve the sweeping efficiency of an agent. In LCRS, sweeping rate is incorporated into the agent's state so that it can avoid overlapping a swept field. The empirical results indicate that an agent using the proposed method behaves more effectively than that with fixed heuristics.

Three future projects are being planned. First, we have to develop a method that will adjust the value of θ to a sweeping condition around an agent dynamically. Second, applying LCRS to multi-agent environments is also required. Our third project is to apply these agents in a real-world environment. In real-world environments, a robot may not detect whether its front field has already swept due to errors in the robot's motor control and sensor noise. This is a potential problem we will be addressing. We plan to introduce stochastic method into LCRS to overcome this problem.

References

[1] Ercan U. Acar and Howie Choset, "Robust Sensor-based Coverage of Unstructured Environments," In Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems, vol.1, pp.61-68, 2001.

[2] Howie Choset and Philippe Pignon, "Coverage Path Planning: The Boustrophedon Decomposition," In International Conference on Field and Service Robotics, 1997.

[3] Yoav Gabriely and Elon Rimon, "Spanning-Tree Based Coverage of Continuous Areas by a Mobile Robot," Annals of Mathematics and Artificial Intelligence, vol.31, pp.77-88, 2001.

[4] Noam Hazon, Fabrizio Mieli and Gal A. Kaminka, "Towards Robust On-line Multi-Robot Coverage," In Proceedings of the IEEE International Conference on Robotics and Automation, vol.31, pp.1710-1715, 2006.

[5] Christian Hofner and Gunther Schmidt, "Path Planning And Guidance Techniques For An Autonomous Mobile Cleaning Robot," In Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems. vol.1, pp.610-617, 1994.

[6] Daisuke Kurabayashi, Jun Ota, Tamio Arai and Eiichi Yoshida, "Cooperative Sweeping by Multiple Mobile Robots," In Proceedings of the IEEE International Conference on Robotics and Automation, vol.2, pp.1744-1749, 1996.

[7] Daisuke Kurabayashi, Jun Ota, Tamio Arai, Shinpei Ichikawa, Shingo Koga, Hajime Asama and Isao Endo, "Cooperative Sweeping by Multiple Mobile Robots with Relocating Portable Obstacles," In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, vol.3, pp.1472-1477, 1996.

[8] Richard S. Sutton and Andrew G. Barto, "Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning series)," The MIT Press, 1998.

[9] Steven D. Whitehead and Dana H. Ballard, "Learning to Perceive and Act by Trial and Error," Machine Learning, vol.7, no.1, pp.45-83, 1991.

[10] Alexander Zelinsky, Ray A. Jarvis, J.C. Byrne and Shin'ichi Yuta, "Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot," In Proceedings of International Conference on Advanced Robotics, pp.533-538, 1993.