Decoupling MDPs Step by Step from a POMDP

Chyon Hae Kim[1], Hiroshi Tsujino[1], Hiroyuki Nakahara[2]

[1] Honda Research Institutes Japan Co.,Ltd.
{tenkai,tsujino}@jp.honda-ri.com
[2] Integrated Theoretical Neuroscience RIKEN Brain Science Institute
hn@brain.riken.jp

Abstract. This paper addresses the problem, how an artificial agent decouples a partially observable Markov decision process (POMDP) to several Markov decision processes (MDPs) according to the dimensional hierarchy of the MDPs. We propose multi-layered reinforcement learning (MLRL) that selectively uses each layer to learn each MDP to reduce the learning cost. The MLRL separately learned two MDPs step by step in a simulated capture task. Also, the MLRL learned faster than SARSAs in the capture task and a simulated guiding task.

1 Introduction

This paper addresses the problem, how an agent learns for Markov decision processes (MDPs) that are inside a partially observable Markov decision process (POMDP).

Traditionally, many kinds of reinforcement learning (RL) systems for MDP and POMDP have been researched. MDP is a special case of POMDP, and there are many techniques to solve MDPs (e.g. SARSA and Q-learning are very famous formulations) [C89,C92,RA00]. On the other hand, there is no technique to solve general POMDPs, since POMDPs are often computationally intractable [LM98,Ke00].

To adapt to general POMDPs, an agent requires these capabilities.

1. Utilization for temporal sequence
2. Decision making that is based on statistical state models.

However, utilization for temporal sequence increases the complexity of learning, and statistical state models require a lot of observation data. In many practical scene, these problems result in large computational cost.

To relax the cost, we discuss how to find learnable MDPs inside a POMDP. If an RL system is able to decouple the MDPs from a POMDP, the system does not need to learn the MDPs using POMDP frameworks as traditional RL systems do. After the MDPs are learned, the remained POMDP will not require large cost to be learned. We propose a multi-layered RL formulation to decouple the MDPs step by step.

The structure of this paper is as follows: In Section 2, we formulate the proposed RL. In Section 3, we describe the experimental systems for a capture task and a guiding task. In Section 4, we show the results of the experiments. In Section 5, we mention our considerations. In Section 6, we conclude this paper.

2 Algorithm

2.1 Definition of a POMDP

POMDP frameworks are defined by a tuple (S, A, O, T, Ω, R), where S is a set of states, A is a set of actions, O is a set of observations, T is a set of conditional transition probabilities, Ω is a set of conditional observation probabilities, $R : S \times A \to R$ is the reward function. An agent that reaches the state $s' \in S$ from the state $s \in S$ using an action $a \in A$ observes $o \in O$ with probability $\Omega(o|s', a)$. In general, observation o is composed of partial observations y_is $o := \{y_1, y_2, \cdots, y_n\}$.

2.2 The least combination of elements to describe transition

In POMDP framework, we assume that state transitions follow MDP. The transition probability T is defined as follows:

$$T := T(s'|s, a) \tag{1}$$

Usually, the state transition is not observed directly, since observation o has smaller number of elements than $s = \{y_1, y_2, \cdots, y_n, \hat{y}_1, \hat{y}_2, \cdots, \hat{y}_n\}$. However, in some cases, s is redundant to describe transition probability T. In these cases, the state transition is observable. For example, in the case that $T(s'|s, a) = T(s'|o, a)$, elements $\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_n$ are redundant, and o has sufficient kinds of elements to observe the state transition. Generally, each transition T has the least combination C of elements to be described. If the combination C is inside the elements of observation o while a transition, the transition is inside an MDP (sub-task) that is inside a POMDP (whole task). So, there is possibility that RL system adapts to the MDPs while learning a POMDP problem. This kind of adaptation will help the system to solve the POMDP, since each MDP is tractable using traditional RL framework. However, we need to consider the hierarchy of MDPs to divide MDPs from a POMDP.

2.3 Approach

To divide a POMDP into several MDPs, which are computationally tractable, and the a remained POMDP, we need to consider the hierarchy of observable information of an agent. We propose a step by step decoupling method that uses an RL system, which equips with multi-layers, based on the consideration. Each layer of the multi-layers is related to one of the hierarchy of MDPs.

In many cases, a POMDP includes many MDPs Ms as sub-tasks of the POMDP. These MDPs are categorized by the combination C of partial observations y_is. We define an MDP that requires only C to be learned as $M_C = M_{\{\cdots\}}$. Among them, the most simple MDPs are $M_{\{y_i\}}$s that include only one element y_i. An RL system RL_1 that learns the POMDP while assuming the POMDP as $M_{\{y_i\}}$ may converge to local optima. However, another RL system RL_2 that

learns the same POMDP while assuming the POMDP as $M_{\{y_i,y_j\}}$ may converge to a better solution than RL_1 (or the best solution of the POMDP. See [JS98] for the case when SARSA converges to an optimal policy of POMDP problems). Therefore, to find MDPs inside a POMDP, an RL system should select the best combination C of partial observations y_is or the best M for each sub-task inside a POMDP. We formulated an RL system for the problem, and examined the system to select Cs.

In our approach, an artificial agent decouples the POMDP to a simple MDP (e.g. $M_{\{y_i\}}$) and the remained POMDP ($\bar{M}_{\{y_i\}}$). To realize the first decoupling, we use the first layer that corresponds to $M_{\{y_i\}}$. This layer is able to learn only for $M_{\{y_i\}}$. To proceed the decoupling, the agent decouples $\bar{M}_{\{y_i\}}$ into an MDP of second level $M_{\{y_i,y_j\}}$ and the remained POMDP ($\bar{M}_{\{y_i,y_j\}}$) again. This layer is also able to learn only for $M_{\{y_i,y_j\}}$. This way, the proposed RL system decouples a POMDP into multi-layered MDPs.

2.4 Formulation

To explain the formulation of the proposed method, we show the formulation of a two layered RL at first, and extend that for a multi-layered RL.

Two Layered RL The two layered RL is composed of two RLs. The first layer has an RL, RL_1, that consists of SARSA or Q-learning. While an agent observes os according to its actions as, and receives rewards r. RL_1 learns the relationship between y_is, as, rs using its partial observations y_is. This means that RL_1 learns Q values for each state action pair $Q_1(y_i, a)$. If a POMDP has a sub-problem, MDP $M_{\{y_i\}}$, inside that, this RL system that has observation y_i is a good solution for the sub-problem.

As in the first layer, the second layer has an RL system, RL_2, that also consists of SARSA or Q-learning. RL_2 learns Q values for each state action pair $Q_2(y_i, y_j, a)$.

If an agent utilizes these two RL systems appropriately while learning a POMDP that includes $M_{\{y_i\}}$ and $M_{\{y_i,y_j\}}$ as sub-tasks, the agent will reduce the learning cost, since RL_1 and RL_2 are good solutions for sub-problems $M_{\{y_i\}}$, $M_{\{y_i,y_j\}}$.

We established the way to combine these two RL systems into an RL system. To utilize these systems, RL_1 and RL_2, we need these considerations.

1. How the whole RL system calculates the Q value from Q_1 and Q_2.
2. How the whole RL system learns Q values.

For the first problem, we use the following linear coupling formulation

$$Q = \sum_{k=1}^{2} w_k Q_k + w_{rem} Q_{rem} \tag{2}$$

where w_k is a weight for Q_k, w_{rest} is a weight for Q_{rest}, Q_{rest} is a Q value for an RL system that learns for a POMDP $\bar{M}_{\{y_i,y_j\}}$ that is the rest of $M_{\{y_i\}}$ and $M_{\{y_i,y_j\}}$. When we do not use an RL system for the remained POMDP, $w_{rem}Q_{rem}$ is 0. In general, an agent inside a POMDP has to consider sub-tasks of hierarchical MDPs. When an agent takes a state $S(y_i, y_j)$ in a MDP $M_{\{y_i,y_j\}}$, the agent takes a state $S(y_i)$ in another MDP $M_{\{y_i\}}$ simultaneously. So, transitions in these MDPs proceed at the same time. Therefore, the agent needs to consider the weight w of these tasks inside $M_{\{y_i,y_j\}}$ and $M_{\{y_i\}}$ to sum up the benefit of the sub-tasks.

For the second problem, if we assume the use of finite states for each layer, we are able to derive the learning formulations of the whole system as follows. When RL_1 observes y_i, RL_2 observes (y_i, y_j). We define the Q value of RL_1 as $Q_1 := Q_1(y_i)$ and define the Q value of RL_2 as $Q_2 := Q_2(y_i, y_j)$. When the RL system is in a sub-task $M_{\{y_i,y_j\}}$, we formulate the error as follows:

$$\begin{aligned} E = \ & \frac{1}{2}\sum_{y_i,y_j}\sum_{y'_i,y'_j} p^{\pi}_{y_i,y_j} P^{\pi}_{y_i,y_j,y'_i,y'_j}(r_{y_i,y_j,y'_i,y'_j} \\ & +\gamma Q'(y'_i, y'_j, \pi) - w_1 Q_1(y_i) - w_2 Q_2(y_i, y_j))^2 \end{aligned} \tag{3}$$

where $p^{\pi}_{y_i,y_j}$ is the probability where RL_2 observes (y_i, y_j), $P^{\pi}_{y_i,y_j,y'_i,y'_j}$ is the transition probability when an agent transits from the observation (y_i, y_j) to another observation (y'_i, y'_j) using an action selection policy π, r_{y_i,y_j,y'_i,y'_j} is the given reward in the transition, and Q' is the Q value of the selected action, which is based on a policy π and selected based on the next observation (y'_i, y'_j) when we apply SARSA type update.

We deduce update functions of each (y_i, y_j) from the error E using the steepest descent method, based on the two assumptions that the terms $r_{y_i,y_j,y'_i,y'_j} + \gamma Q'(y'_i, y'_j, a')$ to be the output targets of the learning system that are independent of y_i and y_j, and the symbols $p^{\pi}_{y_i,y_j}$ and $P^{\pi}_{y_i,y_j,y'_i,y'_j}$ are independent of Q_ns. Steepest descent method derives a formulation that is consistent to traditional RL theory as we mention later.

$$\begin{aligned} & \Delta Q_1(y_m) = -\alpha\frac{\partial E}{\partial Q_1(y_m)} \approx \alpha w_1 \sum_{y_j}\sum_{y'_i,y'_j} p^{\pi}_{y_m,y_j} P^{\pi}_{y_m,y_j,y'_i,y'_j} \\ & (r_{y_m,y_j,y'_i,y'_j}\gamma Q'(y'_i, y'_j, \pi) - w_1 Q_1(y_m) - w_2 Q_2(y_m, y_j)) \end{aligned} \tag{4}$$

$$\begin{aligned} & \Delta Q_2(y_m, y_n) = -\alpha\frac{\partial E}{\partial Q_2(y_m, y_n)} \approx \alpha w_2 \sum_{y'_i,y'_j} p^{\pi}_{y_m,y_n} P^{\pi}_{y_m,y_n,y'_i,y'_j} \\ & (r_{y_m,y_n,y'_i,y'_j} + \gamma Q'(y'_i, y'_j, \pi) - w_1 Q_1(y_m) - w_2 Q_2(y_m, y_n)) \end{aligned} \tag{5}$$

We show the obtained online update functions which are derived from these update functions.

$$\Delta Q_1 = \alpha_1(r + \gamma Q' - w_1 Q_1 - w_2 Q_2) \tag{6}$$
$$\Delta Q_2 = \alpha_2(r + \gamma Q' - w_1 Q_1 - w_2 Q_2) \tag{7}$$

When we change the formulations as follows, the formulations show that RL_1 and RL_2 learn separately for $TD\_Error - w_n Q_n$ as SARSAs. This means that each RL, RL_1 or RL_2, learns the rest of TD error that was learned by another RL.

$$\Delta Q_1 = \alpha_1((r + \gamma Q' - w_1 Q_1) - w_2 Q_2) \tag{8}$$
$$\Delta Q_2 = \alpha_2((r + \gamma Q' - w_2 Q_2) - w_1 Q_1) \tag{9}$$

In a special case when $w_1 = 1$ and $w_2 = 0$, This formulation is the

$$Q = Q_1 \tag{10}$$
$$\Delta Q_1 = \alpha_1(r + \gamma Q' - Q_1) \tag{11}$$

This result is consistent to traditional theory for SARSAs that consider single MDP. When the weight for the $M_{\{y_i,y_j\}}$, w_2, is 0, this multi-layered system negledts $M_{\{y_i,y_j\}}$. In this case, this system does not consider the POMDP as mixture of $M_{\{y_i\}}$ and $M_{\{y_i,y_j\}}$, but $M_{\{y_i\}}$. So, above mentioned formulation, which is consistent to SARSA, is reasonable to calculate reward estimation.

Multi-Layered RL We show the formulations of a multi-layered RL that was deduced the same as the two layered RL.

$$Q = \sum_{k=0}^{n} w_k Q_k \tag{12}$$
$$\Delta Q_i = \alpha_i((r + \gamma Q' - w_i Q_i) - \Sigma_{k \neq i} w_k Q_k) \tag{13}$$

3 Experimental Systems

We conducted two experiments, a capture experiment and a guiding experiment, to validate the proposed RL system.

3.1 Capture Experiment

We established a PC simulation. In this simulation, a learner (abstract robot) having a radius R captured the center mass of an agent in a half circle (Fig.1).

Robot The robot was equipped with the two layered proposed RL system. The robot observes the vertical relative position of the agent x, the horizontal relative position of the agent y, and the horizontal absolute velocity of the agent $\dot{y}$. We set (x, y) for the first layer's observation. The observation is related to an MDP $M_{\{x,y\}}$. We set $(x, y, \dot{y})$ for the second layer's observation. The observation is related to another MDP $M_{\{x,y,\dot{y}\}}$. We divided the input space into 100×4 (horizontal direction $\times$ vertical direction). For the vertical direction, the robot approaches an agent with a constant velocity v. For the horizontal direction, the robot selects its action among three actions: moving the half circle to the left by the length of Δ, moving the half circle to the right by the length of Δ, and remaining current position. The robot gets a reward value, 1, when it captures the agent. If the robot fails that, the robot gets a punishment value, -1.

Settings for the Agent We set two rules for the movement of the agent to make a POMDP environment for the robot. The first rule is that the agent moves left and right randomly using a normal random number $\phi(u, \sigma^2)$. If the robot learns ϕ, the robot improves capture performance. The second rule is that the agent changes the sign of u periodically. We noticed that the robot was not able to observe the sign of u, which decides the state of the agent. Therefore, the sign of u makes a POMDP environment for the robot.

We are able to control the difficulty to guess the rule of the agent by changing the parameter σ^2. This capturing task is easy when σ^2 is small, but is difficult when σ^2 is large. We show the flow of agent's motion as follows:

1. Decide the parameters of normal random numbers u and σ.
2. Add normal random number $\phi(u, \sigma^2)$ to the horizontal position of the agent y.
3. Invert the sign of u.
4. Continue Steps 2 and 3.

Parameters We set the initial position of the agent to be just above the position of the center of the robot. The robot continued to learn for one set (= 20,000 trials) using the same parameters $\Delta = 0.2$, $u = 0.2$, $\sigma^2 = 0.15$, and $R = 0.1$. At the start of the learning process, we initialized all Qs of the first layer and the second layer to zero. In order to obtain the initial bias of the Q value, we added a bias directly to the total Q ($Q = Q_1 + Q_2 + 0.007$). This bias makes optimistic selections of the actions [RA00]. We set the learning rate of each layer to 0.08.

Experiment We performed 100 experiments for each of the systems, SARSA1, SARSA2, and the proposed RL system. The capture rate values, which was obtained from the 100 experiments, were again averaged as a 2,000 trial moving average.

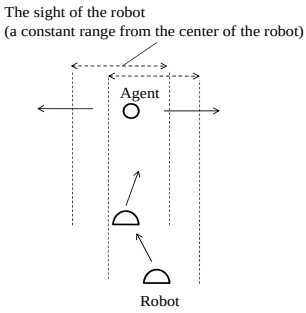

Fig. 1. Simulated capture experiment

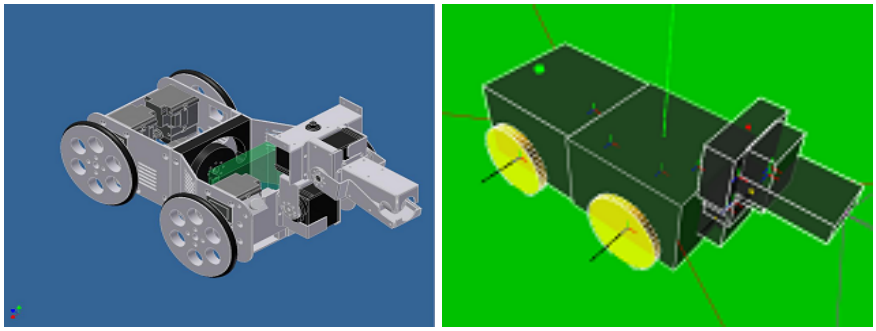

Fig. 2. Mechanical model (left) and computational model (right)

3.2 Guiding Task

We established another numerical experiment to examine the applicability of the proposed RL system for a more complicated POMDP environment. In this experiment, the learner (guiding robot) learns how to guide another robot (guided robot) to a goal. In previous studies, several researchers attempted such guiding tasks using traditional systems, such as a control system using a potential field [RN98,RN00], an evolutionary computation system [AJ96], and a classifier system [OP00]. The control system used by Vaughan gathered a flock of animals at a point using a feedback control [RN98,RN00]. This task is very useful to evaluate the proposed system, since experimenter is able to evaluate the effectiveness of RL systems from the speed to achieve the task, and the guided robot makes POMDP environment.

We modeled the hardware of the robots (Fig. 2 right) and the experimental environment (Fig. 3) on the Webots simulator [URL], which is based on a physical simulation engine called open dynamics engine (ODE). The specifications of the robots are shown in Table 1. We used the same model for both robots, the guiding robot and the guided robot.

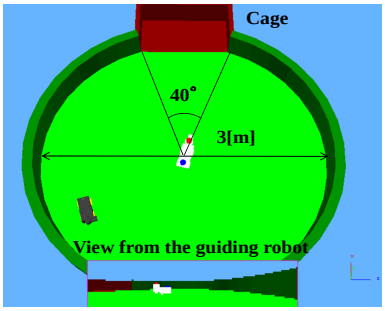

Fig. 3. Experimental environment

The guiding robot (black one) guides the guided robot (white one) into a cage. While the guidance, the guiding robot observes the guided robot using its head mount camera. The guided robot, which was equipped with two LEDs, shows the direction of its body to the guiding robot. The image from the camera is shown at the bottom of this figure.

Table 1. Specifications of the guiding and guided robots

Weight	Head		184.7 [g]
	Body (front)		370 [g]
	Body (back)		300 [g]
Size	Body	Width	120 [mm]
		Length	250 [mm]
	Wheels	Width	10 [mm]
		Radius	40 [mm]
DOF	Track Wheels		(D.O.F = 2)
	Waist		Roll (1)
	Neck		Pitch and Yaw (2)
	Jaw		Raises and lowers snout of robot (1)
Devices	Camera	Field of View	2 radians
		Resolution	128×32 pix.
	IR Sensor	Quantity	4
		Placement	30 degrees from side parallel
	Gyroscope		(not in the model)

Guiding Robot The software system of the guiding robot has three components, a pre-processing system, a learning system, and a behavior generation system.

[Pre-processing system]: This system process the information, which is obtained from the head-mounted camera image. The result is sent to the learning system (Table 2). From the image obtained by the head-mounted camera, the guiding robot extracts the weight centers of the guided robot, the cage, and the LEDs on the guided robot using the thresholds of their colors. The guiding robot then calculates the direction of the guided robot from the position of the LEDs. In addition, the guiding robot extracts the vertical edges of the cage using the Hough transform. The system normalized the horizontal weight centers of the guided robot and the cage, the sine and cosine of the direction vector of the guided robot, and the horizontal positions of the edges of the cage, to the range

Table 2. Observation of the learning system

Target	Information (dimension)	Range
Self (x)	Neck yaw (1)	$[-1, 1]$
Other agent (y)	Horizontal weight center (1)	$[0, 1]$ (detected)
	Rotation ($cos\theta$, $sin\theta$) (2)	-1 (not detected)
Cage (z)	Horizontal weight center (1)	$[0, 1]$ (detected)
	Horizontal corner position (2)	-1 (not detected)

of [0,1]. If the objects are out of view and the guiding robot fails to detect the objects, -1 is assigned to the value of the information. The angle of the guiding robot's neck , which is obtained from the encoder, is also normalized to the range of [0,1].

[Learning system] We applied the proposed two layered RL system with a predictor. The predictor predicts the velocity of the guided robot. We constructed the predictor, which has a mesh type function approximator, using an online learning process of the guiding robot. We set each cell of the mesh to output each prediction of $\tilde{y}$ for the corresponding observation of the robot. The predictor calculates an average value from the training data for $\tilde{y}$, and fixes the output of each cell to the value. We let the guiding robot move randomly using its action primitives (see the following subsection) around the guided robot in the experimental environment. Simultaneously, the predictor of the guiding robot was updated. We continued this update for 10 hours of the simulation time.

We set several rewards according to the state of the robots. The guiding robot rewarded its reinforcement learning system automatically with a reward of 0.1 when the guided robot and the cage overlapped on the image, which is obtained by the head-mounted camera of the guiding robot. From this state, if the guiding robot moved toward the guided robot, the learning system received a reward of 1. When the guiding robot successfully completed the guidance and the guiding robot confirmed the success by the head-mounted camera, the learning system received a reward of 10.

Action primitives We prepared eight action primitives (Table 3). The guiding robot executed one of the primitives that was selected by its learning system. Each action costs each time interval Δt. So, we used γ' instead of constant γ.

$$\gamma' = \gamma^{\Delta t} \tag{14}$$

Guided robot The guided robot moves according to its input from the infrared sensors (IR-sensors) and the force field that is set in the environment. The guided robot avoids obstacles and the guiding robot using its IR-sensors (Table 4). This avoidance has higher priority than movement according to the force field. So, while avoidance, the guided robot neglects the force field.

Table 3. Action primitives

Index	Time interval Δt [s]	Motion
A_0	1	Stay
A_1	1	Move toward the position of the guided robot
A_2	2	Turn clockwise around the guided robot
A_3	2	Turn counterclockwise around the guided robot
A_4	1	Move away from the position of the guided robot
A_5	1	Search for the guided robot
A_6	5	Move away from the cage
A_7	1	Search for the cage

Table 4. Collision Avoidance

Which sensors detect the objects	Command
Two front sensors	Turn left or right at random
Two rear sensors	Move forward
Right sensor only	Turn left
Left sensor only	Turn right

The guided robot follows the force field when nothing is detected by the IR-sensors. When the guiding robot is out of the 0.15 [m] radius from the center of the guided robot, the guided robot follows the force field shown in Fig. 4 (left) and moves toward the center of the field. If the guiding robot is in the circle, then this force field changes its flow, as shown in Fig. 4 (right). Fig. 4 (right) shows the force field when the guiding robot approaches the guided robot from the downward direction. Even if the relative positions of the robots are the same, the guided robot moves differently based on its absolute position in the field.

4 Results

4.1 Capture Task

The obtained capture rate is shown with the standard deviation error bars in Fig. 5. In addition, a 200 trial moving average is shown in Fig. 6 in order to provide detail. Based on the graphs shown in Figs. 5 and 6, we confirmed that the proposed system could achieve a higher success rate than the conventional systems, SARSA1 and SARSA2. The dotted lines in Fig. 5 show references for the performance when the robot performs optimal actions. The robot of Reference 1 does not observe the sign of u and follows the feedback control toward $y(t)$. Therefore, Reference 1 shows the maximum capture rate when the robot follows $M_{\{x,y\}}$. The robot of Reference 2 observes the sign of u and follows the feedback control to $y(t)+u$. Reference 2 shows the maximum capture rate when the robot follows $M_{\{x,y,\dot{y}\}}$.

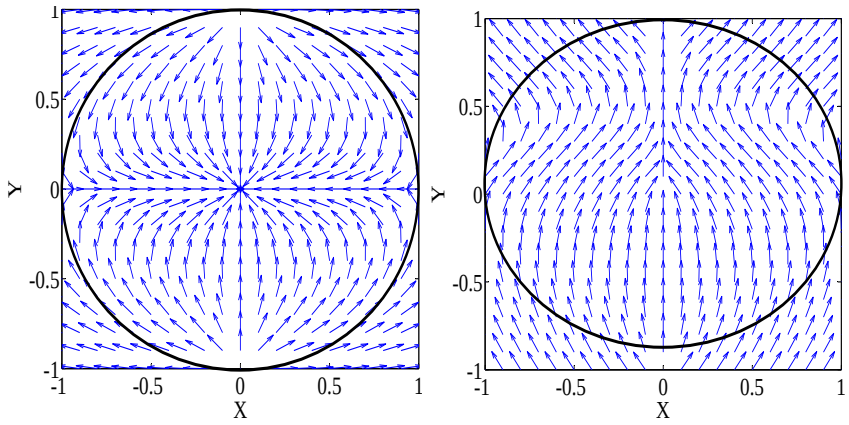

Fig. 4. Force field (left) and force field while avoiding the guiding robot (right)

In order to analyze the role of each layer of the proposed system, we focused on the domination of each layer. When the first layer dominates the action of the robot, the robot moves based only on input (x, y) because the first layer does not get input $\dot{y}$. In this case, the robot is following an MDP $M_{\{x,y\}}$. When the second layer dominates the action of the robot, the robot moves based on input $(x, y, \dot{y})$. In this case, the robot is following another MDP $M_{\{x,y,\dot{y}\}}$. We analyzed the dominations of the layers. We define the action selection of the first layer, $a_1(x, y, \dot{y})$, as whole system's action when Q_2 is neglected ($Q_2 = 0, Q = Q_1$). If the action selection of the first layer, a_1, is equal to the action selection of the whole system, a, then the first layer is the dominant controller of the whole system. Therefore, the whole system is using $M_{\{x,y\}}$ to capture the agent in this case. On the other hand, if $a_1(x, y, \dot{y})$ is modified by the second layer, $a \neq a_1$, then the first layer is no longer the dominant controller of the system. In this case, the whole system may use $M_{\{x,y,\dot{y}\}}$ to decide its action. To confirm the use of $M_{\{x,y,\dot{y}\}}$, we introduced Reference 1 of Fig. 5. We define dominance of the first layer, D_1, as follows:

$$D_1 = \sum_{\dot{y}} \delta_{a(x,y,\dot{y}),a_1(x,y)} \tag{15}$$

where δ is a Kronecker delta. We show the dominance of the robot in Fig. 7. We confirmed that the robot Learned to switch the dominance.

4.2 Guiding Task

We compared SARSA1, SARSA2, and the proposed system using the same learning parameters as those used in the capture task. The success rate of the proposed system tended to be higher than those of the other systems (Fig. 8).

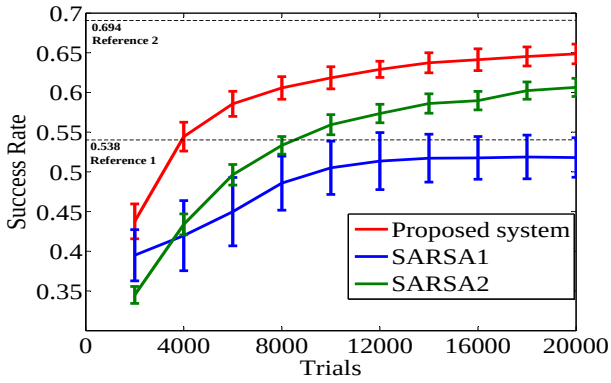

Fig. 5. Capture rate (2,000 trial moving average)

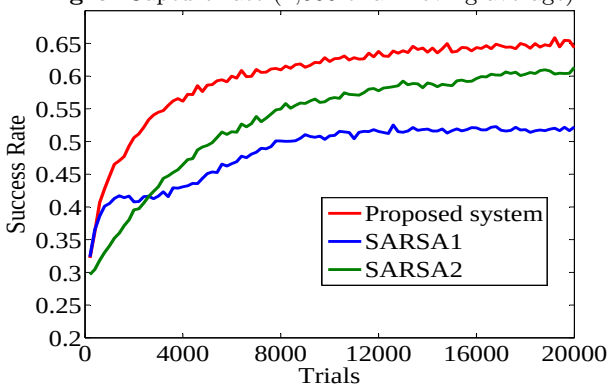

Fig. 6. Capture rate (200 trial moving average)

5 Discussion

5.1 Decoupling MDPs

At the early stage of the capture experiment, the first layer of the proposed RL dominated the action of the robot. So, the robot utilized $M_{\{x,y\}}$ at the early stage. At the last stage, the dominance of the first layer was weak when the agent was near the robot or was almost out of the sight of the robot. The robot utilized $M_{\{x,y\}}$ and $M_{\{x,y,\dot{y}\}}$ according to the situation. Generally, when a human captures an agent like the robot, the human needs the position y of the agent. If the agent is slow and far from us, y is sufficient information to approach the agent. However, if the agent is quick and near to the human, the human may need the speed $\dot{y}$ of the agent. We consider that the robot switched of $M_{\{x,y\}}$ and $M_{\{x,y,\dot{y}\}}$ appropriately according to the situation.

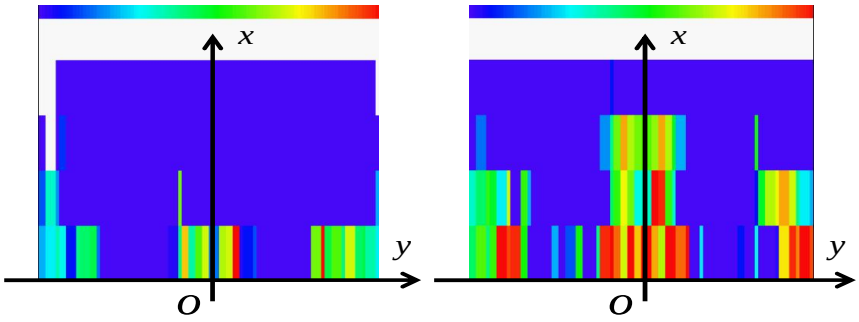

Fig. 7. Attention during the early stage (left) and attention during the last stage (right)

These images show the dominances of the first layer while the robot learns the capture task. We set O as the center position of the robot. Each colored pixel shows the dominance of the first layer. In the color bar at the top of the images, blue indicates that dominance of the first layer is strong (D_1 is high), and red indicates that dominance of the first layer is weak (D_1 is low). At the early stage of learning, the first layer dominated a wide area. This means that the robot focuses on $M_{\{x,y\}}$ and ignores the velocity $\dot{y}$ of the agent. During the final stage of learning, dominance of the first layer is weak at the center, on the left side, and on the right side. While the agent was far from the robot (top of the figure), the robot focused on $M_{\{x,y\}}$. However, when the agent was near the robot or around the limitation of the sight of the robot, the robot focused on the motion of the agent $\dot{y}$ and $M_{\{x,y,\dot{y}\}}$. The proposed learning system realized learning for the switch of two MDPs.

From the analysis for the robot's learning, $M_{\{x,y\}}$ was decoupled from the POMDP at first, and then $M_{\{x,y,\dot{y}\}}$ was decoupled. We consider that the reason why $M_{\{x,y\}}$ was learned faster than $M_{\{x,y,\dot{y}\}}$ is that $M_{\{x,y,\dot{y}\}}$ was more difficult problem, since it includes one more dimension than $M_{\{x,y\}}$.

We consider that the learning process of the robot was step by step as follows. The robot that learned $M_{\{x,y\}}$ followed a sub-optimal policy $\pi_{\{x,y\}}$ at the early stage of learning. The robot searched around $\pi_{\{x,y\}}$ using ϵ-Greedy search to improve the policy. In some observations, the robot found that $M_{\{x,y,\dot{y}\}}$ was more suitable to represent the problem. Then, the robot learned $\pi_{\{x,y,\dot{y}\}}$ during the observations. We may accelerate this learning process using A* search or other kind of model predictive searches, since ϵ-Greedy search is not the best one. However, to utilize model predictive searches, learning system has to make some model of the environment and/or agents while learning Q-values.

5.2 Consideration for a Traditional Theory

Pendrith et al. investigated the conditions for policy stability in non-Markov decision processes. Pendrith et al. mentioned that the TD style of credit assignment method is not guaranteed to have equilibrium points [MM98]. We agree on their consideration. However, in the case when a POMDP includes several MDPs Ms, the TD style of credit assignment method may have equilibrium points of the sub-problem M.

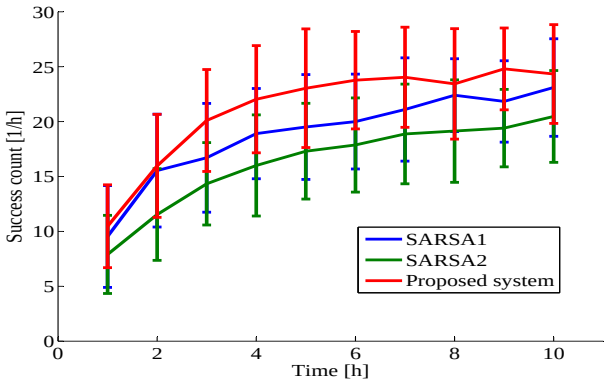

Fig. 8. Success rate of the guiding robot simulation

6 Conclusion

In this paper, we proposed multi-layered RL (MLRL) formulations that decouple a partially observable Markov decision process (POMDP) to several Markov decision processes (MDPs) step by step according to the dimensional hierarchy of the MDPs. From the results of the experiments, we confirmed that a two layered RL that is based on the proposed MLRL formulations decoupled two MDPs, $M_{\{x,y\}}$ and $M_{\{x,y,\dot{y}\}}$, step by step. Also, the MLRL got better success rate and success count than SARSAs in the experiments.

References

[C89] C. J. C. H. Watkins:"Learning From Delayed Reward," Ph.D. thesis of Cambridge University, (1989).

[C92] C. J. C. H. Watkins:"Q-Learning," Machine Learning, Vol. 8, pp. 279-292, (1992).

[RA00] Richard S. Sutton and Andrew G. Barto: "Reinforcement Learning," MIT Press, 55 Hayward Street Cambridge, MA 02142-1493 USA, (2000).

[LM98] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra :"Planning and Acting in Partially Observable Stochastic Domains," ARTIFICIAL INTELLIGENCE, (1998).

[Ke00] Kevin P.Murphy:"A Survey of POMDP Solution Techniques," Technical reports/ informal notes of the author, http://www.cs.ubc.ca/ murphyk/mypapers.html, (2000).

[JS98] John Loch and Satinder Singh:"Using Eligibility Traces to Find the Best Memoryless Policy in Partially Observable Markov Decision Processes," In Proceedings of International Conference on Machine Learning, (1998).

[MM98] Mark D. Pendrith and Michael J. McGarity:"An Analisis of Direct Reinforcement Learning in non-Markovian Domains," in Proceedings of the International Conference on Machine Learning, (1998).

[CT08] Chyon Hae Kim, Tetsuya Ogata, and Shigeki Sugano: "Reinforcement Signal Propagation Algorithm for Logic Circuit," Journal of Robotics and Mechatronics, (2008).

[ED04] Erfu Yang and Dongbing Gu: "Multiagent Reinforcement Learning for Multi-Robot Systems: A Survey," University of Essex Technical Report, (2004).

[Ge03] Gerald Tesauro: "Extending Q-Learning to General Adaptive Multi-Agent Systems," Advances in Neural Information Processing Systems, (2003).

[PG93] Peter Dayan and Geoffrey E. Hinton: "Feudal Reinforcement Learning," Advances in Neural Information Processing Systems, (1993).

[Mo91] Morgan Kaufmann: "Variable Resolution Dynamic Programming: Efficiently Learning Action Maps in Multivariate Real-valued State-spaces," In Proceedings of the International Conference of Machine Learning, (1991).

[RA01] Remi Munos and Andrew Moore: "Variable Resolution Discretization in Optimal Control," Machine Learning, (2001).

[Ri96] Richard S. Sutton: "Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding," Advances in Neural Information Processing Systems, (1996).

[Ha06] Hajime Kimura: "Reinforcement Learning in Multi-Dimensional State-Action Space using Random Tiling and Gibbs Sampling," Transaction of the Society of Instrument and Control Engineers, (2006) (in Japanese).

[MR98] Marco Wiering, Rafa L Salustowicz, and Jürgen Schmidhuber: "CMAC Models Learn to Play Soccer," In Proceedings of the International Conference on Artificial Neural Networks, (1998).

[RN98] Richard Vaughan, Neil Sumpter, Andy Frost, and Stephen Cameron: "Robot Sheepdog Project Achieves Automatic Flock Control," Proc. of the International Conference on Simulation of Adaptive Behavior, (1998).

[RN00] Richard Vaughan, Neil Sumpter, Jane Henderson, Andy Frost, and Stephen Cameron: "Experiments in Automatic Flock Control," Robotics and Autonomous Systems, Vol. 31, pp. 109-117, (2000).

[AJ96] Alan C. Schultz, John J. Grefenstette, and William Adams: "RoboShepherd: Learning a Complex Behavior," In Proceedings of the Robots and Learning Workshop, pp.105-113, (1996).

[OP00] Olivier Sigaud and Pierre Gérard: "Using Classifier Systems as Adaptive Expert Systems for Control," Lecture Notes in Computer Science, pp. 138-157, (2000).

[MS96] Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda: "Purposive Behavior Acquisition for a Real Robot by Vision-based Reinforcement Learning," Machine Learning, Vol. 23, pp. 279-303, (1996).

[URL] http://www.cyberbotics.com/