# Generation of search behavior of robots by an extended probabilistic flow control

Ryuichi Ueda, *Chiba Institute of Technology*

***Abstract*—The probabilistic flow control method (PFC) can generates behaviors of robots that compensate information uncertainty in POMDP (partially observable Markov decision process) problems. We improve PFC and verify it on an actual robot in this paper. The original PFC is a modification of the $Q_{\mathrm{MDP}}$ value method. The $Q_{\mathrm{MDP}}$ value method guides a robot based on the expected values of cost reduction under the probability distribution of state estimation. PFC biases the expected value "optimistically." This bias makes a robot behave as if searching a goal of a task. In this paper, we make the intensity of bias adjustable. With appropriate parameters of the intensity, we find that robots behave more effectively than the original PFC method. Then the improved PFC method is implemented on an actual mobile robot that has poor self-localization ability. The robot shows goal search behavior that compensates the uncertainty of self-localization with the improved method.**

***Index Terms*—probabilistic flow control, uncertainty compensation, Bayes estimation, optimal control**

## I. Introduction

A lack or uncertainty of information is an unfavorable thing to robots and creatures. However, it is interesting that the uncertainty produces a huge variety of behaviors of creatures. For example, a person who wants to go to his/her bedroom in darkness may walk along a wall carefully by checking the location with his/her hands. As another example, we can grasp a can on a table without looking at it by sweeping the table with a hand.

In robotics, a good example of this behavior can be seen in the work of Roy et al. [1]. By using the method named the coastal navigation, a mobile robot with a range sensor goes to a destination along walls so as not to lost its location. This method is a solution of a problem in partially observable Markov decision processes (POMDPs) [2]. We can also find some behaviors of robots that deal with uncertainty in the studies of POMDPs [3], [4], [5].

As a solver of POMDPs, we have also proposed the probabilistic flow control method (hereinafter called PFC). As mentioned later, this method generates behaviors of a mobile robot that compensate a lack of information [6]. PFC enables robots to perform tasks that demand more accuracy of information than that provided by their sensors. For example, it is usable to generate adjustment motion of a robot after it cannot stop a target position. PFC uses a modified formula of the $Q_{\mathrm{MDP}}$ value method (hereinafter called $Q_{\mathrm{MDP}}$) proposed in [7] by Littman *et al.* It was a discovery that the behavior of a robot changed drastically though the modification was small.

In this paper, we tackle with the following two issues. The former is an improvement and generalization of PFC. We introduce a parameter that unifies $Q_{\mathrm{MDP}}$, PFC and its extensions. This improvement increases the cases where PFC works effectively. The latter is to verify that PFC and its extensions work on an actual robot. In our previous works, PFC is only verified on simulations. We observe the phenomena caused by PFC and its extensions in the actual world.

The contents of this paper are as follows. Related studies are introduced in Sec. II. A problem that belongs to POMDPs is given in Sec. III. The extended PFC is applied to the problem in Sec. IV. We have experiments in a simulator and an actual environment respectively in Secs. V and VI. We conclude this paper in Sec. VII.

A part of this work has been already published in our conference paper [8]. In this previous paper, a manipulator is simulated with the extended PFC. In this paper, the extended PFC is precisely defined and applied to an actual mobile robot. We discuss the characteristics of the proposed method with the simulation and experimental results in [8] and this paper.

## II. Related Works

The main issue on POMDPs is how to handle a huge search space of a system [9]. A decision making method for POMDPs must provide a map $\Pi: \mathcal{B} \to \mathcal{A}$ to a robot. The set $\mathcal{B}$ contains all patterns of how the robot recognizes the state of the system. The pattern is usually represented by a belief state, which is a probability distribution in the state space. $\mathcal{B}$ becomes an uncountable set that contains all patterns of the probability distributions. $\mathcal{A}$ is a set of available actions. $\Pi$ is called *a policy*. It is a map from a belief state in $\mathcal{B}$ to an action in $\mathcal{A}$.

In such a huge space as $\mathcal{B}$, search methods are effective for obtaining a partial policy [10], [5]. In search methods, a finite number of nodes are chosen from $\mathcal{B}$. They are connected from a current state of the robot to a goal.

When we want to calculate beforehand a global policy that answers an action in response to any belief state, we need to use something other than search methods. For this case, AMDP (augmented Markov decision process) methods [11] are available. The coastal navigation [1] mentioned in the introduction is classified in this approach. In AMDP methods, the dimension of $\mathcal{B}$ is reduced for calculation within a time limit. The probability distribution of a belief is parameterized and $\mathcal{B}$ is represented by some variables. In [12], $\mathcal{B}$ is reduced by sampling of belief states and used the value iteration algorithm [13], [14] for planning. In [15], [16], [17], sampling of belief states is done in the middle of value iteration.

In both cases of search methods and AMDP methods, how to define a state transition model in a belief space is a problem.

This work is supported by JSPS KAKENHI Grant Number 20K04382.

Ryuichi Ueda is with Department of Advanced Robotics, Faculty of Advanced Engineering, Chiba Institute of Technology, 2-17-1 Tsudanuma, Narashino, Chiba, Japan (ryuichi.ueda@p.chibakoudai.jp)

If we can define it in $\mathcal{B}$, a POMDP problem can be solved as a Markov decision making (MDP) problem. In general, however, when or where information for state estimation can be obtained is unknown. The coastal navigation is an exceptional case where the statistical nature of range sensors in each belief state is relatively predictable.

As a different approach, a decomposition of a POMDP problem into the following sub-problems is worth considering:

- an offline planning problem in which a policy is obtained under the assumption that a state is certainly known, and
- an online decision making problem in which appropriate actions are solved under uncertainty based on the policy obtained in the offline problem.

In short, we do not care the uncertainty at the offline phase, whereas it is considered when the robot is working. Though this approach can be applied only to a subset of POMDPs, a certain number of problems on robotics belong to this category.

We handle this decomposed POMDP problem in this paper. As a solver of this problem, $Q_{\mathrm{MDP}}$ has been proposed by Littman *et al.* three decades ago [7]. Then we have effectively used $Q_{\mathrm{MDP}}$ in real-time environments of robot soccer with particle filters [4], [18]. $Q_{\mathrm{MDP}}$ cannot perform multi-step decision making for dealing with uncertainty. It only gives one step reflective action. In [6], however, we have generated multi-step decision by PFC, which is a small modification of $Q_{\mathrm{MDP}}$.

In state-of-the-art research, more complex problems are discussed. When there are more than one agents in a system, the problem is referred to as Dec-POMDP (decentralized POMDP) [19]. If conflicts of interest exist among agents, the problem belongs to a more difficult category called Partially Observable Stochastic Games (POSGs) [20]. These problems are too difficult to solve with a general method. A problem in these classes may be solved based on special conditions of the problem or may be solved with combinations of methods in the lower classes as POMDPs or MDPs. In those cases, there is a possibility that the methods discussed in this paper can be used as a part of the solver.

## III. Problem Definition

### A. *A system with state uncertainty*

We assume a time-invariant system in which a robot decides its action for finishing a task. A state of the system is represented as $\boldsymbol{x}$. We define a set of states: $\mathcal{X}$ and a set of final states: $\mathcal{X}_{\mathrm{f}} \subset \mathcal{X}$. The state $\boldsymbol{x}$ is not directly observable from the robot. The robot must estimate it through its perception. However, there is an exception that the robot is notified or can sense whether the task is finished ($\boldsymbol{x} \in \mathcal{X}_{\mathrm{f}}$) or not.

We define a set of actions: $\mathcal{A} = \{a_1, a_2, \dots, a_h\}$. The number of actions $h$ is finite. A robot can choose one of them at each time step $t = 0,1,2,\dots,t_{\mathrm{f}} - 1$. Here $t = 0$ and $t_{\mathrm{f}}$ are the start and the end time steps of the task respectively. $t_{\mathrm{f}}$ is not fixed.

When an action $a(t)$ is chosen by the robot at time $t$, the state is changed from $\boldsymbol{x}(t)$ to $\boldsymbol{x}(t+1)$. Since we handle a time-invariant system, the concrete value of $t$ is not important in many cases for discussion. In that case, the three symbols $\boldsymbol{x}(t), a(t)$ and $\boldsymbol{x}(t+1)$ related to a state transition are written as $\boldsymbol{x}, a$ and $\boldsymbol{x}'$ respectively.

We assume that each state transition is noisy. If a state transition occurs from $\boldsymbol{x}$ by $a$, the posterior state $\boldsymbol{x}'$ is different in each case. We assume that we can know the tendency of differences as the following probability distribution:

$$p(\boldsymbol{x}' \mid \boldsymbol{x}, a),$$

which is called the state transition pdf (probability distribution function). We abbreviate this function or its value as $p_{\boldsymbol{x}\boldsymbol{x}'}^{a}$.

### B. *Control*

The purpose of the task is given by the following summation:

$$J[\boldsymbol{x}(0), a(0), \boldsymbol{x}(1), a(1), \dots, \boldsymbol{x}(t_{\mathrm{f}} - 1), a(t_{\mathrm{f}} - 1), \boldsymbol{x}(t_{\mathrm{f}})]$$
$$= V(\boldsymbol{x}(t_{\mathrm{f}})) + \sum_{t=0}^{t_{\mathrm{f}}-1} r_{\boldsymbol{x}\boldsymbol{x}'}^{a}, \qquad (1)$$

where $r_{\boldsymbol{x}\boldsymbol{x}'}^{a} \in \mathbb{R}$ is the cost of a state transition. The sequence of state transitions that minimizes $J$ is regarded as an optimal control. $V(\boldsymbol{x})$ $(\boldsymbol{x} \in \mathcal{X}_{\mathrm{f}})$ is a value of a final state. All tasks discussed in this paper are formulated as a basic control problem in which the number of time steps $t_{\mathrm{f}}$ is minimized. The cost $r_{\boldsymbol{x}\boldsymbol{x}'}^{a}$ and the value of final states are fixed to one step and zero respectively.

Under the conditions, we can expect the existence of the optimal policy:

$$\Pi^{*}: \mathcal{X} \to \mathcal{A} \qquad (2)$$

toward (1). $\Pi^{*}$gives the best action $a^{*} = \Pi^{*}(\boldsymbol{x})$ at any state $\boldsymbol{x}$ to minimize $J$. There also exists the function $V^{*}: \mathcal{X} \to \mathbb{R}$ giving the expected value of $J$ for each state. $V^{*}$ is called the optimal value function. The value $V^{*}(\boldsymbol{x})$ is not changed whether $\boldsymbol{x}$ is the initial state or a halfway state. The values of final states $V^{*}(\boldsymbol{x})$ $(\boldsymbol{x} \in \mathcal{X}_{\mathrm{f}})$ are included in this function.

In this paper, we assume that the robot knows the optimal value function $V^{*}$. Moreover, it knows the state transition pdf $p_{\boldsymbol{x}\boldsymbol{x}'}^{a}$ and the cost $r_{\boldsymbol{x}\boldsymbol{x}'}^{a}$ of the task. In this case, the optimal policy can be derived as

$$\Pi^{*}(\boldsymbol{x}) = \operatorname*{argmin}_{a \in \mathcal{A}} \int_{\boldsymbol{x}' \in \mathcal{X}} p_{\boldsymbol{x}\boldsymbol{x}'}^{a} \{V^{*}(\boldsymbol{x}') + r_{\boldsymbol{x}\boldsymbol{x}'}^{a}\} \mathrm{d}\boldsymbol{x}'. \qquad (3)$$

### C. *State recognition*

The robot imperfectly recognizes the state through the belief $b: \mathcal{X} \to \mathbb{R}$, which is a probability density function. In this paper, the imperfectness or uncertainty of information means that $b$ is a multi-modal distribution, or that its variance is too large for decision making.

We assume that the belief $b$ is calculated by a Bayes filter [11]. In a self-localization problem, a particle filter [21] is typically used as a Bayes filter.

A Bayes filter changes the belief when the robot takes an action or when it obtains sensor measurement. After the robot takes an action, the belief is updated by

$$b(\boldsymbol{x}') = \int_{\boldsymbol{x} \in \mathcal{X}} p_{\boldsymbol{x}\boldsymbol{x}'}^{a} b(\boldsymbol{x}) d\boldsymbol{x}. \qquad (4)$$

This equation represents the flow of the belief $b$ caused by the action $a$.

When the robot obtains sensor measurement or useful information $\boldsymbol{z}$ for the state estimation, the belief $b$ is updated based on the Bayes theorem [22]:

$$b(\boldsymbol{x}|\boldsymbol{z}) = \eta P(\boldsymbol{z}|\boldsymbol{x}) b(\boldsymbol{x}). \tag{5}$$

$b(\boldsymbol{x}|\boldsymbol{z})$ is the belief after $\boldsymbol{z}$ is known. $P(\boldsymbol{z}|\boldsymbol{x})$ is a likelihood function, which translates information $\boldsymbol{z}$ to information in the state space. Likelihood functions are implemented to the robot as previous knowledge. $\eta$ is a constant that makes the summation of $b(\boldsymbol{x}|\boldsymbol{z})$ to one. After the calculation of (4) or (5), $b(\boldsymbol{x})$ is replaced by $b(\boldsymbol{x}')$ or $b(\boldsymbol{x}|\boldsymbol{z})$ respectively as the latest belief.

As mentioned in Sec. III-A, moreover, the robot can observe whether the task is finished or not. This information can be also defined as a likelihood function. When the robot observes that the task is not finished, this information can be reflected to $b(\boldsymbol{x})$ with the following likelihood function:

$$P(\boldsymbol{z} \mid \boldsymbol{x}) = \begin{cases} 0 & (\boldsymbol{x} \in \mathcal{X}_\mathrm{f}) \\ 1 & (\text{otherwise}) \end{cases}. \tag{6}$$

## IV. Generalization of The Probabilistic Flow Control Method

### A. $Q_\mathrm{MDP}$ and PFC

The value used for decision making in the $Q_\mathrm{MDP}$ value method is calculated by

$$Q_\mathrm{MDP}(a, b) = \int_{\boldsymbol{x} \in \mathcal{X} - \mathcal{X}_\mathrm{f}} b(\boldsymbol{x}) \int_{\boldsymbol{x}' \in \mathcal{X}} p^a_{\boldsymbol{x}\boldsymbol{x}'} \{V^*(\boldsymbol{x}') + r^a_{\boldsymbol{x}\boldsymbol{x}'}\} \mathrm{d}\boldsymbol{x}' \mathrm{d}\boldsymbol{x}. \tag{7}$$

This value is an expected sum of posterior values and rewards by an action $a$ when the belief is $b$. The robot chooses an action that minimizes this value.

A problem of $Q_\mathrm{MDP}$ is that the function $Q_\mathrm{MDP}(a, b)$ has local minima. The optimal value function $V^*$ calculated precisely in all parts of a state space has no local minimum. It means that every part of $V^*$ is descending to the goal and the robot always know in which direction to go. Whereas $Q_\mathrm{MDP}(a, b)$ is not optimal in the belief space. When $b$ is uncertain, the gradient of $Q_\mathrm{MDP}(a, b)$ disappears and the robot easily strands.

Differently from $Q_\mathrm{MDP}$, PFC [6] uses the following value:

$$Q_\mathrm{PFC}(a, b) = \int_{\boldsymbol{x} \in \mathcal{X} - \mathcal{X}_\mathrm{f}} w(\boldsymbol{x}) \int_{\boldsymbol{x}' \in \mathcal{X}} p^a_{\boldsymbol{x}\boldsymbol{x}'} \{V^*(\boldsymbol{x}') + r^a_{\boldsymbol{x}\boldsymbol{x}'}\} \mathrm{d}\boldsymbol{x}' \mathrm{d}\boldsymbol{x} \tag{8}$$

where $w(\boldsymbol{x}) = b(\boldsymbol{x}) / \{V^*(\boldsymbol{x}) - V_\mathrm{min}\}$. (9)

$V_\mathrm{min}$ is the smallest value of $V^*(\boldsymbol{x})$ in $\mathcal{X}_\mathrm{f}$. The value of $V^*(\boldsymbol{x})$ at every non-final state must be larger than $V_\mathrm{min}$.

It has been confirmed in [6] that the local minima problem with $Q_\mathrm{MDP}$ goes into remission by PFC. The smaller $V^*(\boldsymbol{x})$ a state $\boldsymbol{x}$ has, the larger weight (8) gives to $\boldsymbol{x}$. In the case of a simple navigation problem, for example, the nearer part to the goal in the distribution of $b$ takes a priority for decision making. This priority makes a robot take searching behavior as mentioned later.

### B. Generalized probabilistic flow control method

We extend the definition of $Q_\mathrm{PFC}$ from (8) to

$$Q_{\mathrm{PFC}m}(a, b) = \int_{\boldsymbol{x} \in \mathcal{X} - \mathcal{X}_\mathrm{f}} w_m(\boldsymbol{x}) \int_{\boldsymbol{x}' \in \mathcal{X}} p^a_{\boldsymbol{x}\boldsymbol{x}'} \{V^*(\boldsymbol{x}') + r^a_{\boldsymbol{x}\boldsymbol{x}'}\} \mathrm{d}\boldsymbol{x}' \mathrm{d}\boldsymbol{x}, \tag{10}$$

where $w_m(\boldsymbol{x}) = b(\boldsymbol{x}) / \{V^*(\boldsymbol{x}) - V_\mathrm{min}\}^m$. (11)

$Q_\mathrm{MDP}$ and $Q_\mathrm{PFC}$ can be redefined to $Q_\mathrm{PFC0}$ and $Q_\mathrm{PFC1}$ respectively with this formulation. When $m > 1$, $w_m$ in (11) gives larger priority to the states with small (good) $V^*$ values than the original function in (9). With (11), the policy is defined as

$$\Pi_{\mathrm{PFC}m}(b) = \operatorname*{argmin}_{a \in \mathcal{A}} Q_{\mathrm{PFC}m}(a, b). \tag{12}$$

The extended PFC with $Q_{\mathrm{PFC}m}$ is called PFC$_m$ in this paper. PFC$_m$ with $m > 1$ will make the robot more speculative than $Q_\mathrm{MDP}$ and the original PFC. It will prevent the robot from the local minima more effectively. However, we should also consider its ill effects.

## V. Comparison and Observation of Behaviors Generated by PF$C_m$

$Q_\mathrm{MDP}$, the original PFC and PFC$_m$ ($m > 1$) are compared with a simulation in this section.

### A. Mobile robot navigation with only one landmark

We use the first simulation in [6] for the comparison at first. Since its detail can be seen in [6], we only describe its important features.

A mobile robot is given a navigation task in an environment shown in Fig. 1, where only one landmark exists. The robot can measure its relative distance and direction. The task of the robot is to step on the goal point with its body.

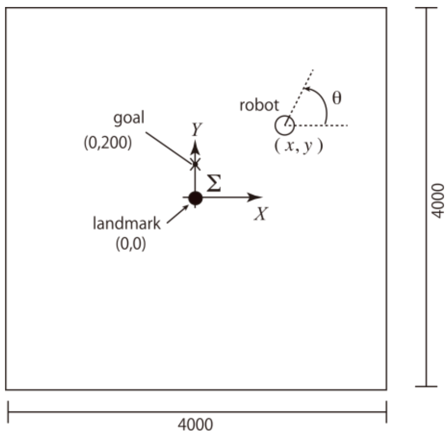

Fig. 1. Environment for the simulation with a mobile robot (modified from a figure in [6])

The robot has a particle filter [21] for self-localization. A distribution of particles shown in a trial is illustrated in Fig. 2. This distribution approximately represents a belief $b$.

The belief in Fig. 2 has the "uncertain" handled in this paper. Since the robot cannot detect its three-dimensional state $(x, y, \theta)$ through observations of the landmark with two-dimensional parameters, particles draw a circular ring. This symmetric property is slightly broken when some particles enter the goal by the update with (6). In the figure, the upper part of the ring is lacked by this update.

In [6], the robot with the original PFC can reach the goal by behaviors that make particles drop into the goal. Figure 3 shows a trajectory obtained with an experiment in [6].

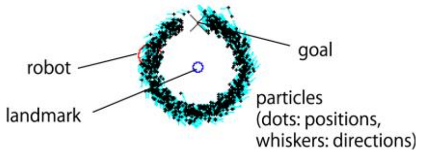

Fig. 2. An example of particle distribution

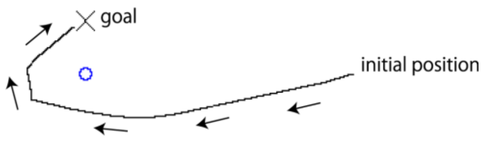

Fig. 3. A trajectory of the robot obtained by PFC method (obtained in the simulation for [6])

We examine PFC$_m$ with $m = 0, 1, 2, \dots, 9$. The robot takes 100 trials for each $m$. In each trial, the robot starts from one of 100 initial states chosen beforehand. The number of steps in each trial is recorded. When the robot cannot finish the task within 1000[step], the trial is regarded as a failure.

### B. Result of the trials

Figure 4 shows the success rates and the average numbers of the trials. First of all, PFC$_m$ with $m > 1$ achieved high success rates when compared to $Q_{\mathrm{MDP}}$ (= PFC$_0$). The advantage of PFC compared to $Q_{\mathrm{MDP}}$ has already reported in [6]. PFC$_m$ with $m > 1$ also has the advantage in this simulation.

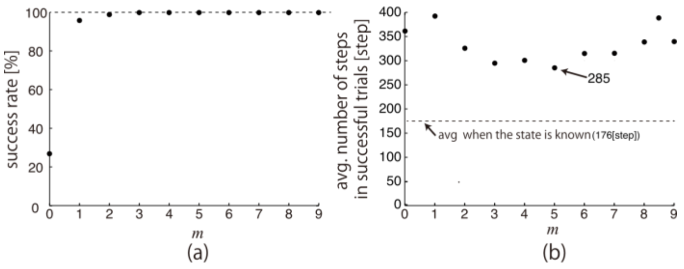

Fig. 4. Success rates and average steps on the mobile robot task

When $m > 1$, all of their success rates and average steps are better than those of the original PFC (= PFC$_1$). This tendency has also been reported on the manipulation task in [8].

While at the same time, the larger $m$ is, the more steps are required when $m \geq 5$ as shown in Fig. 4(b). In [8] also, this increase has been observed when $m \geq 2$. This tendency can be understood if we imagine an extreme case with $m = \infty$. In this case, the robot ignores all particles other than the nearest particle to the goal. This extreme strategy sometimes makes the robots detour, while it is a good strategy for avoidance of deadlock.

## VI. SYMMETRIC MAZE TASK

We examine PFC$_m$ with an actual robot, which is a micromouse type [23] robot shown in Fig. 5. The robot moves a symmetric environment in Fig. 6. This environment consists of eight square regions as illustrated in Fig. (b). Each square area is named "a block" in this paper. The task of this robot is to step into the goal block designated in Fig. (b) with the minimum number of steps from an initial pose. When the robot touches the coin shown in Fig. (a), we regard that the robot has entered in the goal block.

We set a robot coordinate system $\Sigma_{\mathrm{robot}}$ as shown in Fig. 6(b). Its origin is set at the midpoint of the axis of the wheels. The state of the system is the pose $(x, y, \theta)$ of $\Sigma_{\mathrm{robot}}$ in the environment coordinate system $\Sigma_{\mathrm{env}}$.

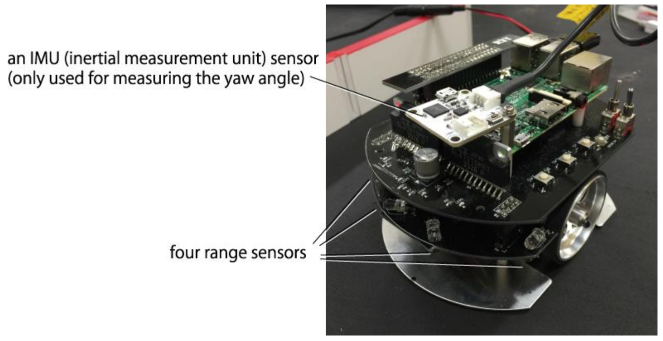
an IMU (inertial measurement unit) sensor
(only used for measuring the yaw angle)
four range sensors

Fig. 5. A micromouse type robot (Raspberry Pi Mouse [24])

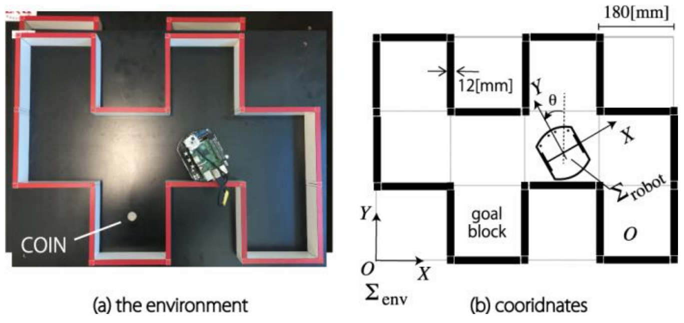

Fig. 6. Environment and coordinate systems

### A. Motion of the robot

The robot has two stepper motors directly connected to the right wheel and left one respectively. It can choose three kinds of actions: $\mathcal{A} = \{\mathrm{ccw}, \mathrm{cw}, \mathrm{fw}\}$, where

- ccw: rotation of 5[deg],
- cw: rotation of −5[deg], and
- fw: forward movement of 40[mm].

Execution of one of these actions is counted as one time step.

The robot can know its direction $\theta$ within one degree accuracy through the IMU (inertial measurement unit) sensor shown in Fig. 5. Whereas the robot cannot use any sensor to directly know the change of the position $(x, y)$.

## *B. Uncertainty and perceptual aliasing of self-localization*

To estimate the position $(x, y)$, the robot uses four range sensors shown in Fig. 5 and dead reckoning. Their information is converted to the position by a particle filter. We explain the implementation in Appendix A. The number of particles is 1000.

Figure 7 illustrates some distributions of particles shown in trials. Particles never converge in most of the time due to the symmetric environment. They make some clusters as shown in Fig. 7. In Figs. (b) and (c), we can observe the perceptual aliasing. Moreover, the position of the robot in a cluster is also inaccurate since the robot frequently collides with the wall and skids on the smooth floor.

To measure how long the robot cannot specify its pose in this environment, we count the number of blocks in which one or more particles exist at each moment from the log files of a set of the trials in Sec. VI-D. In the case of distributions in Figs. 7(a)-(c), for example, the numbers are eight, four and two respectively. The result is shown in Fig. 8. In more than 50[%] of the time span, the number is seven or eight, whereas the moments in which the number is two or less are only 13[%]. When the number is more than three (80[%] of the time span), the robot must decide its action under perceptual aliasing or shortage of its pose information.

## *C. Obtaining an optimal value function of the task*

We use a value iteration algorithm for obtaining the optimal value function $V^*$. The detail of the implementation is described in Appendix B.

To apply value iteration, the $xy\theta$-space is divided into a three-dimensional grid as shown in Fig. 9. The region $0 \le x <$ 720[mm], $0 \le y <$ 540[mm], $-2.5 \le \theta <$ 357.5[deg] of the environment is divided into a cell with 20[mm]×20[mm]×5[deg] size. Though the whole area in the goal block is the final state of this task, we set a non-final state area at the entrance of the goal block as shown in Fig. 9. This area helps the robot to enter the goal block surely.

Figure 10 shows the optimal value function obtained by the value iteration. It is calculated as a three-dimensional look-up table. This function returns the excepted number of steps from every cell to a final state.

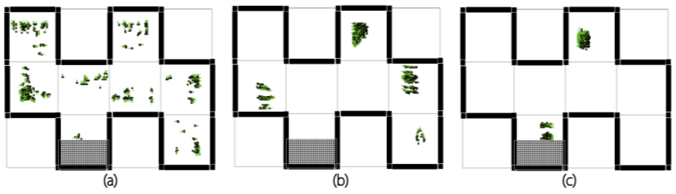

Fig. 7. Examples of particle distributions

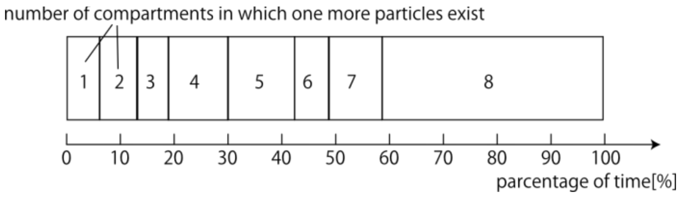

Fig. 8. The severity of self-localization uncertainty in trials with $m$ =4

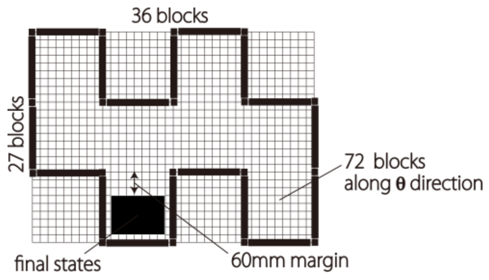

Fig. 9. Discretization

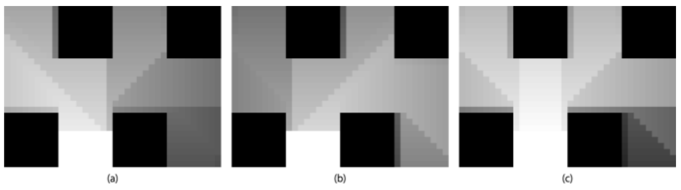

Fig. 10. Obtained value function represented as gray scale images (black: 100[step], white: 0[step]). (a) values of cells with $\theta \in [-2.5, 2.5)$ [deg]. (b) values with $\theta \in$[132.5, 137.5) [deg]. (c) values with $\theta \in$ [267.5, 272.5) [deg].

## *D. Trials and the results*

We have 14 trials for each value of $m = 0, 1, 2, \dots, 9$. In a trial, the robot starts from the center position of a non-goal block. $\theta$ of the initial pose is 90[deg] or 270[deg]. A trial is cut off at 500[step] in consideration of the battery life. Additionally, we should note that the IMU sensor occasionally did not return a value and the robot stopped due to the halt. In that case, we reconnected the USB connector of the IMU sensor and reran the robot in the middle of a trial.

When $m \ge 3$, we could observe that the robot moved from block to block as it searched the goal even though particles were not converged most of the time. The robot frequently got stuck with walls due to the uncertainty of self-localization, and the collisions gave off an awkward impression on the robot. However, the robot could go to the goal block with some reasonable behaviors.

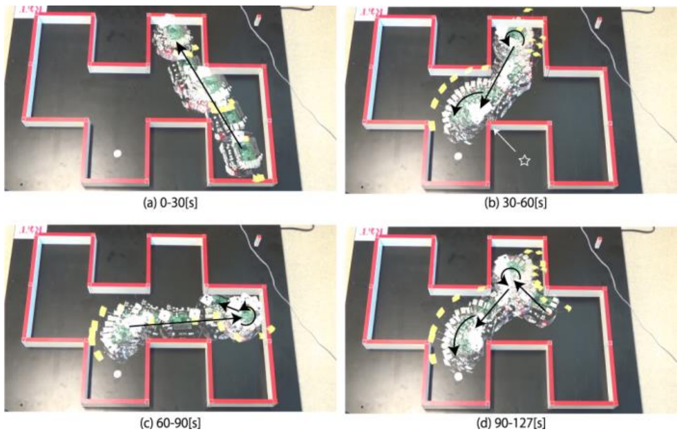

Fig. 11. Behavior of the robot in a trial ($m$ =4, 292[step])

Figure 11 illustrates a typical successful trial. In (a), the robot visited the counter block of the goal. Subsequently, the robot went to the goal block in (b). However, the robot got stuck with the corner marked with a star and got away from the goal shown in (c). After that, the robot went back to the goal block as shown in (d). Though the robot was disturbed by the corner also in this time, it could slide into the goal block.

In Fig. 12, we show the numbers of trials finished within 500[step] for each value of *m*. When $m \leq 2$, the robot frequently fell into repeating choices of cw and ccw. The numbers of these repeating choices are shown in Fig. 13. As with the case of the simulations, an increment of *m* reduced the frequency of deadlocks.

### *E. Discussion of a problem for future work*

We found that the robot frequently collided with the corners of the environment as shown in Fig. 11(b) and (d). Figure 14 illustrates the reason. In the case of this figure, particles near the goal try to the robot turn left, whereas the other particles correctly suggest the robot to go forward. When *m* is large, the particles near the goal are given high weights. Therefore, the robot tends to get around a corner before a sufficient forward movement.

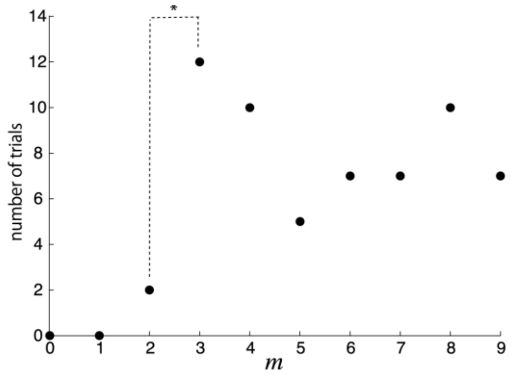

Fig. 12. Numbers of successful trials that finished within 500[step]

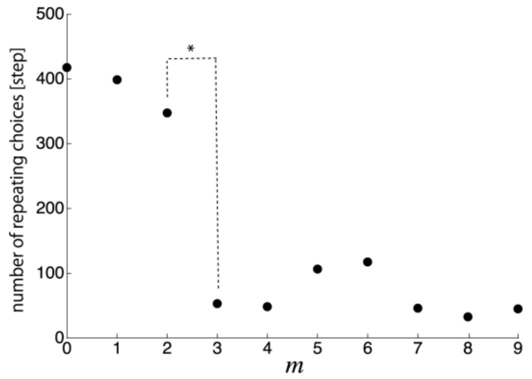

Fig. 13. Average numbers of steps in which cw/ccw is chosen after ccw/cw per trial

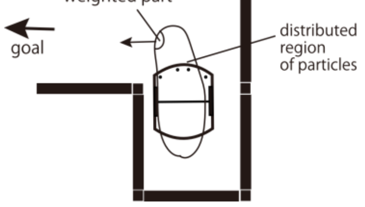

Fig. 14. A typical case where the robot collides with a wall

PFC$_m$ does not consider the existence of obstacles as it is now. To solve this problem, we should add PFC$_m$ to an algorithm to forecast the interference with obstacles. This will be the future work that should be tackled.

## VII. Conclusion

We have proposed an extension of PFC in this paper. This extension unifies the $Q_{\mathrm{MDP}}$ value method, the original PFC method and newly derived methods as PFC$_m$. PFC$_0$ and PFC$_1$ are the $Q_{\mathrm{MDP}}$ value method and the original PFC method respectively. PFC$_m$s $(m \geq 2)$ are the newly defined.

We have examined PFC$_m$ in a simulation and an experiment. Another simulation has also been held in [8]. In all these tasks, the robot achieves the best performance when $m > 1$. These results suggest that the extension is effective. Meanwhile, a problem explained with Fig. 14 is found. We try solving this problem in future. Moreover, we found that a larger $m$ is not always better. We will try changing $m$ dynamically to an appropriate value.

## Appendix

### *A. Implementation of a particle filter for the task in the symmetric environment*

The particle filter used in Sec. VI is explained here. The particle filter is implemented based on [25] with sensor resetting [26] and expansion resetting [27]. The number of particles is 1000. At the beginning of each trial, the particles are distributed uniformly in the maze. Though the robot can receive absolute $\theta$ information from the IMU sensor, we define the particle filter in the $xy\theta$-space since we handle the initial direction of the robot as unknown.

*1) Motion update:* After the robot takes an action, the particles change their poses based on the dead-reckoning. The displacement is ±5[deg] at action cw or ccw, and 40[mm] at action fw. Gaussian noises are added to the displacement on $\theta$-axis and the $xy$-plain independently. The standard deviations of the noises 0.1[%] toward the change on $\theta$ and 10[%] toward the change of position on the $xy$-plain. After that, $\theta$ is corrected by the values from the IMU. When some particles collide with the wall by the motion update, they do not move at the procedure.

*2) Sensor update:* The values of the range sensors are used for judging whether a wall exists or not in front of the robot. If both forward-looking range sensors return values over a threshold, it is judged that no wall exists. If all the four sensors return values under a threshold, it is judged that a wall exists. When one of these conditions is fulfilled, the weights of the particles that contradict the judgment are multiplied by a near zero number ($10^{-10}$ in the implementation).

*3) Reset:* When the sum of weights before normalization is smaller than 0.2 by the sensor update, the particle filter resets the particles. At first, the expansion resetting method is used [27]. When another reset is required by the subsequent sensor update, the sensor resetting [26] is executed. In this reset, all the particles are replaced randomly at the poses that are consistent with the wall judgement.

### *B. Value iteration for the symmetric maze task*

As shown in Fig. 9, the $xy\theta$-space is divided into 36×27×72=69,984 cells. We chose the width of a cell

(20[mm]×20[mm]×5[deg]) in relation to the amount of displacement by the actions. Each cell is regarded as a discrete state $s_i$ $(i = 1,2,\dots,69{,}984)$. We use $S$ to represent the set of these states hereafter.

A state transition is treated as deterministic one for simplification. When the state is changed from $s \in S$ to $s' \in S$, we assume that the robot always starts at the center point of $s$ in a state transition. Then the state $s'$ that contains the posterior pose by an action $a$ is regarded as the only posterior state. When $s'$ contains a wall or it is outside of the maze, we assume that the robot stays at $s$.

Since the purpose of this task is to make the robot go to the goal block as small number of steps as possible, we give 1[step] penalty for each state transition. The optimal value function $V: S \to \mathbb{R}$ obtained with this penalty gives the number of required steps from any state $s$ to the goal.

It takes 6.5[s] with two chips of Intel Xeon CPU E5-2670 v2. 20 threads are used for this calculation. Some parts of the result have been shown in Fig. 10.

## References

[1] N. Roy, W. Burgard, D. Fox, and S. Thrun, "Coastal Navigation - Mobile Robot Navigation with Uncertainty in Dynamic Environments," in *Proc. of IEEE International Conference on Robotics and Automation*, pp. 35–40, 1999.

[2] L. Kaelbling, M. Littman, and A. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[3] J. Pineau and G. J. Gordon, "POMDP planning for robust robot control," in *Robotics Research*, Springer Berlin Heidelberg, pp. 69–82, 2007.

[4] R. Ueda, T. Arai, K. Sakamoto, Y. Jitsukawa, K. Umeda, H. Osumi, T. Kikuchi, and M. Komura, "Real-Time Decision Making with State-Value Function under Uncertainty of State Estimation," in *Proc. of ICRA*, pp. 3475–3480, 2005.

[5] S.-Y. Chung and H.-P. Huang, "Robot Motion Planning in Dynamic Uncertain Environments," *Advanced Robotics*, vol. 25, no. 6-7, pp. 849–870, 2011.

[6] R. Ueda, "Generation of Compensation Behavior of Autonomous Robot for Uncertainty of Information with Probabilistic Flow Control," *Advanced Robotics*, vol. 29, no. 11, pp. 721–734, 2015.

[7] M. L. Littman *et al.*, "Learning Policies for Partially Observable Environments: Scaling Up," in *Proc. of International Conference on Machine Learning*, pp. 362–370, 1995.

[8] R. Ueda, "Searching behavior of a simple manipulator only with sense of touch generated by probabilistic flow control," in *Proc. of IEEE ROBIO*, pp. 594–599, 2018.

[9] B. Bonet and H. Geffner, "Solving pomdps: Rtdp-bel vs. point-based algorithms," in *Proc. of IJCAI*, pp. 1641–1646, 2009.

[10] B. Bonet, "Deterministic POMDPs Revisited," in *Proc. of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pp. 59–66, 2009.

[11] S. Thrun, W. Burgard, and D. Fox, *Probabilistic ROBOTICS*. MIT Press, 2005.

[12] T. Fukase, M. Yokoi, Y. Kobayashi, H. Yuasa, and T. Arai, "Quadruped Robot Navigation Considering the Observation Cost," in *RoboCup 2001: Robot Soccer World Cup V*, A. Birk, S. Coradeschi, and S.Tadokoro, Eds., pp. 350–355, 2002.

[13] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.

[14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: The MIT Press, 1998.

[15] S. C. Ong, S. W. Png, D. Hsu, and W. S. Lee, "Planning under uncertainty for robotic tasks with mixed observability," in *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1053–1068, 2010.

[16] D. Silver and J. Veness, "Monte-Carlo Planning in Large POMDPs," in *NIPS*, vol. 23, pp. 2164–2172, 2010.

[17] S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa, "Online Planning Algorithms for POMDPs," in *Journal of Artificial Intelligence Research*, vol. 2008, no. 32, pp. 663–704, 2008.

[18] Y. Jitsukawa *et al.*, "Fast Decision Making of Autonomous Robot under Dynamic Environment by Sampling Real-Time Q-MDP Value Method," in *Proc. of IROS*, pp. 1644–1650, 2007.

[19] O. Aşık and L. Akın, "Solving multi-agent decision problems modeled as dec-pomdp: A robot soccer case study," in *Randy Goebel and Yuzuru Tanaka and Wolfgang Wahlster (Eds.): RoboCup 2012: Robot Soccer World Cup XVI*, pp. 130–140, 2013.

[20] R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun, "Approximate Solutions for Partially Observable Stochastic Games with Common Payoffs," in *Proc. of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, vol. 1, pp. 136– 143, 2011.

[21] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots," in *Proc. of AAAI*, pp. 343–349, 1999.

[22] A. Doucet, N. de Freitas, and N. Gordon, Eds., *Sequential Monte Carlo Methods in Practice*. New York, NY: Springer-Verlag, 2001.

[23] R. Allan, "Three amazing micromice: hitherto undisclosed details," *IEEE Spectrum*, vol. 15, no. 11, pp. 62–65, 1978.

[24] Y. Nakagawa, M. Aoki, S. Sakura, N. Nakagawa, R. Ueda, and A. Eguchi, "Raspberry Pi Mouse: A Micromouse with Full Linux Environment," in *Proc. of 2015 JSME/RMD International Conference on Advanced Mechatronics (ICAM)*, pp. 14–15, 2015.

[25] D. Fox, S. Thrun, W. Burgard, and F. Dellaert, "Particle Filters for Mobile Robot Localization," A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, pp. 470–498, 2000.

[26] S. Lenser and M. Veloso, "Sensor resetting localization for poorly modelled robots," in *Proc. of IEEE ICRA*, pp. 1225–1232, 2000.

[27] R. Ueda, T. Arai, K. Sakamoto, T. Kikuchi, and S. Kamiya, "Expansion Resetting for Recovery from Fatal Error in Monte Carlo Localization – Comparison with Sensor Resetting Methods," in *Proc. of IROS*, pp. 2481–2486, 2004.

**Ryuichi Ueda** received his BE, ME, and Ph.D. in Engineering from University of Tokyo, in 2001, 2003, and 2007 respectively. He worked in University of Tokyo as an assistant professor from 2004 to 2009, in Universal Shell Programming (USP) Laboratory Ltd. as a researcher from 2009 to 2013, and Advanced Institute of Industrial Technology as an assistant professor from 2013 to 2015. He is an associate professor at Chiba Institute of Technology from 2015. His research interests are in probabilistic estimation and decision making in robotics. He received JSME (The Japan Society of Mechanical Engineers) Education Award for his translation and publication of textbooks on probabilistic robotics in 2020. He has been also known as a master of shell one-liner on Unix/Linux and has published several articles and books.