International Journal of Control, Automation, and Systems (2011) 9(2):328-338 http://www.springer.com/12555
DOI 10.1007/s12555-011-0215-2

SSPQL: Stochastic Shortest Path-based Q-learning

Woo Young Kwon, Il Hong Suh*, and Sanghoon Lee

Abstract: Reinforcement learning (RL) has been widely used as a mechanism for autonomous robots to learn state-action pairs by interacting with their environment. However, most RL methods usually suffer from slow convergence when deriving an optimum policy in practical applications. To solve this problem, a stochastic shortest path-based Q-learning (SSPQL) is proposed, combining a stochastic shortest path-finding method with Q-learning, a well-known model-free RL method. The rationale is, if a robot has an internal state-transition model which is incrementally learnt, then the robot can infer the local optimum policy by using a stochastic shortest path-finding method. By increasing state-action pair values comprising of these local optimum policies, a robot can then reach a goal quickly and as a result, this process can enhance convergence speed. To demonstrate the validity of this proposed learning approach, several experimental results are presented in this paper.

Keywords: Model-based RL, reinforcement learning, robot learning, stochastic shortest path.

1. INTRODUCTION

It is essential for autonomous robots to learn a given task and how to do it by employing limited knowledge and resources effectively in uncertain environments where measurement and movement errors can occur. Reinforcement learning (RL) is a technique that allows an agent to learn an optimal strategy through trial-and-error interaction with a dynamic environment [1]. It is known to be adequate for learning necessary state-action pairs to achieve task completion without prior knowledge of the working environment. As such, an RL-based robot can deduce state-action pairs to accomplish a given task without an explicit model of the environment. Thus, many researchers [2-8] have applied the reinforcement learning technique to real robots without supplying prior knowledge of the working environment.

RL can be classified into two categories: model-free and model-based methods. In model-free methods, an agent learns an action-value function without explicitly representing a model of the environment. In other words, model-free RL methods directly estimate optimal action-value functions using trials [9]. For examples, temporal deference learning [10] and Q-learning [11] are well known model-free RL methods. These model-free methods require less computational time, memory, and effort to implement than model-based reinforcement learning methods. Moreover, model-free RL methods work well in changing environments as they do not require an environment model.

However, these model-free methods are extremely inefficient in the use of gathered data and therefore often require a great deal of data from trial-and-error to achieve good performance [12]. This limits them to the application of model-free reinforcement learning to low-level control problems in which there is a small scale of state space, and/or there is a low cost of interaction with the environment [13-16].

By contrast, model-based reinforcement learning techniques perform offline simulations prior to action-selection by exploiting an internal model, which can reduce unnecessary exploration. As the cost of performing actions for a real robot is higher than the cost of simulating action using the internal model, it is thus effective for a real robot to improve its policy using offline simulations [17-20].

In Dyna-Q, a well known model-based RL method proposed by Sutton [21], a number of offline simulations are carried out inbetween action executions to update action-value functions. Real-time dynamic programming (RTDP) [22,23] is another model-based learning approach that uses a heuristic search to find optimal action-value functions in stochastic environments. It is a stochastically generalized version of the online heuristic search algorithm, Learning Real Time A* [24,25]. RTDP is a popular online method for solving the stochastic shortest path (SSP) problem that exist as a subset of Markov decision processes (MDPs) that have positive costs and an absorbing goal state.

These model-based RL methods show improved convergence speed when compared to model-free RL techniques. As a result, model-based reinforcement learning methods have been applied to various robot

Manuscript received March 31, 2009; revised June 22, 2010 and October 14, 2010; accepted October 21, 2010. Recommended by Editorial Board member Sooyong Lee under the direction of Editor Jae-Bok Song. This work was supported by the Global Frontier R&D Program on <Human-centered Interaction for Coexistence> funded by the National Research Foundation of Korea grant funded by the Korean Government(MEST) (NRF-M1AXA003-2010-0029744)

Woo Young Kwon, Il Hong Suh, and Sanghoon Lee are with the Department of Electronics & Computer Engineering, Hanyang University, 17 Haengdang-dong, Seongdong-gu, Seoul 133-791, Korea (e-mails: wykwon@incorl.hanyang.ac.kr, ihsuh@hanyang.ac.kr, and shlee@incorl.hanyang.ac.kr).
* Corresponding author.

tasks such as mobile robot applications [26-29] and bipedal walking robots [30,31]. In these model-based RL methods, the agent learns the state-transition-model. After the model has been learnt, the agent can then compute the optimal action-value function. Thus, these types of RL methods can also be regarded as indirect RL methods.

Although model-based RL methods are advantageous in terms of learning speed, they mainly perform well in stationary environments. As they are indirect methods, model-based RL can create problems in dynamic environments. Once a robot learns the state-transition model, the model cannot be easily changed. For a static environment, the persistence of this learnt model does not create problems. However, if the environment changes, a learnt model may obstruct the learning process in the new environment. Unfortunately, a real robot usually learns tasks in dynamic and uncertain environments where uncertain sensory information and irregular short-term feedback are found. For example, an environment with robots and humans is dynamic and uncertain because it is hard to predict human behavior. In such an environment, a person can often become an obstacle for a robot llearning a given task. On the contrary, a person may play the role of external trainer for a robot, i.e., the person can give short-term feedback to a robot in order to teach it an effective action-strategy. This short-term feedback is irregular and changeable. Consequently, model-based RL methods are slow to adapt dynamic environments.

An autonomous robot must satisfy two conflicting properties: fast learning speed and adaptability in a dynamic environment. Model-free RL methods have the advantage of being adaptable in dynamic environments, whereas model-based RL methods have the advantage of quick learning. By combining these two types of RL methods, the two conflicting properties can be resolved. In this paper, a novel learning method that combines model-free and model-based RL methods is proposed to improve both learning speed and adaptability in dynamic environments. The Q-learning algorithm was adopted as the model-free RL method, and an extension of the SSP-finding method [32,33] was used as the model-based learning method.

This paper is organized as follows. Section 2 will present an SSP-based Q-learning algorithm that combines both model-free and model-based RL methods. Section 3 will present comparisons to the proposed learning method with other RL methods. Section 4 will present the experimental results, and concluding remarks will follow in Section 5.

2. SSPQL: STOCHASTIC SHORTEST PATH-BASED Q-LEARNING

To improve both learning speed and adaptability in dynamic environments, an integrated method that combines an SSP-finding method (a model-based learning method) with a Q-learning method (a model-free learning method) was proposed. Despite numerous advantages, Q-learning has a serious disadvantage within robotic applications, namely a slow learning speed. However, this disadvantage can be mitigated through the assistance of an external trainer. If a trainer teaches a relevant action with respect to a given state, then learning speed can be increased. The basic idea is to improve the learning speed of Q-learning by using an SSP-finding method that plays the role of trainer. We call this combined method Stochastic Shortest Path-based Q-learning (SSPQL).

2.1. Q-learning

Q-learning is a well-known reinforcement learning technique that learns an action-value function which calculates the expected utility of taking a given action in a given state. The core of the algorithm is a simple value iteration equation given by

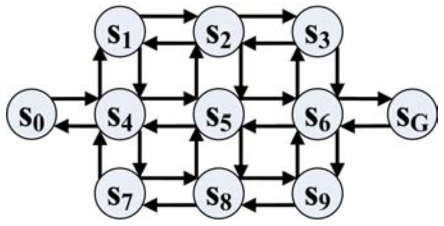
$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)\right], \quad (1)$$

where $Q(s,a)$ is an action-value function of an action a given a state s. r is the reward value, α is the learning rate, and γ is the discount factor. Moreover, A is a set of actions and s' is the state followed by an action a' in a state s. The strength of Q-learning is to learn the optimal policy without requiring a model of the environment. Moreover, it has been proven that the action-value function in Q-learning converges eventually. However, Q-learning may present some difficulties when applied to a real environment: firstly, rewards may not be given immediately after an action is completed. Typically, rewards are given after a goal is achieved through a sequence of actions. In general, this delayed rewarding makes the learning very time-consuming, or even impractical. To apply Q-learning in a real environment, it is necessary to reduce this long learning time mainly due to delayed rewards.

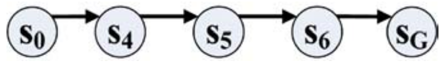
2.2. Stochastic shortest path-finding method

To overcome the slow learning speed of model-free Q-learning, it can be useful for an external trainer to raise the probabilities of important state-action pairs. Although the trainer has only partial knowledge of the working environment, this can still be useful to improve learning speed. A robot can learn an internal state-transition model incrementally, and can infer the important state-action pairs composing of the local optimum strategy from the model. These important state-action pairs can then act as the external trainer and be used to increase learning speed. In the SSPQL, we use an SSP-finding method to obtain these important state-action pairs. By using an internal state-transition model that has been incrementally learnt by the robot, the stochastic shortest path-finding method can then suggest local optimum state-action pairs. In the SSPQL, these local optimum state-action pairs can be given a higher probability of selection by increasing the Q-value corresponding to each state-action pairs.

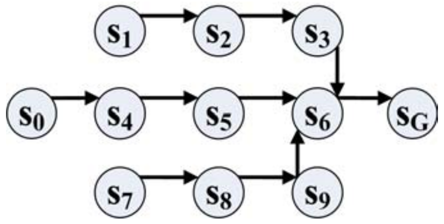
Given a state-transition model learnt by experience as shown in Fig. 1(a), the single-pair shortest path from the



(a) An example of a state transition model.



(b) A shortest path from s_0 to s_G.



(c) Shortest paths from all states to a goal state.

Fig. 1. An example of shortest path finding.

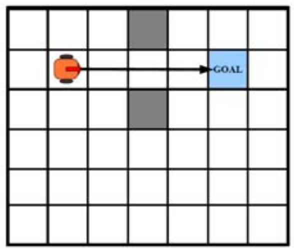
initial state, s_0, to the goal state, s_G, is given by Fig. 1(b). A* is a well known effective heuristic search method for this type of problem. However, our objective is to increase the probabilities of every action that can be obtained from the internal model.

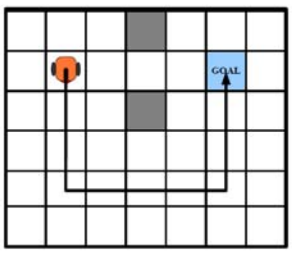
To increase the probabilities of all important state-action pairs, it is necessary to solve the single-destination shortest path problem, whereby the shortest paths from all states in the model to a single goal state are found as shown in Fig. 1(c).

There are several algorithms that find the shortest path, such as the Bellman-Ford algorithm, and Dijkstra's algorithm for deterministic environments. However, these algorithms are not suitable for real robotic applications because they work under the assumption of deterministic state-transitions. For example, a very low-cost path with low probability may be obtained by noisy sensor data and/or non-deterministic state-transitions. A shortest path finding method based on a deterministic state-transition model may misunderstand this low-cost path as an optimal path. Once these paths are stored in the deterministic state-transition model, these misunderstood paths will continuously affect action-selection. To avoid this problem, the stochastic state transition should be considered. Fig. 2 shows two examples of stochastic state transition. On one hand, the path in Fig. 2(a) is short but risky as a robot may encounter obstacles (i.e., a gray cell). On the other hand, the path in Fig. 2(b) is a long but safe path. If a state-transition is more stochastic, the path in Fig. 2(b) might be considered a good path.

In the stochastic environment, it is necessary to consider both the uncertainty of state-transitions and actual costs such as time, energy, and distance. The expected cost of a state-transition from state s to s' is defined as

$$EC(s,s') = weight(s,s')e^{-\lambda P(s'|s)}, \tag{2}$$





(a) Path with short distance, high risk. (b) Path with long distance, low risk.

Fig. 2. Comparison of expected costs in a stochastic state transition.

where *weight*(s,s') is an actual cost function of state-transition from s to s' and λ is a parameter to reflect the state-transition probability.

Moreover, $P(s'|s)$ is the probability of a state transition from s to s' given by

$$P(s'|s) = \sum_{a\in A} P(s'|s,a), \tag{3}$$

where A is a set of all available actions. Next, the state-transition probability to s' given s and a is given by

$$P(s'|s,a) = \begin{cases} N(s,a,s')/N(s,a) & \text{if } N(s,a) \neq 0 \\ \mu(\mu \approx 0) & \text{if } N(s,a) = 0, \end{cases} \tag{4}$$

where $N(s,a,s')$ is the number of observations of s' followed by action a in a state s, μ is a small positive value, and $N(s,a)$ is the frequency of an action a in a state s.

By using the expected cost, the stochastic shortest paths to avoid low-cost but risky paths can be found. Here, Dijkstra's algorithm [34] is used in order to find stochastic shortest paths based on the expected cost. After applying Dijkstra's algorithm, the set of edges which compose the SSP can be obtained. To combine the SSP with Q-learning, we need an SSP-based action-value function, which is represented hear as $Q_{SSP}(s,a)$. If s and a are contained in the SSP, the value of $Q_{SSP}(s,a)$ must be positive; otherwise, it becones to zero. To reflect the uncertainty of actions, the entropy of action must be considered for $Q_{SSP}(s,a)$. Therefore, the SSP-based action-value function is given by

$$Q_{SSP}(s,a) = \begin{cases} e^{-H(s,a)} & \text{If } s \text{ and } s' \in SSP, a = \arg\max_{a'\in A} P(s'|s,a') \\ 0 & \text{else,} \end{cases} \tag{5}$$

where $H(s,a)$ is the entropy of action given by

$$H(s,a) = -\sum_{s'\in S} P(s'|s,a)\log P(s'|s,a). \tag{6}$$

Here, S is a set of states.

Algorithm 1 shows the overall algorithm of the SSP-finding method, where V is the vertex-set, E is the edge-set, w is the edge weighting, and A is the set of actions.

Algorithm 1: function *SSP(V, E, w, goal)*

```
1: for all state $s \in V$, action $a \in A$ do
2:     $Q_{SSP}(s,a)$:=0 {Initializing SSP-value}
3: end for
4: for all edge $e \in E$ do
5:     $s$:= a source vertex of the edge $e$
6:     $s'$:= a target vertex of the edge $e$
7:     $EC(s,s')=H(s,s')\cdot(s,s')$ {Updating expected costs}
8: end for
9: SSPedges:= Dijkstra(V, E, w, goal)
10: for all edge $e$ in the edge-set of SSPedges do
11:     $s$:= a source vertex of the edge $e$
12:     $s'$:= a target vertex of the edge $e$
13:     $a$:= $\arg\max_{a' \in A} P(s'|s,a)$
14:     $Q_{SSP}(s,a) = \exp(-H(s,a))$ {Updating SSP value}
15: end for
16: return $Q_{SSP}$
```

2.3. Exploration bonus

The exploration vs. exploitation problem has received significant attention from the reinforcement learning research community, and several directed/undirected methods have been developed to incorporate exploration strategies into reinforcement learning agents. Undirected methods such as ε-greedy and Boltzmann exploration are based on incorporating randomness into the decision-making procedure by assigning each action a positive probability of being selected and performed. However, directed methods use interaction history to evaluate the expected contribution of every action to exploration [35].

In this paper, we use action-counts where higher exploration bonuses are given to actions that have been chosen less frequently. The bonus estimates the expected contribution of the action to the exploration. In this paper, a simple counter-based exploration bonus is defined as

$$Q_{EXP}(s,a) \equiv \frac{1}{(N(s,a)+1)^{\eta}}, \tag{7}$$

where $N(s,a)$ is the frequency of an action a in a state s, and η is the weighting parameter of Q_{EXP}. An exploration bonus shows a somewhat different property from other action-value functions as it gives higher priority values to actions that will lead to unexplored states, and smaller priority values to a state that is frequently visited. From an experimental viewpoint, an exploration bonus is more influential than a value update.

2.4. Integrating SSP-finding method with Q-learning

Given a perfect state-transition model, optimal state-action pairs can be found to accomplish a task using the SSP method. However, it is hard to provide a perfect state-transition model of a real environment to a robot. In many cases, an incomplete and/or even incorrect state-transition model is provided. Thus, a robot should learn a state-transition model of its environment by interacting with the environment. Our SSPQL algorithm simultaneously learns and uses the state-transition model by combining the SSP method and the Q-learning method. To learn the state-transition model, a previous observed state, a previous action and a current state must be stored in a probabilistic state-transition model at every step. In the early stages of learning, the state-transition model may be almost empty. As a robot explores its environment by using an exploration strategy mainly influenced by the Q-value and the exploration bonus, it will gradually learn the state-transition model. After the robot reaches a goal state, the shortest paths reaching this goal state from all states can be found using the SSP method with an integrated cost function as shown in (2) in the stochastic state-transition model.

In summary, action-selection in SSPQL is based on a linear combination of Q-value, SSP-value, and the value of exploration bonus, which is given by

$$\begin{aligned} a = \arg\max_{a' \in A}[&w_Q Q(s',a') + w_{SSP_t} Q_{SSP}(s',a') \\ &+ w_{EXP} Q_{EXP}(s',a') + \varepsilon]. \end{aligned} \tag{8}$$

Here, w_Q is a weight of Q-learning, w_{SSP_t} is a weight value of SSP , w_{EXP} is a weight of exploration bonus, and ε is a small random number for exploration.

$Q(s',a')$, $Q_{SSP}(s',a')$, and $Q_{EXP}(s',a')$ have different characteristics with respect to the learning performance. By using $Q(s',a')$, the convergence to optimality can be guaranteed. However, the learning speed when using only the Q-value is very slow. On the other hand, $Q_{SSP}(s',a')$ can affect a fast convergence speed. The SSP-finding method is a batch process, while Q-learning is an incremental process. Therefore, $Q_{SSP}(s',a')$ in the early stages of learning can have a meaningful value to reach the goal state. Finally, $Q_{EXP}(s',a')$ will lead an agent to unexplored states. If an agent visits a state, there will be a rapid decrease in Q_{EXP}. Thus, Q_{EXP} mainly affects the learning performance in the first episode.

The influence of the three action-value functions in (8) are as follows:

1) At the first trial for learning, Q_{EXP} mainly affects the number of steps, The number of steps in the first trial can be decreased by using Q_{EXP}.
2) During the early stages of learning, Q_{SSP} mainly affects the convergence speed. In these stages, Q and Q_{EXP} in (8) will be very small in value or will be zero.
3) In the latter stages of learning, Q would mainly affect the convergence to optimality. At this stage, Q_{SSP}, and Q_{EXP} in (8) approach to zero. For this, w_{SSP_t} is designed to have the property of time decay.

The overall algorithm of SSPQL is shown in Algorithm 2.

Algorithm 2: SSPQL

```
1: repeat
2:   repeat
3:       Observe a current state $s'$ and reward $r'$
4:       Update state-transition model {V,E, w, goal} by
         using $P(s'|s,a)$
5:       Choose action $a$ with
           $a = \arg\max_{a' \in A}[w_Q Q(s',a') + w_{SSP_t} Q_{SSP}(s',a')$
                    $+ w_{EXP} Q_{EXP}(s',a') + \varepsilon]$
```

```
6:      Execute $a$
7:      Update $Q$-value with
          $Q(s',a) := Q(s',a) + \alpha[r' + \gamma \max_{a' \in A} Q(s',a')$
                                                    $- Q(s',a)]$
8:      $s=s'$
9:    until a robot reaches a goal state
10:   Update SSP-value with
        $Q_{SSP} := SSP(V, E, w, goal)$
11: until learning ends
```

2.5. Convergence to optimality, and computational complexity of SSPQL

For a Q-learning method, convergence and optimality have been formally proven with appropriate parameters of MDPs [36] under the following conditions: bounded rewards $|r| \leq const$, learning rates $0 \leq \alpha \leq 1$, and decreasing learning rate over time.

Convergence to optimality in SSPQL can be given in the similarly to Q-learning. The action-value function of SSPQL converges to optimum when the additional condition is added to the convergence-condition of Q-learning. The additional condition is as follows:

$$0 \leq w_{SSP_t}(s,a) \leq 1, \lim_{t \to \infty} w_{SSP_t}(s,a) = 0, \quad \forall s, a. \tag{9}$$

For instance, $w_{SSPt}(s,a)=[N(s,a)]^{-1}$ is a simple equation to satisfy the conditions specified in (9). Under the conditions, w_{SSPt} approaches 0 as t approaches infinity. Moreover, Q_{EXP} also converges to zero as t approaches infinity; the value of $N(s,a)$ become infinity as t goes to infinity. Therefore, (8) will be simplified to be the same equation as found for Q-learning when t goes to infinity. As Q-learning has been proven, the convergence of SSPQL can be assumed.

Finally, the computational complexity of SSPQL can be calculated by considering the complexity of finding shortest paths and computing Q_{SSP} to reach a goal state. In the SSP-finding method, the most time-consuming process is Dijkstra's algorithm. This algorithm has a computational complexity given by $O(m+n\log n)$ with n nodes and m edges when the Fibonacci heap is used as a priority queue to implement the extract-min function [37]. Therefore, in big-O notation, the complexity of the SSP-finding method is $O(m+n\log n)$. For each episode, therefore, the additional computational complexity of SSPQL is $O(SA+S\log S)$, where S is the number of states and A is the number of available actions. Even so, the complexity of this method is less than many other model-based RL methods.

3. COMPARISON WITH OTHER MODEL-BASED REINFORCEMENT LEARNING METHODS

3.1. Computational complexity

As mentioned in Section 2.5, SSPQL has an additional computational complexity of $O(SA+S\log S)$ per episode, where S is the number of states and A is the number of available actions. The use of Dyna-Q or Prioritized Sweeping requires additional computation for each step. To compare the computational complexities of Dyna-Q with SSPQL, the average number of steps per episode should be defined.

If there is an efficient exploration method such as a counter-based exploration method, averaged steps per episode can be given as $O(S)$ [35]. Dyna-Q or prioritized sweeping requires k, the number of offline simulations per step. This means the additional computational complexity is proportional to kS when a counter-based exploration method is used. It is difficult to find the optimal value of k as this is dependent upon domain-specific characteristics such as the size of the state-space and the complexity of state-transitions. Although k must be chosen manually, it is known that the optimal value of k is proportional to SA. Therefore, the additional computational complexity in Dyna-Q or prioritized sweeping will be greater than $O(S^2A)$. From this, SSPQL is seen to be more effective in computational complexity than other model-based RL methods.

3.2. Convergence speed

If a learning method contains unnecessary computations at every step, the convergence speed of that method will be slow. Therefore, SSPQL can be compared to other model-based RL using a criterion corresponding to the number of unnecessary computations at every step. To compare SSPQL and other model-based RL methods in convergence speed, two factors will be considered: the optimal number of offline simulations and the cyclic state-transition.

For fast convergence speed in model-based RL methods such as Dyna-Q, the best number of offline simulation must be selected; however, this is difficult to specify because the ideal number might change according to the application domain. A large number of offline simulations increase computational overhead, while a small number decreases the convergence speed. In SSPQL, however, there is no need to select the number of offline simulations. SSPQL requires the minimum computations for fast convergence.

Next, cyclic state-transition badly affects the convergence speed in other model-based RL methods. A state-transition diagram with cycles is illustrated in Fig. 3 to explain why Q-values for Dyna-Q or Prioritized Sweeping hardly reach their optima under a non-deterministic cyclic state-transition. Cyclic state-transitions cause the Q-value to decrease and slows the convergence speed. For example, it can be observed from Fig. 3 that action a_1 at state s_i will make the transition to s_{i+1} with a probability of 0.3, and will remain in s_i with a probability of 0.7. In this case, action a_1 is an action to reach a goal. However, $Q(s_i, a_1)$ will

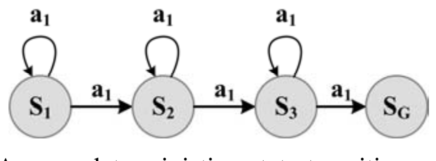
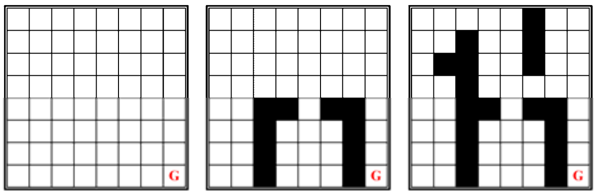


Fig. 3. A non-deterministic state-transition example with cycling.

not be updated whenever state changes are not made with a probability of 0.7. For example, the sequence $(s_i, a_1) \rightarrow (s_i, a_1) \rightarrow s_I$ may occur. In this case, the value of $Q(s_i, a_1)$ will be very slowly updated. Even in the case of a high decay rate, the Q-value may not converge to the optimum. Conversely, our proposed SSPQL will not have such difficulties, because Djikstra's algorithm removes cyclic state-transitions when finding the shortest path.

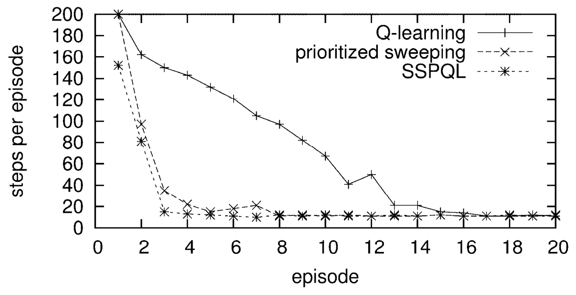
In order to compare the convergence speed of SSPQL with other RL methods, the grid world problem as shown in Fig. 4 was used. All state transitions in the grid world are stochastic, where 10% of actions lead to unexpected results. During all simulations, parameters for Q-function are chosen as γ=0.8 and α=0.8.

Firstly, an agent learns action-value functions without any prior knowledge of the environment shown in Fig. 4 through Q-learning, prioritized sweeping, and SSPQL, respectively. Whenever an agent reaches the goal state marked *G*, the next episode starts with a random initial state. Fig. 5(a) shows the average number of steps per episode without any knowledge of the environment illustrated in Fig. 4(a). From these results, SSPQL show a faster convergence rate than other RL methods. After action-value functions has been learnt, the agent learns optimal policies by using the previously learnt action-value functions in a different situation in Fig. 4(b). The convergence speeds for this example are shown in Fig. 5(b). Again, using action-value functions learnt in the situation shown in Fig. 4(b), the agent learns action value-functions in a different environment in Fig. 4(c). These results are shown in Fig. 5(c). We can observe from the results that prioritized sweeping has a slower convergence speed compared to SSPQL and Q-learning because the previously learnt internal state-transition model hinders the learning of action-value functions in new environments.

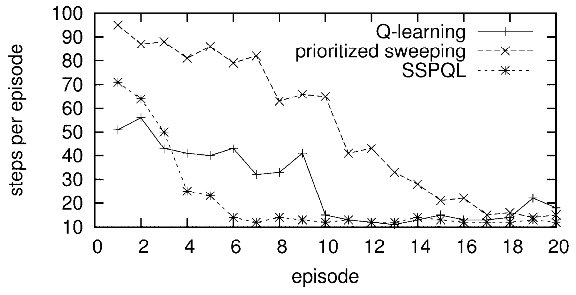


(a) 64 states. (b) 54 states. (c) 47 states.

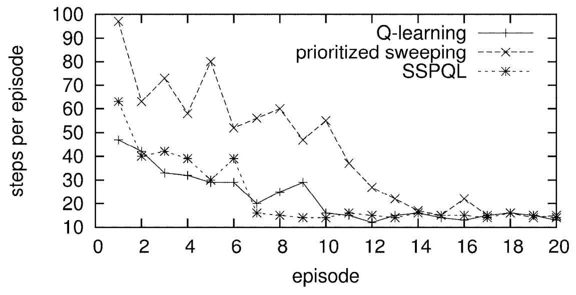
Fig. 4. Simulational environments of grid world.



(a) Learning of 64 states from scratch.



(b) Learning of 54 states using experience from Fig. 5(a).



(b) Learning of 47 states using experience from Fig. 5(b).

Fig. 5. Average number of steps per episodes in the simulation.

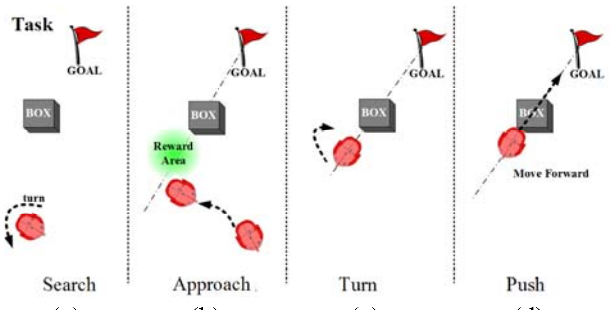
4. EXPERIMENTS

4.1. Task description

To validate the proposed reinforcement learning method, experiments were performed by giving a robot the task of pushing a box into a goal. To improve the effectiveness of the learning process, the task is broken down into four more detailed sub-tasks: search, approach, turn, and push. Firstly, the robot searches for the positions of both the box and goal using a camera. Secondly, the robot moves to the area in which it can receive a reward as depicted in Fig. 6(b). Thirdly, the robot aligns itself with the box and the goal to form a line as shown Fig. 6(c). Finally, the robot pushes the box into the goal. For the 'search' and 'push' subtasks, predetermined rules were designed and used, negating the need for learning. The sub-task 'approach' and 'turn' were learnt from scratch.

4.2. Experimental setup

To apply reinforcement learning in robotic tasks, state and action spaces must be quantized into a finite number



(a) (b) (c) (d)

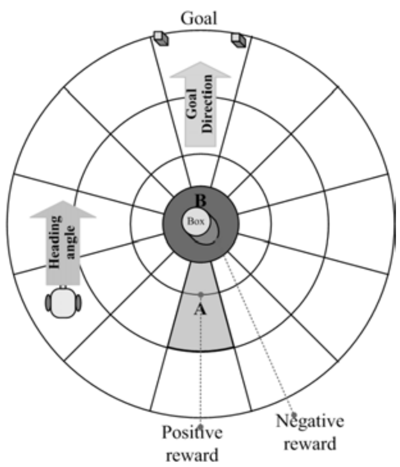
Fig. 6. A task description for pushing a box into a goal.

of cells. In many robotic applications, however, it is difficult to determine an appropriate quantization level to provide enough resolution of accuracy and low quantization error. If state and action spaces are fine-grained, uncertainty of state-transition will be small; however, the number of cells grows exponentially with the number of quantization levels. On the contrary, if space and action spaces are coarse-grained, there is a smaller number of cells; however, the uncertainty of state-transition is high. Although sensors and actuators of a robot are perfect, the robot must resolve uncertain state-transitions caused by the quantization of state and action spaces.

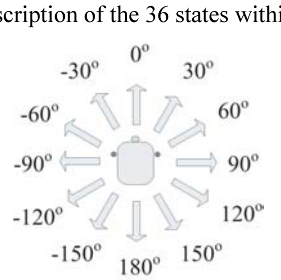
In this paper, state and action spaces are manually designed to reduce this quantization problem. If a state is defined such that object position with respect to the robot position in Cartesian coordinates is implied, state-transitions would be too complex to manage. Therefore, new coordinates were defined with the target object at the center. The transformed coordinates of the box, goal, and robot can then be given by

$$\begin{aligned} x_{goal_{new}} &= (x_{goal} - \Delta x)\cos(\Delta\theta) - (y_{goal} - \Delta y)\sin(\Delta\theta), \\ y_{goal_{new}} &= (x_{goal} - \Delta x)\sin(\Delta\theta) + (y_{goal} - \Delta y)\cos(\Delta\theta), \\ x_{robot_{new}} &= (-\Delta x)\cos(\Delta\theta) - (-\Delta y)\sin(\Delta\theta), \\ y_{robot_{new}} &= (-\Delta x)\sin(\Delta\theta) + (-\Delta y)\cos(\Delta\theta), \\ \theta_{robot_{new}} &= \Delta\theta, \end{aligned} \tag{10}$$

where $\Delta\theta$ = -arctan[$(y_{goal}$ - $y_{box})/(x_{goal}$ - $x_{box})$], Δx = $-x_{box}$, and Δy = $-y_{box}$. Here, x_{goal} and y_{goal} represent the x and y coordinates of the goal space, respectively. And, x_{box} and y_{box} denote the x and y coordinates of the box space, respectively.



(a) Description of the 36 states within $\boldsymbol{S}_{pos}$.



(b) Description of the 12 states within $\boldsymbol{S}_{heading}$.

$$\mathbf{S} \Leftarrow \mathbf{S}_{pos} \times \mathbf{S}_{heading}$$

Fig. 7. Definition of state space.

S_{pos} is a space of position-dependent states. S_{pos} is designed to have 36 different states determined by 12 robot orientations as well as 3 distances from the box to the goal as shown in Fig. 7(a). $S_{heading}$ is a state space containing the robot heading angle, and is designed here to have 12 states as shown in Fig. 7(b). The value of $x_{robotnew}$ and $y_{robotnew}$ are used to determine the state in S_{pos}, and $\theta_{robotnew}$ is used to determine the state in $S_{heading}$. The whole state space is defined as the product of S_{pos} and $S_{heading}$.

Next, four primitive actions are defined: 'move forward by 0.3m', 'move backward by 0.2m', 'turn 30º', and 'turn -25'. A reward signal is generated whenever the robot arrives at the state where S_{pos} is marked 'A' and $S_{heading}$=0 º as shown in Fig. 7(a).

4.3. Experimental results

It is difficult to conduct experiments to verify reinforcement learning in various situations using a real robot due to a time-consuming exploration process. Thus, reinforcement learning tasks were simulated in a virtual environment to find the best parameters. Using the best parameters, experiments for a real robot to verify the SSPQL algorithm. Firstly, to simulate interactions between the working environment and a real robot with the previously defined sensors and a drive system, open source simulator, Player and Stage [38] was used.

The mobile robot has a differential drive locomotion, a ring of 16 sonar sensors, a single CCD camera to determine the directions of objects, and a scanning range finder to measure distances to objects and walls. Higher level features such as the relative positions of the box and goal object are computed from raw distance information and color segmentation of the camera image. As the relative positions of both the box and goal from the robot are unknown when the learning trial is initiated, the robot must acquire these positions using the camera and scanning range finder.

During all experiments, parameters for the Q-function are chosen as γ=0.8 and α=0.8. Figs. 8 to 11 show the average learning curves of the 'approach' sub-task for various weight values for w_q, w_{SSP}, and w_{EXP}. Fig. 12 shows an average learning curve for the 'turn' sub-task.

Fig. 8 shows the simulational learning performances for the case when weight values were specified as w_q=1, w_{SSPt}=1, and w_{EXP}=1. These parameters correspond to action selection using only Q-values. In contrast, when the weight values were chosen as w_q=0, w_{SSP}=1, and w_{EXP}=0, simulational learning performance is shown in Fig. 9. This case corresponds to action selection using only SSP-values. It is observed from Figs. 8 and 9 that the number of steps per episode in Fig. 9 converges after the 5th episode, while the number of steps per episode in Fig. 8 converges after the 15th episode. Thus, the learning performance when only SSP-values were used for the action-value function is better than when only Q-

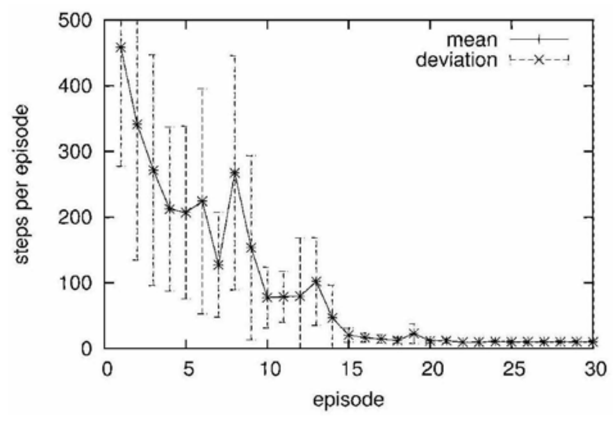


Fig. 8. SSPQL learning performance for the 'approach' sub-task, where w_q=1, w_{SSP}=0, and w_{EXP}=0.

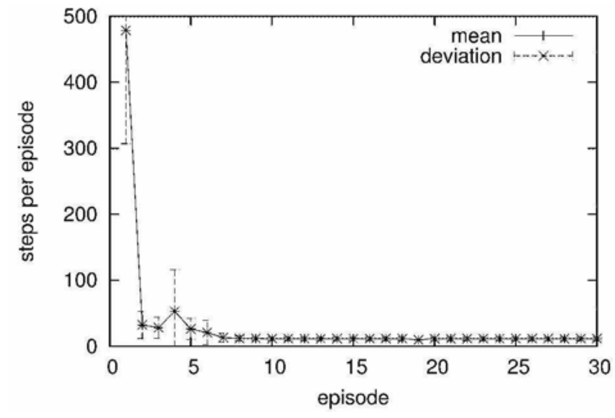


Fig. 9. SSPQL learning performance for the 'approach' sub-task, where w_q=0, w_{SSP}=1, and w_{EXP}=0.

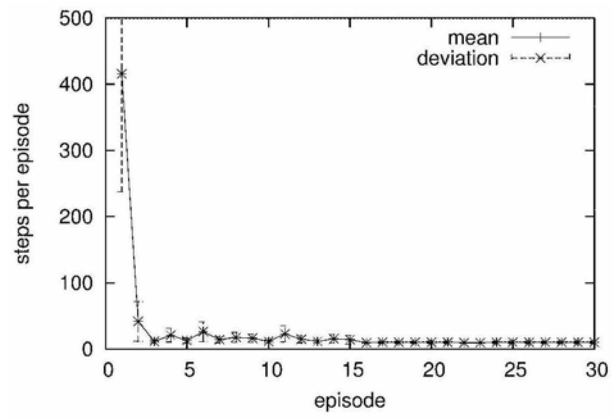


Fig. 10. SSPQL learning performance for the 'approach' sub-task, where w_q=1, w_{SSP}=1, and w_{EXP}=0.

values were used.

Fig. 10 shows the averaged learning performances for the system with weight values given as w_q=1, w_{SSP}=1, and w_{EXP}=0. The average number of steps per episode for the case using only SSP-value as in Fig. 9 is less than when both Q-value and SSP-values were used as shown in Fig. 10. However, the standard deviation of the number of steps for the SSP only case shown in Fig. 9 is observed to be larger than that found in Fig. 10. From

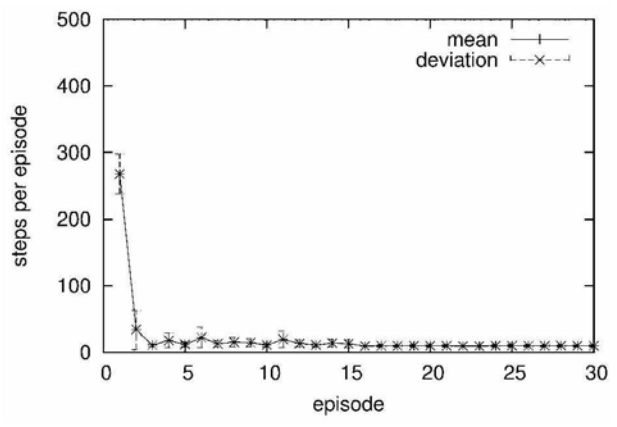


Fig. 11. SSPQL learning performance for the 'approach' sub-task, where w_q=1, w_{SSP}=1, and w_{EXP}=0.1.

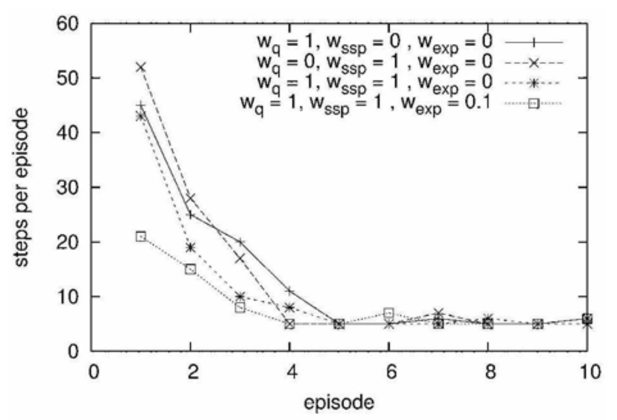


Fig. 12. SSPQL learning performances for 'turn' sub-task.

Fig. 9, action-selection using only SSP-values may occasionally not follow an optimal path. When a state-transition model does not include a global optimal path, action selection using only SSP-value would be sub-optimal. Consequently, the combination of Q-value and SSP-value methods enhances optimality.

Fig. 11 shows the learning performance of SSPQL when employing an exploration bonus, where w_q=1, w_{SSP}=1, and w_{EXP}=0.1. When compared to SSPQL's learning performance without an exploration bonus in Fig. 10, the number of steps during the first episode shown in Fig. 11 is observed to be nearly half. By using exploration bonus values, the number of trials during the first episode can be drastically decreased. Finally, the learning performances for the 'turn' sub-task is shown in Fig. 12. In this task, the size of the state-space is smaller than that of the 'approach' sub-task. Moreover, available actions are restricted to two: left turn and right turn. As the size of the state space become smaller, changes in learning performances for various weight values for w_Q, w_{SSP} and w_{EXP} reduce accordingly.

Table 1 shows the total number of steps from the 1st episode to the 30th episode for the Q-learning, SSP, SSPQL, and SSPQL with exploration bonus, respectively. From the result, it can be shown that the learning method by using both SSPQL and exploration bonus is most efficient in convergence speed.

Table 1. The total sum of steps from the 1st episode to the 30th episode.

Learning methods	The number of steps
Q-learning	2849
SSP	920
SSPQL	822
SSPQL with exploration bonus	638

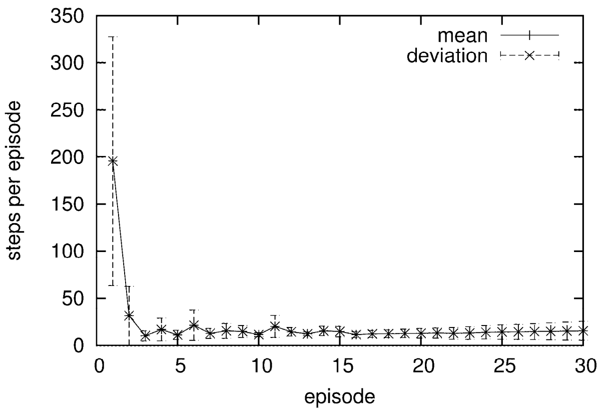


Fig. 13. SSPQL learning performance for the 'approach' sub-task in a real robot, where w_q=1, w_{SSP} =1, and w_{EXP} =0.1.

To verify the SSPQL algorithm in a real-life environment, a Pioneer mobile robot was used with the robot holding the same specifications as in the simulation.

Experimental results for the 'approach' sub-task are shown in Fig. 13, where the number of steps per episode is obtained by averaging the number of steps of five starting from two different initial locations. Parameters for the Q-value update equation in Fig. 1 are given as γ=0.8 and α=0.8. Weight values are given as w_q=1, w_{SSP}=1, and w_{EXP}=0.1. When compared with the simulation results in Fig. 11, the experimental results in Fig. 13 show that (1) the number of steps per episode decreases, especially for early episodes (from the 1st to 5th episodes), and (2) the deviation of number of the steps per episode also increased.

The difference in the experimental results between the real robot experiment and simulation can be explained by the uncertainty of actuators and sensors. The larger the odometry error is, the greater the deviation of the learning performance per episode is. Due to the uncertainty of sensors and actuators, there is no unique shortest path in the real environment. Therefore, deviation in the real experiments becomes larger than the simulation results. Nevertheless, from the experimental results, the SSPQL method was successfully applied to a practical learning task without any drastic deterioration of learning performance.

5. CONCLUSIONS

In this paper, a Stochastic Shortest Path-based Q-learning (SSPQL) method was proposed, integrating the Q-learning method with a stochastic shortest path finding algorithm. By combining the two learning methods, both learning speed and adaptability showed improvement when compared to previous RL methods. Experimental results show that the convergence speed of SSPQL exceeds both Q-learning and the prioritized sweeping method. Moreover, the result also shows that SSPQL has good performance of adaptability in dynamic environments. Thus, SSPQL would be suitable as a learning method for an autonomous robot in the real-life environments, where performing many explorations is not possible.

REFERENCES

[1] C. M. Witkowski, *Schemes for Learning and Behaviour: A New Expectancy Model*, Ph.D. dissertation, University of London, 1997.

[2] S. Lee, I. Suh, and W. Kwon, "A motivation-based action-selection-mechanism involving reinforcement learning," *International Journal of Control, Automation, and Systems*, vol. 6, no. 6, pp. 904-914, 2008.

[3] W. Smart and L. Kaelbling, "Effective reinforcement learning for mobile robots," *Proc. of the IEEE International Conference on Robotics and Automation*, May 2002.

[4] J. Peters, S. Vijayakumar, and S. Schaal, "Reinforcement learning for humanoid robotics," *Proc. of 3rd IEEE-RAS International Conference on Humanoid Robots*, 2003.

[5] S. K. Chalup, C. L. Murch, and M. J. Quinlan, "Machine learning with aibo robots in the four-legged league of robocup," *IEEE Trans. on Systems, Man and Cybernetics, Part C: Applications and Reviews*, vol. 37, no. 3, pp. 297-310, May 2007.

[6] D. H. Grollman and O. C. Jenkins, "Learning robot soccer skills from demonstration," *Proc. IEEE 6th International Conference on Development and Learning*, pp. 276-281, 11-13 July 2007.

[7] W. Yang and N. Chong, "Imitation learning of humanoid locomotion using the direction of landing foot," *International Journal of Control, Automation and Systems*, vol. 7, no. 4, pp. 585-597, 2009.

[8] Q. Jiang, H. Xi, and B. Yin, "Dynamic file grouping for load balancing in streaming media clustered server systems," *International Journal of Control, Automation and Systems*, vol. 7, no. 4, pp. 630-637, 2009.

[9] A. Barto, S. Bradtke, and S. Singh, "Learning to act using real-time dynamic programming," *Artificial Intelligence*, vol. 72, no. 1-2, pp. 81-138, 1995.

[10] R. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9-44, 1988.

[11] C. Watkins, *Learning from Delayed Rewards*, Ph.D. dissertation, University of Cambridge, 1989.

[12] L. P. Kaelbling, M. L. Littman, and A. P. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237-285, 1996.

[13] G. Konidaris and G. Hayes, "An architecture for behavior-based reinforcement learning," *Adaptive*

Behavior, vol. 13, no. 1, pp. 5-32, 2005.
[14] I. H. Suh, I. H. Suh, S. Lee, W. Y. Kwon, and Y.-J. Cho, "Learning of action patterns and reactive behavior plans via a novel two-layered ethology-based action selection mechanism," *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1799-1805, 2005.
[15] I. H. Suh, I. H. Suh, S. Lee, B. O. Kim, B. J. Yi, and S. R. Oh, "Design and implementation of a behavior-based control and learning architecture for mobile robots," *Proc. IEEE International Conference on Robotics and Automation*, vol. 3, pp. 4142-4147, 2003.
[16] A. Kleiner, M. Dietl, and B. Nebel, "Towards a Life-Long Learning Soccer Agent," *Proc. International RoboCup Symposium*, Fukuoka, Japan, pp. 119-127, 2002.
[17] M. Wiering and J. Schmidhuber, "Fast online Q(l)," *Machine Learning*, vol. 33, no. 1, pp. 105-115, 1998.
[18] J. Schmidhuber, "Exploring the predictable," *Advances in Evolutionary Computing*, A. Ghosh and S. Tsuitsui, Eds., Kluwer, 2002.
[19] B. Bakker, B. Bakker, V. Zhumatiy, G. Gruener, and J. Schmidhuber, "Quasi-online reinforcement learning for robots," *Proc. IEEE International Conference on Robotics and Automation*, V. Zhumatiy, Ed., pp. 2997-3002, 2006.
[20] J. Morimoto and C. Atkeson, "Learning biped locomotion," *IEEE Robotics and Automation Magazine*, vol. 14, no. 2, pp. 41-51, June 2007.
[21] R. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," *Proc. of the 7th International Conference on Machine Learning*, 1990.
[22] O. Buffet and D. Aberdeen, "Robust planning with (L)RTDP," *Proc. of the 19th International Joint Conference on Artificial Intelligence*, 2005.
[23] P. Plamondon, B. Chaib-draa, and A. Benaskeur, "A Q-decomposition and bounded RTDP approach to resource allocation," *Proc. of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, ACM, p. 200, 2007.
[24] A. Strehl, L. Li, E. Wiewiora, J. Langford, and M. Littman, "PAC model-free reinforcement learning," *Proc. of the 23rd International Conference on Machine Learning*, ACM, p. 888, 2006.
[25] R. E. Korf, "Real-time heuristic search," *Artificial Intelligence*, vol. 42, no. 2-3, pp. 189-211, 1990.
[26] S. Babvey, O. Momtahan, and M. Meybodi, "Multi mobile robot navigation using distributed value function reinforcement learning," *Proc. IEEE International Conference on Robotics and Automation*, vol. 1, 14-19, pp. 957-962, September 2003.
[27] W. Zhu and S. Levinson, "Vision-based reinforcement learning for robot navigation," *Proc. International Joint Conference on Neural Networks*, vol. 2, 15-19, pp. 1025-1030, July 2001.
[28] M. A. Wiering, R. P. Salustowicz, and J. Schmidhuber, "Model-based reinforcement learning for evolving soccer strategies," *Proc. of Computational Intelligence in Games*, Vienna, Austria, Austria: Physica Verlag Rudolf Liebing KG, pp. 99-131, 2001.
[29] T. Nishi, Y. Takahashi, and M. Asada, "Incremental behavior acquisition based on reliability of observed behavior recognition," *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 70-75, Oct. 29 2007-Nov. 2 2007.
[30] J. Morimoto, J. Nakanishi, G. Endo, G. Cheng, C. Atkeson, and G. Zeglin, "Poincare-map-based reinforcement learning for biped walking," *Proc. of the IEEE International Conference on Robotics and Automation*, April 2005.
[31] M. Ogino, Y. Katoh, M. Aono, M. Asada, and K. Hosoda, "Vision-based reinforcement learning for humanoid behavior generation with rhythmic walking parameters," *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, 27-31, pp. 1665-1671, October 2003.
[32] W. Y. Kwon, S. Lee, and I. H. Suh, "A reinforcement learning approach involving a shortest path finding algorithm," *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, 27-31, pp. 436-441, October 2003.
[33] W. Kwon, I. H. Suh, S. Lee, and Y.-J. Cho, "Fast reinforcement learning using stochastic shortest paths for a mobile robot," *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 82-87, Oct. 29 2007-Nov. 2 2007.
[34] T. Cormen, *Introduction to Algorithms*, The MIT Press, 2001.
[35] S. B. Thrun, "Efficient exploration in reinforcement learning," *Tech. Rep. CMU-CS-92-102*, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1992.
[36] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279-292, 1992.
[37] R. Neapolitan, *Foundation of Algorithms: Using C++ Pseudocode*, Jones and Bartlett Publishers, 1998.
[38] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: tools for multi-robot and distributed sensor systems," *Proc. of International Conference on Advanced Robotics*, pp. 317-323, 2003.

Woo Young Kwon received his B.S. degree in Mechanical Engineering at Hanyang University in 2001. He received his M.S. degree from the Department of Information and Communication at the same University in 2003. Currently, he is a Ph.D. student in the Department of Electronic and Computer Engineering at Hanyang University. His research interests include artificial intelligence, reinforcement learning, probabilistic modeling and inference.

Il Hong Suh received his Ph.D. degree in Electrical Engineering from the Korea Institute of Science and Technology (KAIST), Seoul, in 1982. From 1985 to 2000, he was with the Department of EECS, Ansan Campus, Hanyang University, where he was a full professor. From 2000 to the present, he has been with Department of Computer Science and Engineering, Hanyang University, Seoul, Korea, where he is a full professor. He has served as the President of the Systems and Control Society of Korea Institute of Telecommunications Engineers (2005-2007), and president of the Korea Robotics Society in 2008. He has also served as Editor-in-chief for Journal of Intelligent Service Robotics, Springer, and an Associate Editor for IEEE Transaction on Robotics. His research interests lie in the area of control and intelligence for robots including visual perception, ontology-based robot intelligence, learning, and intelligence architecture. He has published more than 170 contributions on robotics and control.

Sanghoon Lee received his B.S. degree from the Department of Mathematics, and his M.S. and Ph.D. degrees in Electronics, Electrical, Control and Instrumentation Engineering from Hanyang University, Korea, in 1994, 1997 and 2006, respectively. Currently, he is a Research Fellow at the Education Center for Network-based Intelligence Robotics, Hanyang University. His research interests include ethology-based action selection architecture, robot vision, and ontology-based robot intelligence.