Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

Behavioral-Fusion Control Based on Reinforcement Learning

Kao-Shing Hwang Yu-Jen Chen Chun-Ju Wu Cheng-Shong Wu
National Chung Cheng University
Electrical Engineering
Chia-Yi, Taiwan
hwang@ccu.edu.tw yujen@mail.educities.edu.tw 490350260@s90.tku.edu.tw cswu@ee.ccu.edu.tw

***Abstract*—To design appropriate actions of mobile robots, the designers usually observe the sensory signals on the robots and decide the actions from the viewpoint of some desired purposes. This approach needs deliberative consideration and abundant knowledge on robotics for a variety of situations. To improve the actions of robots, it is hard to sense the error by human eyes and takes time in trial-and-error. In this article, we propose a novel learning algorithm, Fused Behavior Q-Learning algorithm (FBQL) to deal with such situations. The proposed algorithm has the merit of simplicity in designing individual behavior by means of a decision tree approach to state aggregation which is eventually recoding the domain knowledge. Furthermore, these learned behaviors are fused into a more complicated behavior by a set of appropriate weighting parameters through a Q-learning mechanism such that the robots can behave adaptively and optimally in a dynamic environment.**

***Keywords*—reinforcement learning, behavior-based control, multiple behaviors, decision tree induction**

I. INTRODUCTION

In general, designing appropriate actions to control robots is more difficult because of the system complexity. The considered parameters and conditions are enormous, and there are some tradeoffs between these actions. Sometimes, experiences and intuitions do not help judge the tradeoff, especially in complex conditions. Subsumption method is proposed to overcome the difficulty [1-3]. It develops some basic behavior module to achieve the task goal. In general conditions, only one behavior is activating by suppressing other behaviors so that such kind of algorithms raise some other problems. For example, it must decide what behavior suppresses others appropriately in every condition. Fused Behavior Q-Learning algorithm (FBQL) proposed in this article can overcome the problem. Like Subsumption method, it also develops some basic behavior modules and it utilizes a reinforcement learning (RL) to fuse the outputs of all behavior modules[4,5].

However, even a simple basic behavior also suffers some problems. Since sensory inputs are usually real values, it is difficult to define a clear range to represent an output. If it is an easy task, it may use fuzzy methods. But in the perplexity tasks, if there is no knowledge about the inference rules, it is difficult to create a suitable output. In general, although we do not know how to create an output, we can easily judge whether the result is better or worse. And further, if it is a Markov Decision Process (MDP), it can utilize RL to learn the mapping from the sensory inputs to the outputs [6,7].

For general RL, it also suffers the problem about partitioning state space suitably. If it partitions the state space coarsely, it results in worse performance. On the contrary, although fine partitioning results in better performance, it needs more and more time to learn. Several methods have been proposed to overcome the problem. Neural network can utilize less neurons to predict the state values, but it has a drawback in convergence. The Cerebellar Model Articulation Controller (CMAC) ignores the dissimilarity between different regions of the state space[8]. In other words, some regions do not need fine partitions, but some regions need fine partitions for sensitive control. Although decision tree considers the dissimilarity, the drawback is that it spends more time in learning [9-10].

We proposed Reinforcement Learning-based Decision Tree algorithm (RL-based Decision Tree algorithm) to ease the design complexity [11]. The instructor only needs to demonstrate the behavior a few times and the robots can imitate the instructor's experience and induce these combinations of the sensory inputs and outputs. To improve the behavior, we proposed Decision Tree-based Q-Learning (DT-based Q-Learning) to train and adjust the behavior.

Section II describes the RL-based Decision Tree to clone single behavior. The details of the FBQL algorithm including whole structure, meaning of each parameter, and the processes of the algorithm are also presented in this section. In section III, we experiment with FBQL and Subsumption method to fuse three single behaviors and discuss the results. Finally, conclusions are presented in Section IV.

II. FUSED BEHAVIOR Q-LEARNING ALGORITHM

As the Subsumption method, we should develop some basic behavior before constructing our system. Humans usually learn new behaviors through imitation and then practices over and over times. This idea inspired us to propose the RL-based Decision Tree [11]. First, the instructor controls the robot from his/her instincts or experience, and the robot should record the patterns constructed from the sensory inputs denoted as

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

attributes and the robot's outputs denoted as classes. Second, the robot classifies all the patterns according to the attributes by the RL-based Decision Tree algorithm. After that, the robot can imitate the behavior as demonstrated by the instructor. To improve the behavior of the robot, we proposed Decision Tree-based Q-Learning (DT-based Q-Learning) [11].

A. RL-based Decision Tree

In general, the decision tree induction algorithms are based on a top-down greedy strategy to make a decision at each node, but it implies that the algorithms may fall into a local minimum [12,13]. In other words, the optimal decision tree must take account of successors while making a decision. In this work, the process of decision tree induction is regarded as an MDP, where the decision specifies a most appropriate split which has maximum long-term (accumulated) payoff in building the decision tree.

1) Algorithm

The whole state space of RL corresponds to the root node of the tree. Each unique state corresponds to a unique leaf node of the tree. Each state is represented by a vector, representing the ranges of sensory input in this state. The representation of the state corresponding to node ϕ is defined as $s(\phi) \equiv (x_1^{\min}, x_1^{\max}, \ldots, x_{N_{\dim}}^{\min}, x_{N_{\dim}}^{\max})$, which is actually the vector consisting of the coordinates of the vertices of the hyper-cuboids where the sensory inputs reside in and $N_{\dim}$ is the dimensions of sensory space. The maximum and minimum input values of i-th dimension in the node are denoted as $x_i^{\max}$ and $x_i^{\min}$, respectively. The sensory inputs $\{\mathbf{x} \in R^{N_{\dim}} \mid x_i^{\min} \le x_i < x_i^{\max}, i = 1 \sim N_{\dim}\}$ belong to the state $s(\phi)$.

The action set $A(s)$ of the state s contains a no-split and half-split actions with every input dimensions. If the state $s(\phi)$ fits in with one of the following stop-splitting criteria, only the no-split action can be selected.

1. The range of state $s(\phi)$ is too small.
2. There are only few patterns in $s(\phi)$.
3. After taking a half-split action, either child node contains no data.
4. The impurity in node ϕ is good enough.

A node ϕ is viewed as a leaf node, when the selected action of the state $s(\phi)$ is a no-split action.

In general, the performance of a decision is evaluated by the error rate and tree size. Since the reward function leads the learning direction, the reward function contains two parts. The first part of the reward function is the information gain, $I_{gain}(\phi, a)$. The information gain $I_{gain}(\phi, a)$ can be described as

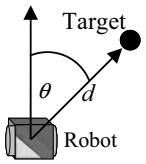

Fig. 1 The sensory inputs of the robot soccer

$$I_{gain}(\phi, a) = Entropy(\phi) - ((n_{\phi l} / n_{\phi}) Entropy(\phi_l) + (n_{\phi r} / n_{\phi}) Entropy(\phi_r)) \quad (1)$$

And the entropy function is defined as

$$Entropy(\phi) = -\sum_{i=1}^{C} p_i \ln(p_i), \quad (2)$$

where C is the number of classes and p_i is the probability (or percentage) of patterns belonged to class i in the node ϕ. If the chosen action gets larger information gain, the patterns of child nodes are more pure such that the error rate is small. n_{ϕ_l}, n_{ϕ_r}, and n_{ϕ} are the numbers of patterns in the left and right child nodes, ϕ_l and ϕ_r, of parent node ϕ, respectively[14]. However, without a restriction to the splitting process, the process of the decision tree induction may be too trivial and become over-fitting one. Therefore, a splitting cost, r_{split}, is introduced to the reward function as a small penalty on the efforts on split. The splitting cost is a negative constant so as to make the bias of the decision tree growing toward ceasing growing. The reward function is listed as

$$R(s(\phi), a) = \begin{cases} r_{split} + I_{gain}(\phi, a) & \text{if } a \text{ is a splitting action} \\ 0 & \text{if } a \text{ is a no splitting action} \end{cases}. \quad (3)$$

In general reinforcement learning, after taking an action, the

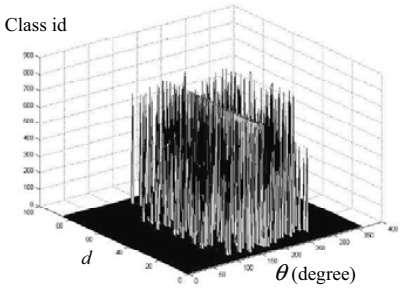

Fig. 2 The distribution of 1000 collected patterns

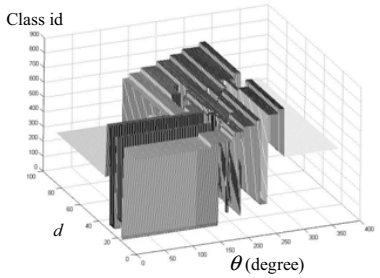

Fig. 3 The distribution of the patterns classified by RL-based DT

current state translates only to the next one state. But in the RL-based Decision Tree, after taking a splitting action, it has two child nodes as the next states. The updating rule is modified as

$$Q_{t+1}(s(\phi),a) = Q_t(s(\phi),a) + \alpha(R(s(\phi),a) + \gamma(\max_{b\in A(s)} Q_t(s(\phi_l),b) + \max_{b\in A(s)} Q_t(s(\phi_r),b))/2), \quad (4)$$

where ϕ_l and ϕ_r are the left and right child nodes of parent node ϕ. For demonstrating the performance, a simulation about the target seeking in soccer robot is done.

2) Simulation

The target approaching problem is performed with FIRA robot soccer simulator. At first, an instructor demonstrates the target approaching behavior for the robot, and it records the relationship between sensory inputs and the outputs at each time period. The sensory inputs have two parameters drawn in Fig. 1: one is the distance *d* between the target and the robot, and the other is the angle θ between the directions of the robot and the target

The outputs are the rotating velocities of the two wheels of the robot. A training pattern of the RL-based decision tree includes the sensory inputs as its attributes and the outputs as its class. Since the classes of patterns should be discrete but the outputs are continuous values, the velocities of the two wheels are quantized to 29 levels and encoded to 841 (that is 29*29) classes.

In the simulation, the agent records 1000 patterns demonstrated by an instructor in advance and the agent induce a RL-based decision tree by using these patterns. Fig. 2 shows the distribution of the 1000 collected patterns. After 50 trails, the RL-based decision converges and the number of leaf nodes is 110. The distribution of the patterns classified by the RL-based decision tree is shown in Fig. 3. After inducing the RL-based decision tree, the robot can imitate a similar behavior demonstrated by the instructor in advance. To improve the performance, the outputs are adjusted slightly at each leaf node.

B. FBQL Algorithm

In this section, a new algorithm called Fused Behavior Q-Learning (FBQL) algorithm is proposed to control a robot with some behaviors. The FBQL keeps the merit of simplicity in designing the individual behavior by RL-based decision tree and further considers the multiple behaviors at the same time on improving actions by responding to reinforcement signals. In the beginning, the robot imitates each behavior individually, and then fuses the learned behaviors by a set of appropriate weighting parameters. These parameters are learned through Q-learning such that the robot can behave adaptively and optimally in an unknown environment.

1) The Architecture of FBQL Algorithm

Depending on the many sensory inputs observed from the environment, the learning system can decide how to make fused behavior. Before performing the FBQL, all behaviors must use the RL-based DT method to learn and imitate the instructor's demonstrations individually. Since different behaviors induct different decision trees, the state space of FBQL must consider all the output combinations of all behaviors. In a behavior, the sensory inputs activate the corresponding leaf nodes. Each active leaf node of all behaviors represents one dimension of the state space of FBQL.

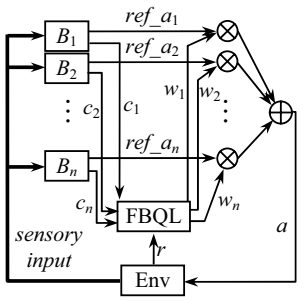

Fig. 4 The architecture of FBQL algorithm

The architecture of FBQL algorithm is shown in Fig. 4. *Bi* is the *i*-th behavior which is constructed by the RL-based DT. According to the structure of the associated decision tree with each behavior, the leaf node c_i where the sensory inputs fall onto is activated and outputs a reference action ref_a_i. The FBQL outputs a weighted action from each behavior. In the other words, the system sums up the all products of ref_a_i and w_i and outputs a fused behavioral action *a* to the environment. The environment will send a reward by the effect from this fused behavior *a*, and the FBQL can use this reward to adjust the system parameters.

When we receive a sensory input, each behavior *Bi* activates a leaf node c_i and outputs a reference action ref_a_i. These activated leaf nodes are combined to represent the input state, denoted as $s=(c_1,c_2,\ldots,c_n)$, of FBQL. In each state, FBQL maintains *n* weighting parameters as its outputs and and maintains *n* Q values corresponding to each weighting parameter. The Q value is updated by (5). The reward is divided according to each weight of the behavior. The more effective the behavior, the more responsibility it takes.

$$Q_{k+1}(s,B_i) = Q_k(s,B_i) + \alpha[w_k(s,B_i)r + \gamma\sum_{j=1}^{n} w_k(s',B_j)Q_k(s',B_j) - Q_k(s,B_i)]. \quad (5)$$

In the FBQL algorithm, the Q value represent the experiment after accumulating many times. The weight value means the actual output actions. Update the value by referring to the Q value as (6).

$$w_{k+1}(s,B_i) = w_k(s,B_i) + \begin{cases} (1-w_k(s,B_i))\delta(Q_{k+1}(s,B_i)+f), \text{if } B_i = \arg\max_{B_i} Q(s,B_i) \\ -w_k(s,B_i)\delta(Q_{k+1}(s,B_i)+f), \text{otherwize} \end{cases}. \quad (6)$$

The weight value increases if the behavior is better than others, and other weight values decrease at the same time. δ is a scale parameter, and *f* is an offset to ensure that the weight value is always positive. To limit the sum of the weight value, we normalize the value.

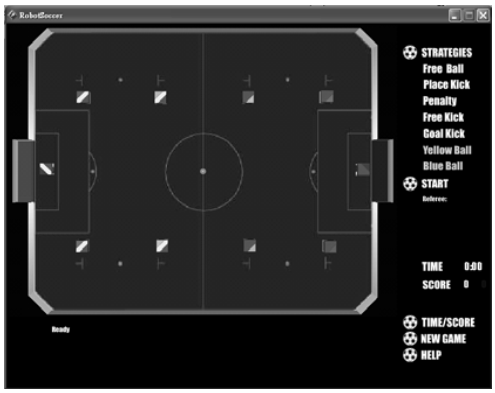

Fig. 5 FIRA SimuroSot Simulation

In the domain of reinforcement learning, there are two ways to design a reward. One is a dense reward, and the other one is a sparse reward. Dense reward is almost a reward function. This reward would hint the learning system to train toward some specific direction. The system can learn efficiently, but the reward function is very hard to design. The system would not achieve the goal if the reward function is not designed thoughtfully. The sparse reward is used in the FBQL algorithm. This reward is almost zero in the greater part of the states. In the other special state, the reward is non-zero. It is very easy for a designer to encourage or punish the robot. But the learning system need more time to learn the solution because it gets less information. The experience of learning by sparse reward is the real experience of the robot without the hint of the designer.

2) Convergency

Before briefly proofing the convergency of the FBQL, some symbols are predefined. $r_{\max}$ is the maximum value of the reward function for all states and behaviors. The $Q_{\max}$ is the maximum $Q^*(s,B)$ for all states and behaviors, where $Q^*(s,B)$ is the optimal q value of state s and action a. Hence

$$w(s,i) * r \le r_{\max} \tag{7}$$

$$\sum_{j=1}^{n} w(s', B_j) Q_k(s', B_j) \le \sum_{j=1}^{n} w(s', B_j) Q_{\max} = Q_{\max} \tag{8}$$

From (5), (7) and (8),

$$\begin{aligned} &\Delta Q_k(s, B_i) \\ &= [w_k(s, B_i) r + \gamma \sum_{j=1}^{n} w_k(s', B_j) Q_k(s', B_j) - Q_k(s, B_i)]^{\cdot} \\ &\le [r_{\max} + \gamma Q_{\max} - Q_k(s, B_i)] \end{aligned} \tag{9}$$

If the learning rate α has the famous constraints $\sum_{k=1}^{\infty} \alpha_k = \infty$ and $\sum_{k=1}^{\infty} \alpha_k^2 < \infty$, the q value can converge.

III. SIMULATION

To verify the FBQL algorithm, the FIRA simulation, used in the FIRA SimuroSot competition as Fig. 5, is performed. There are three single behaviors developed in advance based on the human experience. The three behaviors are described as follows:

1. Ball-Seeking Behavior: This behavior lets the robot to trace and get the ball as soon as possible. The sensory inputs are the distance from the ball to the robot and the angle from the direction of the robot to the direction of the ball, shown in Fig. 6.
2. Wall-Avoiding: This behavior can keep the robot safe and avoid getting stuck by the wall. We define the closest point of the wall as the position of the wall. Then the sensory inputs are the distance from the point of the wall to the robot and the angle from the direction of robot to the direction of the point, as shown in Fig. 7.
3. Approaching Behavior: This behavior is a tactic to push the ball to the front goal. It moves the robot toward the target along the approaching direction. Since the target is set to the front goal, the robot detours round the ball and then push the ball to the goal direction. The sensory input is the approach direction defined from the robot to the target, as shown in Fig. 8.

For each behavior, it uses the DT-based reinforcement learning described in section III to learn from the human experience. After inducting these decision trees to imitate human experience, the decision trees of Ball-Seeking Behavior, Wall-Avoiding Behavior, and Approaching Behavior have 122, 11, and 37 leaf nodes, respectively.

These single behaviors deal with different intentions. The

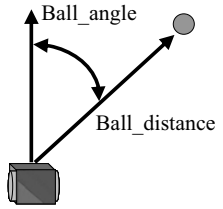

Fig. 6 The variables in Ball-Seeking Behavior

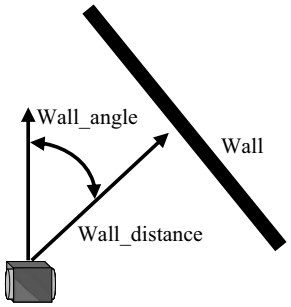

Fig. 7 The variables in Wall-Avoiding Behavior

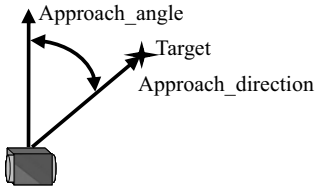

Fig. 8 The variables in Approaching Behavior

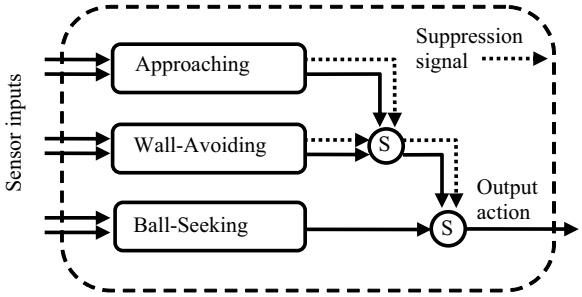

Fig. 9 The Subsumption method

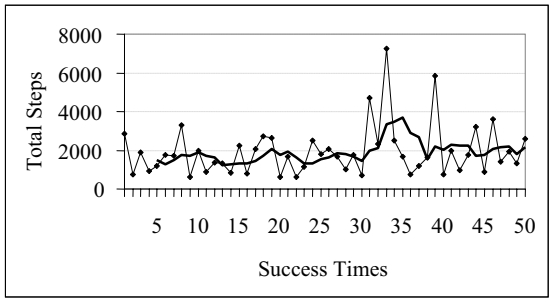

Fig. 10 The total steps of Subsumption method

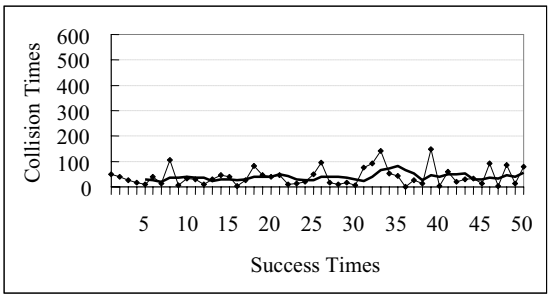

Fig. 11 The collision times of Subsumption method

target of the simulation is to utilize these behaviors to accomplish the offense strategy. The ball is set on the free-ball position and the robot is set on the front of the blue goal. It will quickly score a goal when we use the three behaviors wisely

A. Simulation with Subsumption method

We construct a Subsumption method to fuse these behaviors as Fig. 9. If more than one behavior is encouraged, we use Ⓢ suppression signal to keep harmony. The lowest level is the Ball-Seeking behavior to dribble the ball. The robot would use Wall-Avoiding behavior if it is near the wall. The Approaching behavior is high property to stand a good position to score a goal.

The robot kicks the ball until it scores a goal 50 times, and it fails if the ball is kicked to its gate. In the simulation, the robot fails 12 times in the process before succeeding 50 times. Observing the 50 success courses, we can see the total steps from the beginning to the goal as Fig. 10. The bold line is a moving average and the period is 5. The average steps are

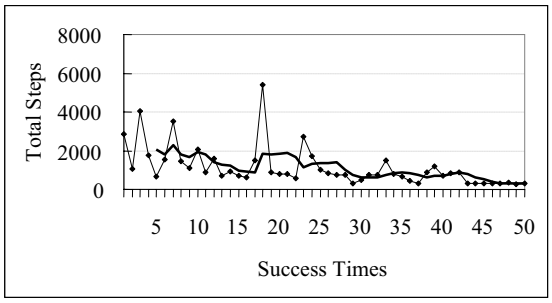

Fig. 12 The total steps of FBQL

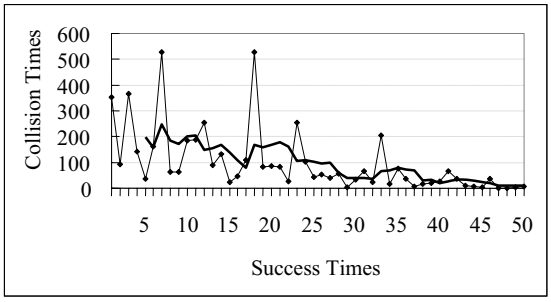

Fig. 13 The collision times of FBQL

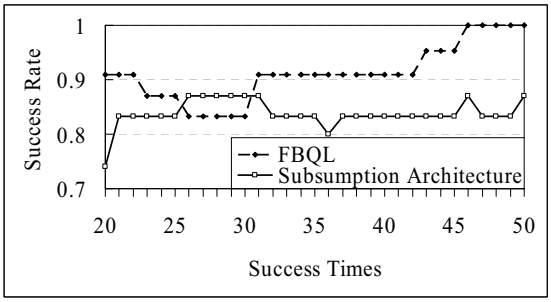

Fig. 14 Success rate comparison for FBQL and Subsumption method

between 1200 and 3700 steps. To verify the effect of the Wall-Avoiding behavior, the collision times of the simulation is shown as Fig. 11. The bold line is a moving average and the period is 5. The average collision times are stable between 21.2 and 80.8.

B. Experiment with FBQL algorithm

As described above, since these basic behaviors have 122, 11, and 37 leaf nodes, the FBQL has $122 \times 11 \times 37 = 49654$ states. Besides, the simulation uses sparse rewards. The rewards of success and failure are 10 and -10, respectively; the reward of collision is -5, otherwise the reward is 0. The large state space and sparse rewards result in endless learning time. To reduce the time of the initial learning stage, the Subsumption method is applied to guiding the FBQL in learning for 20 episodes first. The robot takes the action selected by the Subsumption method, and then according to the FBQL algorithm, it updates the q values and weights.

After 20 instructions, the robot selects the actions according

to the FBQL. At the same time, it continues updating the q values and the weights of FBQL. The robot also does the simulation which is the same as Subsumption method until it scores a goal 50 times. It only fails 4 times in this approach. Observing the 50 success courses, the total steps in each course is shown as Fig. 12. The bold line is a moving average with period 5.

The average steps are between 300 and 2300 steps, and the tendency is downward. Figure 13 also shows that the tendency of the collision times is downward.

Comparing these two approaches, the success rates of Subsumption method and FBQL are shown as Fig. 14. The success rate of FBQL increases with success times. Form Figs. 12-14, the performance gets better with success times; it means that the FBQL learns to fuse these basic behaviors to score a goal. The comparison of total steps also shows that the FBQL is better than Subsumption method, but the comparison of collision times show an opposite trend. Although the Subsumption method is better than FBQL in early episodes, the tendency in the FBQL is downward and better than Subsumption method. Some of the reasons why the FBQL is not good at collision times are:

1. The FBQL approach is interested in the long term reward.
2. In the Subsumption method, the Wall-Avoiding Behavior has priority. It is very hard to collide with the wall.

IV. CONCLUSIONS

In this article, we proposed a systematic method to design a behavior controller, especially in a complex task. We do not need to consider all conditions at the same time to design a complicated controller. We only decompose the complex task into some basic behaviors. The proposed FBQL can fuse these basic behaviors with different weights. But it has a drawback in state space size. The number of states increases with the number of behaviors. It means that it requires a more learning time. Besides, designing a density reward function is difficult. A sparse reward function is easy to be designed but results in a more learning time. These problems can be solved by demonstrating some episodes in advance. FBQL even learns better than the demonstrations in the final episodes.

For designing basic behaviors, we proposed the RL-based decision tree to imitate the experience of the expert who cannot clearly express his controlling rules. It is not like fuzzy system which needs "clear" inference rules. All he has to do is demonstrating how to control a basic behavior. Combining the FBQL and RL-based decision tree, we can easily design a controller with expert experience, and even with better performance.

REFERENCES

[1] R. C. Arkin, "*Behavior-Based Robotics*," The MIT Press, Cambridge, Massachusetts, London, England, pp. 130-141, 1998.

[2] D. Toal, C. Flanagan, C. Jones, and B. Strunz, "*Subsumption method for the Control of Robots*," University of Limerick, 1996.

[3] J. Simpson, C. L. Jacobsen, and M. C. Jadud, "Mobile Robot Control: The Subsumption method and Occam-Pi," *Communicating Process Architectures 2006*, Amsterdam, The Netherlands, pp. 225-236, 2006.

[4] R. S. Sutton and A. G. Barto, "*Reinforcement Learning*," The MIT Press, Cambridge, Massachusetts, London, England, 1998.

[5] N. J. Nilsson, "*Introduction to Machine Learning*," Robotics Laboratory Department of Computer Science Stanford University, pp.159-174, 1997.

[6] P. He and S. Jagannathan, "Reinforcement learning-based output feedback control of nonlinear systems with input constraints," IEEE Transactions on Systems Man and Cybernetics, Part B-Cybernetics, vol. 35, no. 1, pp. 150-154, 2005.

[7] C. J. C. H. Watkins, "*Learning from Delayed Rewards*," PhD thesis, Cambridge University, 1989.

[8] K. S. Hwang and C. S. Lin, "Smoothing Trajectory Tracking of Three-Link Robot: A Self-Organizing CMAC Approach," IEEE Transactions on System, Man, and Cybernetics, Part B, vol. 28, no. 5, pp. 680-692, Oct. 1998.

[9] S. R. Safavian and D. Landgrebe, "A Survey of Decision Tree Classifier Methodology," *IEEE Transactions on Systems, Man, and Cybernetics*, vol.21, pp.660-674, 1990.

[10] S. K. Murthy, "Automatic Construction of Decision Trees form Data: A Multi-Disciplinary Survey," *Data Mining and Knowledge Discovery*, vol.2, pp.345-389, 1998.

[11] K. S. Hwang, Y. J. Chen, and T. H. Yang, "Behavior Cloning by a Self-Organizing Decision Tree," in *Proceedings of the 2007 IEEE International Conference on Integration Technology*, Shenzhen, China, pp. 731-734, 2007.

[12] M. Dong and R. Kothari, "Look-Ahead Based Fuzzy Decision Tree Induction," *IEEE Transactions on Fuzzy System*, vol.9, 2001.

[13] S. K. Murthy and S. Salzberg, "Lookahead and Pathology in Decision Tree Induction," in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, San Mateo, CA, pp.1025-1031, 1995.

[14] L. Hyafil and R. L. Rivest, "Constructing Optimal Binary Decision Trees is NP-Complete," *Information Processing Letter*, vol.5, 1976.