# Reinforcement Learning with Multiple Heterogeneous Modules: A Framework for Developmental Robot Learning

Eiji Uchibe
Neural Computation Unit
Initial Research Project
Okinawa Institute of Science and Technology
12-22, Suzaki, Uruma, Okinawa 904-2234, Japan
Email: uchibe@irp.oist.jp

Kenji Doya
Dept. of Computational Neurobiology
ATR Computational Neuroscience Labs.
2-2-2 Hikaridai, "Keihanna Science City,"
Kyoto 619-0288, Japan
Email: doya@irp.oist.jp

***Abstract*— Developmental learning approach by changing the internal state representation from simple to complex is promising in order for a robot to learn behaviors efficiently. We have proposed a Reinforcement Learning (RL) method for multiple learning modules with different state representations and algorithms. One of interesting results we showed is that a complex RL system can learn faster with the help of simpler RL systems that can not obtain the best performance. However, it did not consider the difference in sampling rates of learning modules. This paper discusses how the interaction among multiple learning modules with different sampling rates affects the robot learning. Experimental results in navigation task show that developmental learning described above is not always good strategy.**

## I. Introduction

Reinforcement Learning (RL) [1] is an attractive learning framework with a wide range of possible application areas. However, when we apply RL methods naively to robotic tasks, it often takes prohibitively long time to obtain good policy (controller). One promising approach is to develop or switch the perceptual and behavioral capabilities of the robot from simple to complex state through the learning process, suggested by recent studies about developmental robotics [2], [3], [4].

We think that the key for successful developmental learning is interaction among multiple modules with different state representation, learning algorithm, and meta-parameters. Recent neuroscience research revealed that there are parallel reinforcement learning pathways in the human brain, each specialized for reward prediction at different time scales [5]. We have proposed a CLIS (Cooperative-Competitive-Concurrent Reinforcement Learning with Importance Sampling) [6]. One of the interesting results was that a complex RL system can learn much faster with the help of simpler RL systems that can not obtain the best performance.

However, we assume that all modules are updated synchronously. Therefore, CLIS did not take into account the difference in sampling rates (control cycle) of learning modules. This difference in sampling rates is typically due to the computational complexity of the module. In general, sampling rate significantly affects the original robot performance. For example, the robot should slow down its moving speed if it uses complex module. Little attention, however, has been paid to the effects of the difference in sampling rates in the field of developmental robotics.

This paper discusses how the interaction among multiple modules sharing the same sensory-motor system affects the learning process. Each module has its own state representation and sampling rate. Importance sampling [7] is used to fill the gap between the behavior policy and the target policy while rate conversion by interpolation deals with the difference in sampling rates. We apply the proposed method to a simple navigation task by using a mobile robot called Cyber Rodent [8]. CR has three heterogeneous learning modules $\mathcal{M}_{\rm in}$, $\mathcal{M}_{\rm vi}$, and $\mathcal{M}_{\rm pl}$. The first module $\mathcal{M}_{\rm in}$ is an embedded module, and the stochastic policy of $\mathcal{M}_{\rm in}$ is given by a human designer. The second module $\mathcal{M}_{\rm vi}$ tries to obtain mapping from immediate sensory information to motor commands. The third module $\mathcal{M}_{\rm pl}$ learns behaviors based on the result of spatial learning. Although the world will be partially observable for $\mathcal{M}_{\rm vi}$, the update rate of $\mathcal{M}_{\rm vi}$ is higher than that of $\mathcal{M}_{\rm pl}$. On the contrary, this navigation task is completely observable for $\mathcal{M}_{\rm pl}$ because the spatial learning uniquely identifies the world's state. Experimental results show that one-way development from simple to complex is not always good strategy for developmental learning.

## II. CLIS Architecture

### A. Competitive Policy Selection

The robot has $N$ learning modules $\mathcal{M}_i$ $(i = 1, \ldots, N)$ based on reinforcement learning. Let denote $\boldsymbol{x}$ and $\boldsymbol{u}$ are the state and action, respectively. Each module has a stochastic policy $\pi_i(\boldsymbol{x}, \boldsymbol{u})$ and a state value function $V_i(\boldsymbol{x})$. $\pi_i(\boldsymbol{x}, \boldsymbol{u})$ represents the probability to select $\boldsymbol{u}$ at $\boldsymbol{x}$. The value of a

state $\boldsymbol{x}$ under the policy $\pi_i$ is given by

$$V_i(\boldsymbol{x}) = \mathrm{E}_{\pi_i}\{R(\boldsymbol{x})\} = \mathrm{E}_{\pi_i}\left\{\sum_{k=0}^{\infty}\gamma_i^k r(t+k)|\boldsymbol{x}(t)=\boldsymbol{x}\right\}, \tag{1}$$

where $\mathrm{E}_{\pi^i}$ denotes the expected value given that the robot follows policy $\pi^i$. $R(\boldsymbol{x})$ is the weighted sum of rewards starting from $\boldsymbol{x}$, called return. $\gamma^i$ is a parameter, $0 \leq \gamma_i \leq 1$, called a discount rate. $r(t)$ is an immediate scalar reward at the discrete time $t$. After $M$ episodes are experienced, the estimated state value function is represented by

$$V_i(\boldsymbol{x}) = \frac{1}{M}\sum_{m=1}^{M} R_m(\boldsymbol{x}), \tag{2}$$

where $R_m(\boldsymbol{x})$ is the actual return in the $m$-th episode.

The robot selects the policy of one module following the probability

$$\Pr(i|\boldsymbol{x}) = \frac{\exp\left[\beta(1-\gamma_i)V_i(\boldsymbol{x})\right]}{\sum_{j=1}^{N}\exp\left[\beta(1-\gamma_j)V_j(\boldsymbol{x})\right]}, \tag{3}$$

where $\beta$ is a positive parameter called the inverse temperature. Low $\beta$ causes (nearly) equi-probable selection of all modules, while high $\beta$ causes selection of the module with the highest value with probability close to one.

### *B. Cooperative Learning by Importance Sampling*

Let us consider how to train all modules $i = 1, \ldots, N$ using an episode obtained by a selected stochastic policy $\pi_{i'}$. Later, we call $\pi_{i'}$ *behavior policy*. Studies of "off-policy" reinforcement learning suggest that it is possible and sometimes better to estimate the value function of a policy $\pi_i$ by following a behavior policy $\pi_{i'}$ $(i \neq i')$. The choice of distribution obviously makes a difference to the efficiency of the method. Importance Sampling is a method to compensate the mismatch between the behavior policy $\pi_{i'}$ and the target policies $\pi_i$ $(i = 1, \ldots, N)$.

Let $h_m$ be the sequence of states, actions, and rewards of the $m$-th episode generated by $\pi_{i'}$:

$$\begin{aligned} h_m = \{&\boldsymbol{x}(0), \boldsymbol{u}(0), r(\tau_{i'}), \boldsymbol{x}(\tau_{i'}), \boldsymbol{u}(\tau_{i'}), r(2\tau_{i'}), \ldots, \\ &\boldsymbol{x}((n-1)\tau_{i'}), \boldsymbol{u}((n-1)\tau_{i'}), r(n\tau_{i'}), \boldsymbol{x}(n\tau_{i'})\}, \end{aligned} \tag{4}$$

where $\tau_{i'}$ is a sampling rate of the selected module. After $M$ episodes are experienced, the Monte Carlo estimate of the state value function is given by [1]

$$V_i(\boldsymbol{x}) = \frac{1}{M}\sum_{m=1}^{M} R_m(\boldsymbol{x})\frac{\Pr(h_m|\pi_i)}{\Pr(h_m|\pi_{i'})}, \tag{5}$$

where $M$ is the number of episodes. $\Pr(h_m|\pi_i)$ and $\Pr(h_m|\pi_{i'})$ are the probabilities that the episode $h_m$ occurs by following $\pi_i$ and $\pi_{i'}$, respectively. $\Pr(h_m|\pi_i)/\Pr(h_m|\pi_{i'})$ is given by

$$\frac{\Pr(h_m|\pi_i)}{\Pr(h_m|\pi_{i'})} = \prod_{l=0}^{n-1}\frac{\pi_i(\boldsymbol{x}(l\tau_i), \boldsymbol{u}(l\tau_i))}{\pi_{i'}(\boldsymbol{x}(l\tau_i), \boldsymbol{u}(l\tau_i))} = \prod_{l=0}^{n-1}\rho_i(l\tau_i), \tag{6}$$

where $\rho_i$ is the correction factor to deal with the differences between the behavior and the target policies. We only need the ratio $\rho_i$ of the action selection probabilities. Note that the behavior policy should satisfy $\pi_{i'}(\boldsymbol{x}, \boldsymbol{u}) > 0$ if $\pi_i(\boldsymbol{x}, \boldsymbol{u}) > 0$.

Eq. (6) will often increase rapidly over time, especially if the behavior policy $\pi_{i'}$ and the target policy $\pi_i$ are very different each other. In this case, importance sampling may be numerically unstable. Therefore, we introduce a small positive constant $\varepsilon$,

$$\tilde{\rho}_i(l\tau_i) = \frac{\pi_i(\boldsymbol{x}(l\tau_i), \boldsymbol{u}(l\tau_i)) + \varepsilon}{\pi_{i'}(\boldsymbol{x}(l\tau_i), \boldsymbol{u}(l\tau_i)) + \varepsilon} \tag{7}$$

to avoid divergence although this approximation causes a bias to the Monte Carlo estimator.

### *C. TD learning with an innate behavior*

It is not realistic that the robot executes various actions randomly. Suppose that a mobile robot learns to navigate in a crowded environment. This mobile robot finally learns collision avoidance behavior after it makes collisions with the obstacles. This is not a problem in computer simulation. However, the collisions with obstacles in the real environment will cause serious hardware troubles in many cases. In order to realize robot learning in the real environment, it is important to introduce a priori knowledge. If the designer has some *a priori* domain-specific knowledge, CLIS enables incorporating it in a hand-made controller, which improves initial performance and learning of other modules.

Since the stochastic policy of the innate module is fixed, we can use the method of TD-learning to evaluate it. Let denote $\boldsymbol{w}_i$ and $\boldsymbol{b}_i(\boldsymbol{x})$ are the weight vector and the basis function vector at the state $\boldsymbol{x}$, respectively. The state value function is approximated by

$$V_i(\boldsymbol{x}) = \boldsymbol{w}_i^T\boldsymbol{b}_i(\boldsymbol{x}),$$

where $\boldsymbol{w}_i^T$ is a transpose of $\boldsymbol{w}_i$. For given $(\boldsymbol{x}(t), \boldsymbol{u}(t), r(t+\tau_i), \boldsymbol{x}(t+\tau_i))$, the weight $\boldsymbol{w}_i$ is updated by using the following rule:

$$\begin{aligned} c_i &\leftarrow \tilde{\rho}_i(t+\tau_i)c_i \\ \delta_i &\leftarrow r(t+\tau_i) + \gamma_i c_i V_i(\boldsymbol{x}(t+\tau_i)) - V_i(\boldsymbol{x}(t)), \\ \boldsymbol{e}_i &\leftarrow \lambda_i\gamma_i\tilde{\rho}_i(t+\tau_i)\boldsymbol{e}_i(t) + c_i\boldsymbol{b}_i(\boldsymbol{x}(t)), \\ \boldsymbol{w}_i &\leftarrow \boldsymbol{w}_i + \alpha_i\delta_i\boldsymbol{e}_i, \end{aligned} \tag{8}$$

where $\alpha_i$ and $\lambda_i$ are the learning rate and trace rate for eligibility, respectively.

### *D. Policy Gradient based Reinforcement Learning*

As a behavior learning module, we have a choice between two major types: Action Value function based Reinforcement Learning (AVRL) and Policy Gradient based Reinforcement Learning (PGRL). PGRL algorithms [9], [10] have recently been re-evaluated since they have some advantages with respect to function approximation and hidden state problems. We omit to explain AVRL used in our previous experiments [6] because we do not use AVRL in the experiments shown in this paper.

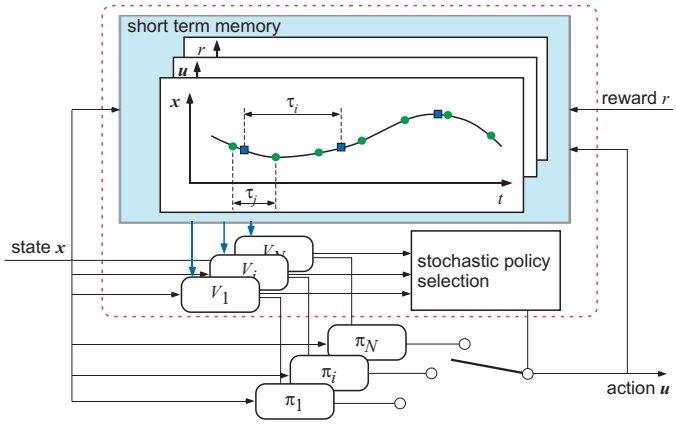


Fig. 1. A multi-module learning system.

PGRL is basically an on-policy method [1] because it estimates the gradient at a given point in the policy space by acting precisely in the manner of the corresponding policy during learning trials. In PGRL, the stochastic policy $\pi_i$ represented by a parameter vector $\boldsymbol{w}_i$ is updated to the direction of the gradient of the expected performance $J_i$,

$$\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i + \alpha_i \frac{\partial J_i(\boldsymbol{w}_i)}{\partial \boldsymbol{w}_i},$$

where

$$J_i(\boldsymbol{w}_i) = \sum_{\boldsymbol{x}} \sum_{h_m} V_i(\boldsymbol{x}) \Pr(h_m|\pi_i),$$

and $V_i(\boldsymbol{x})$ is given by (5). Consequently, the update rules of PGRL with importance sampling are given by

$$\begin{aligned} c_i &\leftarrow \tilde{\rho}_i(t+\tau_i)c_i \\ \boldsymbol{e}_i &\leftarrow \boldsymbol{e}_i + \frac{\partial \ln \pi_i(\boldsymbol{x}(t), \boldsymbol{u}(t))}{\partial \boldsymbol{w}_i}, \\ \boldsymbol{w}_i &\leftarrow \boldsymbol{w}_i + \alpha_i \gamma_i^t r(t+\tau_i) c_i \boldsymbol{e}_i, \end{aligned} \tag{9}$$

where $\boldsymbol{e}_i(t)$ is an eligibility trace for the policy gradient method. Although this implementation does not require a state value function $V_i$ explicitly, we compute $V_i$ by using (8) for the sake of module selection by (3).

There are several ways to implement the stochastic policy. We use a multivariate normal density as follows:

$$\pi_i(\boldsymbol{x}, \boldsymbol{u}) = \eta_i \exp\left(-\frac{1}{2\sigma_i^{2d}} \|\boldsymbol{u} - \boldsymbol{W}_i \boldsymbol{b}_i(\boldsymbol{x})\|^2\right), \tag{10}$$

where $d$ is a dimension of $\boldsymbol{u}$, $\eta_i$ is a normalizing constant, respectively. In this formulation, the policy parameter $\boldsymbol{w}_i$ is $(\boldsymbol{W}_i, \sigma_i)$.

### E. Dealing with Difference in Sampling Rates

CLIS architecture described above assumes that only one sampling rate is used throughout a processing system. Here we show some modifications to deal with difference in sampling rates. Fig. 1 shows the whole architecture of our extended CLIS. Each module determines action *asynchronously* while all modules *synchronously* update parameters by (8) and (9). The extended CLIS is implemented in a multi-threaded program, in which there are $N$ controller threads and one learning thread.

In addition the extended CLIS has a short term memory to save an episode. The actual tuple of $(\boldsymbol{x}, \boldsymbol{u}, r)$ is obtained every $\tau_{i'}$ steps while the state for $\mathcal{M}_i$ is obtained every $\tau_i$ steps. Therefore, we have to resample the data points. We use cubic spline interpolation to estimate them. Finally, we show the setting of the discount rate. In the original CLIS, we use different discount rates to deal with different time scales. In this paper we, however, use the same time scale for simplicity. From the theory of semi Markov decision processes (ex. see [11]) the discount rate of $i$-th module is set to

$$\gamma_i = \gamma^{\tau_{i'}}, \tag{11}$$

where $\gamma$ is a common discount rate. Consequently, each module is evaluated with the same objective function.

## III. Task and Assumptions

### A. Cyber Rodent Hardware

Fig. 2 (a) shows a Cyber Rodent (CR) we have developed in order to investigate learning, developmental, and evolutionary systems from a viewpoint of computational neuroscience and robotics [8]. Especially, the main goal of the CR project is to study the adaptive mechanisms of artificial agents under the same fundamental constraints as biological agents, namely self-preservation and self-reproduction. CR has two main features: the abilities to exchange data and program via an IR-communication port for self-reproduction, and to capture and recharge from battery packs in the environment for self-preservation. Its body is 22 [cm] in length and 1.75 [kg] in weight. A CR is endowed with a variety of sensory inputs, including an omni-directional CMOS camera, an IR range sensor, seven IR proximity sensors, gyros, and accelerometer. Its motion system consists of two wheels that allow CR to move at a maximum speed of 1.3 [m/s]. CR has an FPGA chip for real-time image processing, such as color blob detection.

The task we consider is an autonomous navigation to the stationary destination placed at the center of the experimental field shown in Figs. 2 (b) and (c). There are four short obstacles and two tall obstacles that hide the destination from the robot. Four landmarks are prepared for self-localization described later. A wall is placed around the experimental field.

In order to acquire navigation behavior, we prepare three learning modules: innate module, view-based module, and place-based module. From the visual information about the destination and the landmarks and sensor readings of proximity sensors, the state $\boldsymbol{x}$ is constructed for each module separately. The action $\boldsymbol{u}$ is desired velocities of left and right wheels. Then we explain these modules in detail.

### B. Innate module $\mathcal{M}_{\rm in}$

$\mathcal{M}_{\rm in}$ is an embedded module that gives an initial controller. Therefore, $\mathcal{M}_{\rm in}$ has a fixed stochastic policy $\pi_{\rm in}$ and a value function $V_{\rm in}$ trained by (8). CLIS can easily integrate $\mathcal{M}_{\rm in}$ as an innate module by initializing $V_{\rm in}$ to large values.

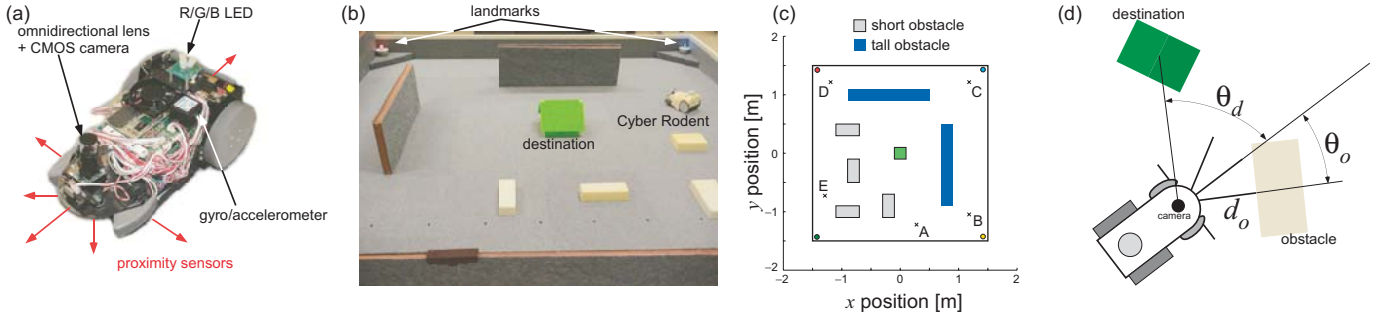


Fig. 2. Environmental configuration. (a) Cyber Rodent hardware. (b) The real environment. (c) Positions of the objects in the environment. Five cross points (A, B, C, D, and E) are the starting positions to evaluate performance during learning. (d) State variables used in $\mathcal{M}_{\rm in}$ and $\mathcal{M}_{\rm re}$.

CR controlled by the stochastic policy of $\mathcal{M}_{\rm in}$ just moves to the destination directly without considering avoiding collisions. If the destination is not observed, it generates random action. Therefore, CR makes a collision when the shorter obstacle is located between CR and the destination. The state $x$ is the relative angle to the destination $\theta_d$ (see Fig. 2 (d), and $j$-th component of the basis function vector is given by

$$b_{{\rm in},j}(\boldsymbol{x}) = \exp\left(-\|\boldsymbol{x}-\theta_j\|^2/\varphi_{\rm in}\right),$$

where $\boldsymbol{x} = [\theta_d]$ and $\varphi_{\rm in}$ is a width of the basis function, respectively. Nine basis functions are uniformly distributed over one dimension angle input space, and one basis function is considered to represent with the situation when the destination gets out of CR's sight field. Sampling rate of $\mathcal{M}_{\rm in}$ is set to 250 milliseconds.

### C. View-based module $\mathcal{M}_{\rm vi}$

$\mathcal{M}_{\rm vi}$ is the simple module. Since this module uses three instantaneous sensor outputs without estimation of the environment, this navigation task is partially observable. However, this module is sometimes applicable if there are no obstacles on the path. In addition, the computational cost is cheap.

State representation is constructed from $\theta_d$ and a distance to the nearest obstacle $d_n$ (see Fig. 2 (d)).

$$b_{{\rm vi},j}(\boldsymbol{x}) = \exp\left(-\|\boldsymbol{x}-\boldsymbol{x}_j\|^2/\varphi_{\rm vi}\right),$$

where $\boldsymbol{x} = [\theta_d, d_o, d_n]$ and $\varphi_{\rm vi}$ is a width of the basis function, respectively. We distribute 81 basis functions uniformly for three dimensional space. Furthermore, one basis function is added to represent with the situation when the destination is invisible. It takes 500 milliseconds to make decisions.

### D. Place-based module $\mathcal{M}_{\rm pl}$

The state used in $\mathcal{M}_{\rm pl}$ is the estimated global position. In the real rat's hippocampus, neurons with spatial firing properties have been reported by several researchers. Although developing a computational model for spatial learning is also important research issue for robotics and computational neuroscience, we just use the existing engineering approach based on the method of *Markov localization* [12]. We adopt a grid-based representation, where the spatial resolution is 15 cm and the angular resolution is 30 degrees. For our environment of size $3 \times 3$ m$^2$ the state space consists of 450 states.

$b_{{\rm pl},j}(\boldsymbol{x})$ $(j = 1,\ldots,450)$ denotes the robot's belief at position $\boldsymbol{x} = \boldsymbol{x}_j$. Here, $\boldsymbol{x} = [x, y, \theta]$ is a location where $x$ and $y$ are Cartesian coordinates and $\theta$ is the robot's orientation. Initially, $b_{{\rm pl},j}(\boldsymbol{x})$ reflects the initial state of knowledge: if the robot knows its starting position, $b_{{\rm pl},j}(\boldsymbol{x})$ is centered on the correct location; if the robot does not know its initial location, $b_{{\rm pl},j}(\boldsymbol{x})$ is uniformly distributed to reflect the global uncertainty of the robot. The latter is the case in all our experiments. When CR make a transition from position $\boldsymbol{x}$ to $\boldsymbol{x}'$ by executed action $\boldsymbol{u}$,

$$b_{{\rm pl},j}(\boldsymbol{x}') \leftarrow \sum_{\boldsymbol{x}} \Pr(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u}) b_{{\rm pl},j}(\boldsymbol{x}),$$

where $\Pr(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})$ denotes the probability obtained from a model of the robot's kinematics. Let $\boldsymbol{s}$ denote sensor readings about the direction of four landmarks, and $\Pr(\boldsymbol{s}|\boldsymbol{x})$ the likelihood of perceiving $\boldsymbol{s}$ at $\boldsymbol{x}$. $\Pr(\boldsymbol{s}|\boldsymbol{x})$ is usually referred to as map of the environment. When CR senses $\boldsymbol{s}$, the belief is updated according to the following rule:

$$b_{{\rm pl},j}(\boldsymbol{x}) \leftarrow \eta \Pr(\boldsymbol{s}|\boldsymbol{x}) b_{{\rm pl},j}(\boldsymbol{x}),$$

where $\eta$ is a normalizer that ensures that the belief $b_{{\rm pl},j}(\boldsymbol{x})$ sums up to 1 over all $\boldsymbol{x}$. $\mathcal{M}_{\rm pl}$ requires $1,200$ milliseconds for making a decision. This module can be regarded as the most complex module.

### E. Other settings

CR is initially placed at the position A of the environment. If CR reaches the destination or the pre-specified time interval (1 minute) expires, the episode is considered to be over and a new one begins. In order for CR to move to the new starting position, CR used a policy $\pi^{\rm wander}$ given by the human designer.

CR receives a positive reward $r = 1$ when it reaches the destination. If, instead, it makes a collision with the obstacle or the wall, a negative reward $r = -0.1$ is given to CR. Otherwise, $r = 0$. Learning is performed in the real environment. We do not control the inverse temperature, that is $\beta = 1$. Other parameters are set as follows: $\varepsilon = 0.2$, $\alpha_{\rm in} = \alpha_{\rm vi} = \alpha_{\rm pl} = 0.1, \gamma = 0.995$. These values were determined by trial and error.

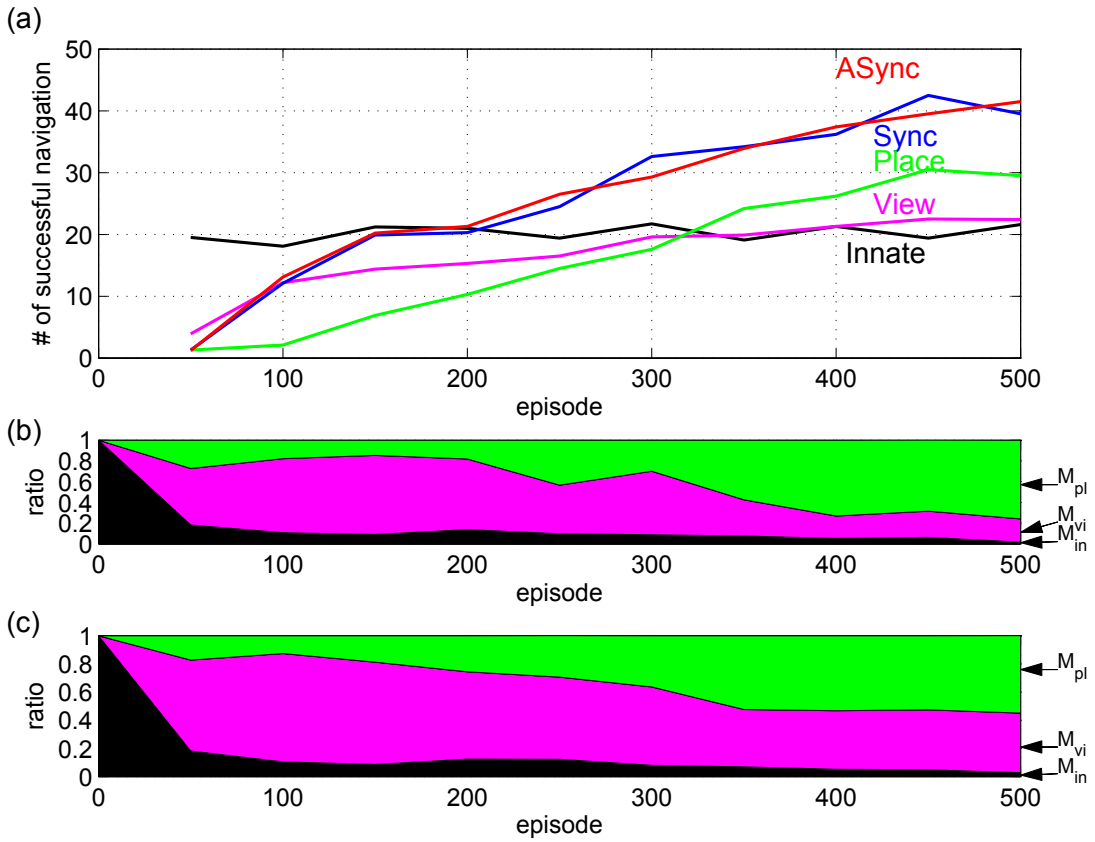


Fig. 3. Experimental results in the navigation task in the real environment. (a) The number of successful navigation in 50 episodes for each method. (b) The ratio for **Sync** to select one out of three modules in 50 episodes. Black, magenta, and green bars represent the ratio of using $\mathcal{M}_{\rm in}$, $\mathcal{M}_{\rm vi}$, and $\mathcal{M}_{\rm pl}$, respectively. (c) The ratio for **Sync** to select one out of three modules. Same color coding is used in (b).

## IV. Experimental Results

### A. Learning curves with respect to the number of episodes

To elucidate the advantage of our proposed method, we test the following three methods: **Innate**, **View**, and **Place**, where they mean that CR is controlled by the module $\mathcal{M}_{\rm in}$, $\mathcal{M}_{\rm vi}$, and $\mathcal{M}_{pl}$, respectively. **Sync** is our previous method [6] in which all modules are updated at the same sampling rate. **Async** is the proposed method taking into account the sampling rates. Since we have already shown the efficiency of the use of importance sampling [6], we did not evaluate the method without importance sampling in this experiment.

Every 50 episodes, we counted the number of successful navigation from five starting points in order to measure the performance. Fig. 3 (a) shows the learning curves of the number of successful navigation. Since the stochastic policy of **Innate** was not changed, their performance did not improved at all. Although **View** achieved the same performance as **Innate** after about 300 episodes, its performance did not exceed that of **Innate**. By observing the obtained behaviors, we found that **View** could avoid collisions with obstacles but it sometimes lost the destination. The performance of **Place** was improved much slower than that of **View** because the search space of **Place** is very large. However, **Place** obtained better performance than **View** did since this navigation task is completely observable problem for **Place**.

It took about 150 episodes for both **Sync** and **Async** to reach the same performance of **Innate**. At the end of learning, they obtained the best performance. Figs. 3 (b) and (c) show the selection ratio during learning. They often used $\mathcal{M}_{\rm vi}$ at the middle of learning. **Sync** often used $\mathcal{M}_{\rm pl}$ at the end while **Async** switched $\mathcal{M}_{\rm vi}$ and $\mathcal{M}_{\rm pl}$ with almost same probability. These results showed that $\mathcal{M}_{\rm pl}$ was trained efficiently by simple module $\mathcal{M}_{\rm vi}$ thanks to CLIS architecture. Note that the appropriate modules were automatically selected according to the progress of learning. Although these results were consistent with our previous experiments [6], there were no significant differences between **Sync** and **Async**. Then, we investigated the original performance in more detail.

### B. Effects on the actual performance

Fig. 4 shows the probability to select $\mathcal{M}_{\rm pl}$ over the location at the end of learning. Since $\mathcal{M}_{\rm in}$ was seldom selected at the end of learning, we can roughly approximate the relation

$$\Pr(\mathcal{M}_{\rm vi}|\boldsymbol{x}) + \Pr(\mathcal{M}_{\rm pl}|\boldsymbol{x}) \simeq 1, \text{for all } \boldsymbol{x},$$

where $\Pr(\mathcal{M}_i|\boldsymbol{x})$ is the probability to select the module $\mathcal{M}_i$ at the state $\boldsymbol{x}$. Therefore, Fig. 4 also suggests the probability

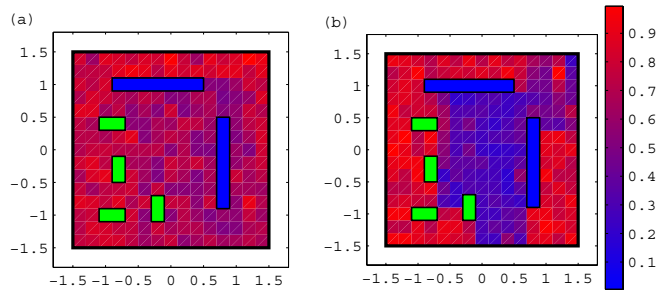


Fig. 4. The probability to select $\mathcal{M}_{\rm pl}$ according to the location at the end of learning. The orientation of CR was set to the direction to the destination. Green bars represent the obstacles. (a) CR is controlled by **Sync**. (b) CR is controlled by **Async**.

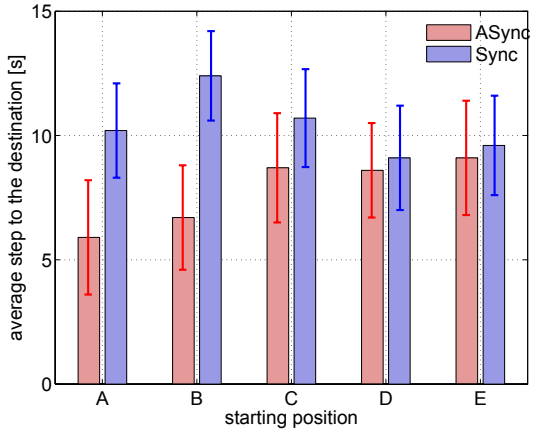


Fig. 5. Average steps for CR to move to the destination.

to select the module $\mathcal{M}_{\rm vi}$. Fig. 4 (a) shows that **Sync** usually used $\mathcal{M}_{\rm pl}$ at the end of learning. On the contrary we found that **Async** sometimes decided to use $\mathcal{M}_{\rm vi}$ according to the location shown in Fig. 4 (b). Roughly speaking, **Async** used $\mathcal{M}_{\rm vi}$ when there were no obstacles on the path to the destination. Since $\mathcal{M}_{\rm vi}$ had a capability to react faster than $\mathcal{M}_{\rm pl}$, $\mathcal{M}_{\rm vi}$ could increase the maximum velocity to be executed. As a result, CR with **Async** moved a little bit quickly when $\mathcal{M}_{\rm vi}$ was available for accomplishing the task.

In order to compare the actual performance, we measured the average time steps from five starting positions to the destination. Fig. 5 shows the comparison of **Sync** and **Async**. When CR started from the positions A, B, and C, On the contrary, we found no significant differences in the elapsed time from the starting positions D and E between **Async** and **Sync**. These experimental results suggested that the extended CLIS could improve its original performance performance by considering the reaction time.

## V. Discussion

This paper proposed the developmental learning methods for the robot with multiple heterogeneous learning modules, in which each module updates the internal parameters in a different sampling rate. We have applied our proposed method to the navigation task. Experimental results showed that the learning module with the simple state representation was important because

- it helped the learning module with the simple state representation by collecting meaningful experience, and
- it was selected to improve the original robot performance if the module with the simple state representation is possible to accomplish the task.

We claim that one-way development from the simple to the complex state representation is not always good strategy for developmental learning. In other words, multiple state representations should be used simultaneously for successful development.

We plan to integrate the proposed method with a spatial learning mechanism based on models of the rodent hippocampus [13]. Although CLIS is not plausible computational model from a viewpoint of computational neuroscience, we believe that CLIS gives us the first step to elucidate developmental learning system suited for real animals.

## References

[1] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, MIT Press / Bradford Books, 1998.
[2] M. Asada, K. F. MacDorman, H. Ishiguro, and Y. Kuniyoshi, "Cognitive developmental robotics as a new paradigm for the design of humanoid robots," Robotics and Autonomous Systems, vol. 37, pp. 185–193, 2001.
[3] M. Lungarella, G. Metta, R. Pfeifer, and G. Sandini, "Developmental Robotics: A Survey," Connection Science, Vol. 4, pp. 151–190, 2003.
[4] J. Weng, "Developmental robotics: Theory and experiments," International Journal of Humanoid Robotics, vol. 1, no. 2, pp. 199–236, 2004.
[5] S. Tanaka, K. Doya, G. Okada, K. Ueda, Y. Okamoto, and S. Yamawaki, "Prediction of immediate and future rewards differentially recruits cortico-basal ganglia loops," Nature Neuroscience, vol. 7, pp. 887–893, 2004.
[6] E. Uchibe and K. Doya, "Competitive-Cooperative-Concurrent Reinforcement Learning with Importance Sampling," in Proc. of International Conference on Simulation of Adaptive Behavior: From Animals to Animats, 2004, pp. 287–296.
[7] R. Rubinstein, Simulation and the Monte Carlo method, New York: Wiley, 1981.
[8] K. Doya and E. Uchibe, "The Cyber Rodent project: Exploration of adaptive mechanisms for self-preservation and self-reproduction," Adaptive Behavior, (in press).
[9] J. Baxter and P. L. Bartlett, "Infinite-horizon gradient-based policy search," Journal of Artificial Intelligence Research, vol. 15, 2001, pp. 319–350.
[10] R. S. Sutton, D. McAllester, S. Sing and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," Advances in Neural Information Processing Systems 12, 2000, pp. 1057–1063.
[11] M. L. Puterman, "Markov Decision Problems," Wiley, New York, 1994.
[12] D. Fox, W. Burgard, and S. Thrun, "Active Markov localization for mobile robots," Robotics and Autonomous Systems, vol. 25, no. 3-4, pp. 195–207, 1998.
[13] O. Trullier, S. I. Wiener, A. Berthoz and J.-A. Meyer, "Biologically-based artificial navigation systems: Review and prospects," Progress in Neurobiology, vol. 51, pp. 483–544, 1997.