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

Real-Valued Q-learning in Multi-agent Cooperation

Kao-Shing Hwang, *Senior Member, IEEE*, Chia-Yue Lo, and Kim-Joan Chen

***Abstract*—In this paper, we propose a Q-learning with continuous action policy and extend this algorithm to a multi-agent system. We examine this algorithm in a task that there are two robots taking action independently but connected with a straight bar. The robots must cooperate to move to the goal and avoid the obstacles in the environment. Conventional Q-learning needs a pre-defined and discrete state space but fails to identify the variances of the different situation in the same state. We introduce a Stochastic Recording Real-Valued unit to Q-learning to differentiate the actions corresponding to different state inputs but categorized to the same state. This unit can be regarded as an action evaluation module, which models and produces the expected evaluation signal and an action selection unit that generates an action with the expectation of better performance using a probability distribution function that estimates an optimal action selection policy. The results from both the simulation and experiment demonstrate better performance and applicability of the proposed learning model.**

I. Introduction

In the article we proposed a method called the Stochastic Recording Real-Valued (SRRV) learning algorithm to tackle the problems mentioned. Eventually, the problems that the proposed method coped with are two-fold; first, the work deals with the problem of quantization in the state space from sensory receptive field for the associated value function [1]. In other words, the architecture of the proposed algorithm can divided into two layers. The first layer is akin to a conventional Q-learning [2, 3] but with a scheme of state aggregation. On the second layer, the concept of stochastic action generation is applied to policy learning through exploration for Q-learning. The algorithm uses a Gaussian distribution to produce a stochastic and real-valued output, and adjusts the mean and the variance of the Gaussian distribution so as to increase the probability of producing the optimal real-valued output for each state. In other words, in conventional reinforcement learning [4,5] algorithms, such as Adaptive Heuristic Critics (AHC) [6], where the LMS rule of Widrow and Hoff approach is applied, the critic network may be trapped into local maximums. To overcome this problem, the SRRV learning algorithm is proposed to record the optimal reward and shows that it converges more quickly.

Furthermore, the capability of state aggregation is designed for the Q-learning with SRRV unit such that the system can differentiate actions corresponding to different state inputs but categorized (aggregated) to the same state since the conventional Q-learning needs a pre-defined and discrete state space but fails to identify the variances of the different situation in the same state. We denote this evolving Q-learning system as Real-Valued Q-learning (RVQ). As a matter of fact, the RVQ unit can be regarded as being composed of two functionality units; an action evaluation unit, which models and produces the expected evaluation signal, and an action selection unit that generates an action with the expectation of better performance based on a probability distribution function that estimates an optimal action selection policy.

The structure of this paper is organized as follows. In Section II, the learning algorithm of RVQ is derived based on the maximum likelihood. Section III describes the proposed algorithm of RVQ unit. The experimental results are demonstrated in Section IV, where the task is to assign two small-size succor robots holding two ends of a stick to learn to go through a narrow gate safely. Finally, a brief conclusion on the merit and drawback of the proposed algorithm is drawn in the last section.

II. Learning Agent in The RVQ Network

If a reinforcement learning scheme, instead of the supervisory learning scheme, is applied to this percetron-like unit, an action evaluation unit is supposed to be embedded into the model.

A. Action Selection with Stochastic Search

In the proposed learning algorithm, the gradient information is instead estimated by a stochastic exploration method, based particularly on the multi-parameter distributions used for the stochastic network output unit search [7]. In estimating the gradient information, the output y of the action network does not act directly on the environment. Instead, it is treated as a mean (expected) action. The actual action $\hat{y}$ is chosen by exploring a range around this mean point. This range of exploration corresponds to the variance in a probability function that is the normal distribution here. The amount of exploration $\sigma(t)$ is some nonnegative monotonically decreasing function of the predicted reinforcement signal. For example, $\sigma(t)$ can be interpreted as an extent to which the output node searches for a better action. Since $\hat{r}(t)$ is the predicted evaluation signal, if $\hat{r}(t)$ is small, the exploratory range $\sigma(t)$ will be large. On the contrary, if $\hat{r}(t)$ is large, $\sigma(t)$ will be small. This amounts to narrowing the search about the mean $y(t)$ if the predicted reinforcement signal is large. This can provide a higher probability of choosing an actual action $\hat{y}(t)$, which is very close to $y(t)$, since it is expected that the mean action $y(t)$ is very close to the best action possible for the current given input vector. On the other hand, the search range about the mean $y(t)$ is broadened if the predicted reinforcement signal is small such that the actual action has a higher probability of being quite different from the mean action $y(t)$. Thus, if an expected action has a smaller predicted reinforcement signal, we can have more novel trials.

In the above two-parameter distribution approach, the predicted reinforcement signal $\hat{r}(t)$ is necessary to determine the search range $\sigma(t)$. This predicted

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

reinforcement signal can be obtained from the action evaluation network that is described in the following subsection. Once the variance has been decided, the actual output of the stochastic node can be set as

$$\hat{y}(t) = \Psi(y(t), \sigma(t)). \tag{1}$$

That is, $\hat{y}(t)$ is a normal or Gaussian random variable with a density function:

$$\text{Prob}(\hat{y}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{\frac{-(\hat{y}-y)^2}{2\sigma^2}\right\}. \tag{2}$$

For a real-world application, $\hat{y}(t)$ should be properly scaled to the final output to fit the input specifications of the controlled plant [7]. In other words, the actual action is chosen by exploring a range around the expected action and this range of exploration corresponds to the variance of the normal distribution, σ determined by the action evaluation unit.

The interaction between the action unit and the environment is described as follows:

$$\mu(t) = \sum_{i=1}^{n} w_i(t) x_i(t) \tag{3}$$

At time t, the unit receives an input vector $\mathbf{x}(t)$. That unit uses the input vector $\mathbf{x}(t)$ and two internal parameter vectors $\mathbf{w}(t)$ and $\mathbf{v}(t)$ to compute the two parameters $\mu(t)$ and $\sigma(t)$ of the normal distribution to generate the unit's output. The mean output, $\mu(t)$, is an estimate of the optimal output which can maximize and compute a weighted sum of the inputs of the unit:

For a given input, the standard deviation $\sigma(t)$ should depend on how close the current expected output is to the optimal output for that input. The expected reinforcement $\hat{r}(t)$ is used to compute the standard deviation $\sigma(t)$ as:

$$\sigma(t) = s(\hat{r}(t)), \tag{4}$$

where $s(.)$ is a monotonically decreasing, nonnegative function of $\hat{r}(t)$. For instance, it can be designed as $s(\hat{r}(t)) = 1 - \hat{r}(t)$, so that the standard deviation becomes zero and causes the unit's output to be the optimal mean output when the maximum reinforcement is expected, i.e. $s(1.0) = 0.0$. Based on the mean, $\mu(t)$, and the deviation, $\sigma(t)$, of the normal distribution, $\Psi(\mu, \sigma)$, the unit calculates its activation, *a*(*t*), as

$$a(t) \sim \Psi(\mu(t), \sigma(t)), \tag{5}$$

where *a*(*t*) is the normally distributed random output generated by the network.

The unit uses above equations to calculate its output at a given time step. Assume that the environment provides a reinforcement signal *r(t)* that is the evaluation of the unit's output at time *t*. The weights computing the mean $\mu(t)$ are updated at each time step as follows:

$$w_i(t+1) = w_i(t) + \alpha \Delta w(t) x_i(t), \tag{6}$$

where α is the learning rate and

$$\Delta w(t) = (r(t) - \hat{r}(t))\left(\frac{a(t) - \mu(t)}{\sigma(t)}\right). \tag{7}$$

The fraction in (7) can be viewed as a normalized noise that has been added to the mean activation of the unit for the given input. If this perturbation has caused the unit to receive a reinforcement signal *r(t)* that is greater than the predicted evaluation $\hat{r}(t)$, then it is desirable for the unit to produce an output closer to the current output *a(t).* This should change the mean output value in the direction of the perturbation. That is, if the perturbation is positive, the unit should update its weights so that the mean value increases. Conversely, if the perturbation is negative, the weights should be updated so that the mean value decreases. On the other hand, if the reinforcement signal is less than the expected evaluation, then the unit should adjust its mean in the direction opposite to that of the perturbation.

Updating the weights $\mathbf{v}(t)$ used for computing the expected evaluation is relatively straightforward. The predicted evaluation $\hat{r}(t)$, generated by the action evaluation network, follows the reinforcement signal *r(t)*. Hence, *r(t)* can be viewed as a desired output and the delta rule is used to learn the association between the reinforcement signal and the expected evaluation. The update rule is given by:

$$v_i(t+1) = v_i(t) + \beta \Delta v(t) x_i(t), \tag{8}$$

where β is the learning rate and $\Delta v(t) = r(t) - \hat{r}(t)$.

B. *Action Evaluation with Maximum Likelihood*

As previously mentioned, an action evaluation unit is used to predict the reinforcement signal $r(t)$ from the environment, instead of modeling the discounted cumulative reinforcement [6]. Eventually the environment is usually a controlled plant, from where the observed reinforcement signals always come along with some kind of uncertainty due to the locality of the gradient search. In other words, the signal $r(t)$ may be regarded as a random signal better or worse around the true one. To accommodate the learning agent for this problem, an action evaluation unit based on the maximum likelihood method is developed.

Assuming that the reinforcement signal $r(t)$ can be modeled by $r = \overline{r} + g_n$, where g_n is an imposed Gaussian noise, the predicated evaluation signal $\hat{r}(t)$ can be represented as an estimated mean μ_r with the variance σ_r^2, i.e., $\hat{r}(t) = \mu_r$ (t) + norm $\cdot\, \sigma_r(t)^2$, where the norm is a normal distribution. The action evaluation unit computes the two parameters μ_r and σ_r^2 of a Gaussian distribution respectively. The final output of the action evaluation unit is selected randomly from this distribution. Not only should the action evaluation unit learn an output function that estimates the expected value μ_r of the conditional target distribution, but it should also have the capability to estimate the variance σ_r^2 of that distribution. In Short, the proposed architecture of the network with two output units; the unit, called y-unit, predicts the conditional mean μ_r of the output distribution. The output of the y-unit can be written as a weighted sum of the inputs:

$$\mu_r = \sum_{i=1}^{n} u_i x_i \tag{9}$$

The other unit, v-unit, predicts the conditional variance (i.e., estimate of σ_r^2) of that distribution. Since σ_r^2 must be positive, an exponential activation function is chosen for each v-unit to limit this boundary. Therefore the output of the v-unit can be written as:

$$\sigma_r^2 = \exp(\sum_{i=1}^{n} v_i x_i + v_{th}), \tag{10}$$

where v_{th} is the offset, and v_i is the weight between input unit x and the v-unit.

The y-unit predicts the conditional mean μ_r and v-unit predicts the conditional variance σ_r^2.

Assuming that the error model around *r(t)* is normal distribution, the probability distribution function can be written as :

$$P(r \mid x_i) = \frac{1}{\sqrt{2\pi\sigma_r^2(x_i)}} \exp\left\{ -\frac{[r-\mu_r(x_i)]^2}{2\sigma_r^2(x_i)} \right\} \tag{11}$$

The probability density function may be viewed as a likelihood function. Since the logarithm is a monotonic increasing function of its argument, a negative log-likelihood function is defined as follows:

$$-\ln P(r \mid x_i) = \frac{1}{2}\ln(2\pi\sigma_r^2(x_i)) + \frac{(r-\mu_r(x_i))^2}{2\sigma_r^2(x_i)} \tag{12}$$

$$-\frac{\partial \ln P}{\partial \mu_r} = -\frac{r-\mu_r}{\sigma_r^2}, \tag{13}$$

$$-\frac{\partial \ln P}{\partial \sigma_r} = -\frac{(r-\mu_r)^2 - \sigma_r^2}{\sigma_r^3} \tag{14}$$

The characteristic eligibilities of μ_r and σ_r are shown in (13), (14), respectively.

Let $\hat{r}$ be a normally distributed random variable. The update rule is obtained by setting the learning constant proportional to σ_r. Therefore, the update rules of the weights connected to the y-unit and v-unit are shown as follows, respectively,

$$\begin{aligned} u(t+1) &= u(t) + \eta \frac{1}{\sigma_r(t)}(r(t) - \mu_r(t)) x_i(t) \\ v(t+1) &= v(t) + \beta \frac{1}{\sigma_r^2(t)} \{[r(t) - \mu_r(t)]^2 - \sigma_r^2(t)\} x_i(t) \end{aligned}, \tag{15}$$

where η , β is a positive learning rate.

III. RVQ Learning Algorithm

In the Stochastic Real-Valued (SRV) [8] unit, the mean value $\mu(t)$ and the variance are approximated by recursive rules. Since it applies the LMS rule, the convergent results may fall into local solution even if it has ever reached the optimal solution. In order to overcome this drawback, the SRRV unit is presented. One unit $\mu(t)$ is to record the optimal solution up to now. Another unit $r_{\max}(t)$ is used to record the reinforcement signal gain at that optimal solution (i.e. the maximum $y(t)$ in this case). One last unit is a recursive unit $v(t)$. This unit determines the variance σ of the normal distribution $\Psi(\mu,\sigma)$.

Consider the problem of searching maximum solution of the function $y(x) = x\sin(1/\eta x)$ again. At time *t*, the action is generated as the equation $x(t) \sim \Psi(\mu(t),\sigma(t))$, where

$$\sigma(t) = 1 - \left(\frac{1}{1+e^{-v(t)}} \right). \tag{16}$$

Then we receive $y(t)$ and $r(t)$. The updating rules are separated into two phases:

Phase 1

If a better solution is obtained, the solution and $y(t)$ are recorded as $\mu(t+1)$ and $r_{\max}(t+1)$. And the variance decrease to explore wider. The updating rules are as follows:

if $r(t) > r_{\max}(t)$ then

$$\mu(t+1) = x(t)$$
$$r_{\max}(t+1) = r(t)$$
$$v(t+1) = v(t) + \alpha \mid r(t) - r_{\max}(t) \mid - \beta \mid x(t) - \mu(t) \mid$$

Phase 2

If a worse solution is obtained, the mean value $\mu(t)$ and $r_{\max}(t)$ will not be updated and $v(t)$ will increase.

The updating rule is as follow:

if $r(t) < r_{\max}(t)$ then

$$v(t+1) = v(t) + \alpha \mid r(t) - r_{\max}(t) \mid .$$

These two phases both converge because of the term $\alpha \mid r(t) - r_{\max}(t) \mid$. In phase 1, the divergence term $\beta \mid x(t) - \mu(t) \mid$ is added to explore wider when the learning agent discover a better solution.

The v -unit is the prediction of the variance (i.e. the estimation of σ) of the distribution. It contains two terms. One is the convergence term $\alpha \mid r(t) - r_{\max}(t) \mid$ which controls the convergence rate of the variance. α is the convergence rate. Since it is an absolute value, this term is a positive term. As the time increasing, the v -unit shall be updated larger, it will result in a smaller variance $\sigma(t)$. If the new explored reward $r(t)$ is much larger (or smaller) than the recorded reward $r_{\max}(t)$, it will speed up the convergence rate. The other term $-\beta\,|\,x(t)-\mu(t)\,|$ enlarges the variance, since it is a negative value. β is a divergence rate. If the new explored reward is larger than the recorded, it represents that we still can find another better solution, so we expand our exploration (i.e. increase the variance). So this term is added. The divergence rate is determined according to the location of the newly explored reward. If the location is far from the recorded mean value, it expands wider. Otherwise, it expands smaller.

A. Input Space Representation

The modified Fuzzy Cerebellar Model Articulation Controller (FCMAC) can be regarded as the fuzzification operation on fuzzy sets. The standard univariate basis function of the CMAC [9] is binary so that the network modeling capability is only piecewise constant. The univariate basis functions with higher-order piecewise polynomials can generate a smoother output. A crisp set can be considered as a special form of fuzzy set, to which an instance can to some degree either belong or not belong. This property is similar to the problem where the state variable x excites a specific region in the FCMAC sensor layer. The membership function, $\mu_i(x)\rightarrow[0,1]$, associates each state variable x with a number. It represents the grade of membership of x in $\mu_i(x)$. Triangles are adopted such that only one maximum output exists in the hyperspace after the aggregation operation. These membership grades have a peak value at the center of the excited region and decrease as the input moves toward the edge of the excited region. A two-dimensional RVQ operation where each subset is offset relative to the others along a hyperdiagonal in the input hyperspace is illustrated in Fig. 1.

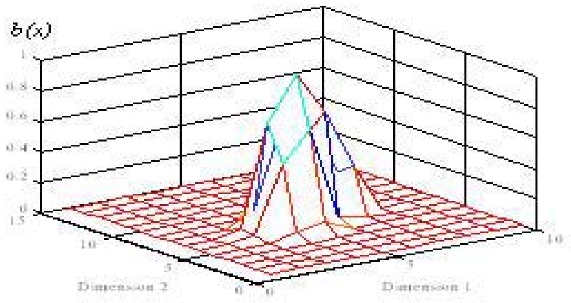


Fig.1. The nonlinear mapping result on the RVQ receptive field.

Fig. 1 shows a result that the RVQ maps the observed input space to the receptive field with various exciting strength in certain activated areas. In this way, different inputs can generate the outputs with discrepant representation in spite of the resolution by which the field is divided. From the other viewpoint, a FCMAC is actually a percetron-like action unit, which determines the output, connecting to a state space quantizer. The output of the action unit can be written as $p=\sum_i w_i b_i(x)$ where b_i is a fuzzy membership function.

B. RVQ Unit

In this section, a new RVQ model is proposed to produce a real-valued output. In this learning model, the stochastic recording real-valued unit is used to search for the action under a greedy policy in Q-learning. The proposed learning model is depicted as follows:

The robot exists in an environment consisting of a set, S of states. At time t, the observed input vector X, is sent into the Q-learning. The Q-learning receives this state $X(t)$, suppose we have an optimal policy $\pi^*(X)$ that maps input states into movements. The action space is separated into several sub-sets. The movement derived from Q-learning will correspond to these sets. The action generated from RVQ consists in one of these sets. Once the RVQ unit receives the action $a(t)$ under a policy, the action is also considered as an index to RVQ unit. Then a mean value under this policy $\mu(t)$ of the normal distribution was computed as follow:

$$\mu(t)=w(X,a,t), \tag{17}$$

where $w(X,a,t)$ is the recorded mean value.

The $v(X,a,t)$ value shall also be obtained. We use a temporary note $p(t)$ to represent $v(t)$ under the action. The variance $\sigma(t)$ of the normal distribution will then be obtained based on the $p(t)$ under the policy $\pi^*(X)$. Therefore we have

$$p(t)=v(X,a,t),$$

$$\sigma(t)=1-\frac{1}{1+e^{-p(t)}}. \tag{18}$$

Based on the mean value $\mu(t)$ and variance $\sigma(t)$, the action can be generated by the normal distribution

$$a_s(t)\sim\Psi(\mu(t),\sigma(t)). \tag{19}$$

Assume that the environment provides a reinforcement signal $r(t)$ that is the evaluation of the unit's output at time *t*. The state leads to next state $X'(t)$. The Q-learning is updated according to this reinforcement signal. The updating rule is as the original Q-learning:

$$Q_t(X,\pi)=(1-\alpha)Q_{t-1}(X,\pi)+\alpha[r(t)+\gamma V_{t-1}(X'_t)]. \tag{20}$$

When the agent receives the next state $X'(t)$, we can derive a weight vector ($b_1,b_2,\ldots,b_n$) from FCMAC. This weighting b_i factor represents how a continuous state $X'(t)$ is influenced by a representative state $X(t)$. When the agent perceives its environment as a continuous reward $r_s(t)$, where $r_s(t)$ is the reward of RVQ unit. This current $r_s(t)$ is described by the weighted linear combination of representative state b_i as follows:

$$r_s(t)=\sum_{i=1}^{n}V^{\pi^*}(X_i,t)b_i, \tag{21}$$

where $V^{\pi^*}(X_i,t)=\sum_a \pi(X_i,a)Q_t(X_i,a)$ is the optimal value function [5].

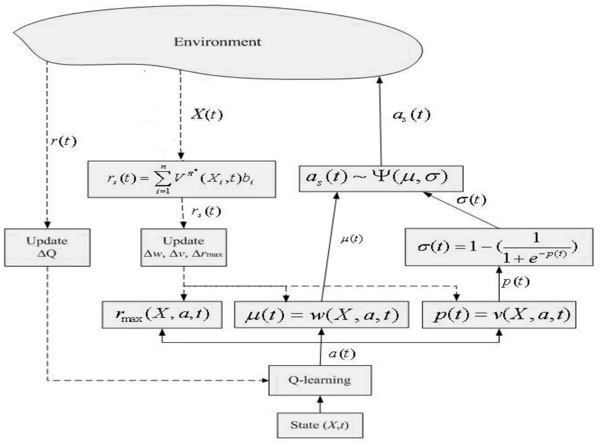

Environment
$X(t)$
$a_s(t)$
$r(t)$
$r_s(t)=\sum_{i=1}^{n} V^{\pi^*}(X_i,t)b_i$
$a_s(t) \sim \Psi(\mu,\sigma)$
$\sigma(t)$
$r_s(t)$
Update ΔQ
Update Δw, Δv, $\Delta r_{\max}$
$\mu(t)$
$\sigma(t)=1-(\frac{1}{1+e^{-p(t)}})$
$p(t)$
$r_{\max}(X,a,t)$
$\mu(t)=w(X,a,t)$
$p(t)=v(X,a,t)$
$a(t)$
Q-learning
State (X,t)

Fig. 2. The data flow diagram of the RVQ model.

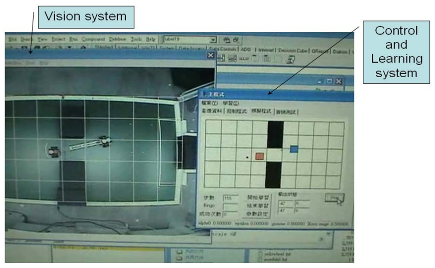


Fig. 3. The photo of the experimental environment where the task is to assign two small-size succor robots holding two ends of a stick to learn to go through a narrow gate safely.

This reward $r_s(t)$ determines which update rule should it use. The update rules are as follows:

$$\text{if } r_s(t) > r_{\max}(X,a,t) \tag{22}$$
$$r_{\max}(X,a,t+1) = r_s(t)$$
$$w(X,a,t+1) = a_s(t)$$
$$v(X,a,t+1) = v(X,a,t) + \alpha \cdot |r_s(t) - r_{\max}(X,a,t)| - \beta \cdot |a_s(t) - w(X,a,t)$$
$$\text{else } v(X,a,t+1) = v(X,a,t) + \alpha \cdot |r_s(t) - r_{\max}(X,a,t)|.$$

Fig. 2 shows the data flow diagram of the RVQ model. The signals used to calculate the output of the network are shown with solid lines, and the dash lines show the signals used for learning.

IV. Experiments

The proposed RVQ is implemented in two tasks to demonstrate its achievement. The first task is to assign a robot to avoid the obstacle and to reach to the goal. The other task is that there are two robots taking action independently and both are connected with a straight bar. Fig. 3 shows the experimental environment of this task.

A. The Environment of Experiments

The field is 900 in length and 500 in width. The action a_s generated is a direction command. The action a generated from Q-learning contains 4 commands which are UP, DOWN, LEFT and RIGHT. Each command maps to the direction up, down, left, right. The range of each direction's angle is from -45 to 45 degree. When the robot receives this command, it will move a distance toward this direction. The

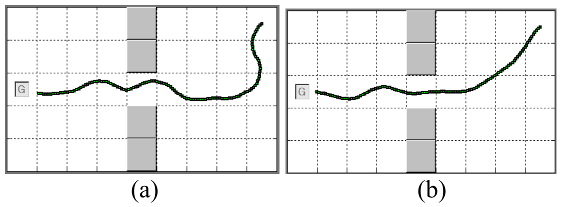


Fig. 4. (a) The path of the robot applying the SRV unit. The steps are 126 steps. (b) The path of the robot applying the RVQ unit. The steps are 105 steps.

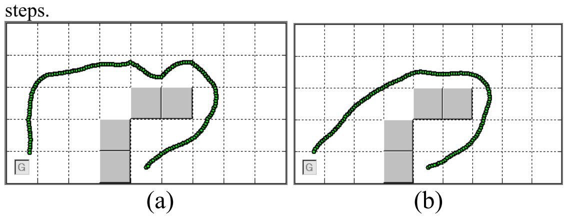


Fig. 5. (a) The path of the robot applying the SRV unit after 95 times learning cycles. (b) The path of the robot applying the RVQ unit after 95 times learning cycles.

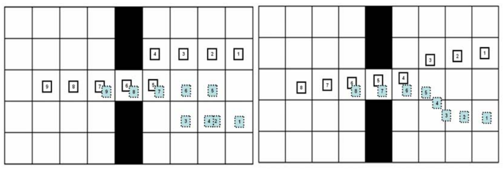
Q-table is a $9\times5\times4$ table, which means position x, position y, activity respectively. Each element in Q-table has corresponding μ, v and $r_{\max}$.

B. Results of Experiments

Fig. 4 (a) shows the path recorded at the 126th epoch is derived from SRV learning. For comparison, the path after 105 times training by the proposed RVQ is shown in Fig. 4 (b). It is notable that the later path is smoother and shorter through with less number of learning epoch. Fig. 5 shows the results of the other cases.

As the figure depicts, the path of the agent is rough and line-piece wide. It makes poor performance for robots to take actions in real applications and has worse performance. It is worthy to mention that when implementing in real robot applications, if the output action command can not be controlled precisely, the robots will transit to a state that may not be a well-defined quantized state. Because the position of robot in real world is continuous, the quantized state defined by the conventional Q-learning can not provide a fine resolution of agents' state. And because the inflexibility of the discrete action output value, the efficiency of robots in real world is poor.

In the second experiment, two robots will try to reach the goal. They are connected with a straight bar, and the bar's length is flexible but limited. Its length limit is from 200 cm to 282.8427 cm. We use the master-slave structure to implement this method. The master robot leads and decides the path, the slave one tries to balance the straight bar and tries not to collide with the obstacles. The slave robot uses only the RVQ algorithm since its goal is not quite complicated. Its reward is based on the length of the straight bar. Since it must keep the straight bar not to short or not too long, the reward of 1 is given best when the length of straight bar is at the averaged length of maximum and minimum length. When the straight bar's length is longer or shorter than the averaged length, the reward value becomes less. Once the straight bar stocks with the obstacle, the reward then is set to zero. As well, when the slave robot bumps into obstacles or goes out the field, the reward is given zero, too.



(a) (b)

Fig. 6. (a) The trace of agents in SRV method after 100th trials. (b) The trace of agents in RVQ after 100th trials.

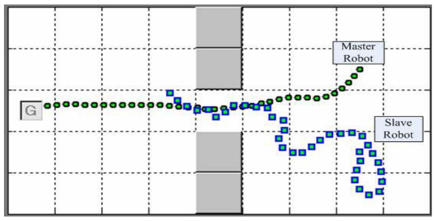


Fig. 7. The result trace of the robots in cooperation through a gate.

The master robot shares the information of its position to the slave robot. So the state space of the slave robot will be added one dimension of the relative position of master robot. The basic actions are up, right up, right, right down, down, left down, left and left up. The initial weights of slave robot are random. The speeds of the two robots are differentiated, master's highest speed is a little slower than slave's, because we must guarantee that slave can catch up with master. To perform the algorithm proposed above, we give a simple task to two robots which have a straight bar connected with each other. The goal of them is to cooperate for passing through the gate. Both agents perform actions and update Q-values simultaneously.

Fig. 6 (a) depicts the trace of both agents in the SRV method. The bold line square is the position of the first agent, and the dash line square is the position of the second agent. The number in the center of the square is the time step (trial) of the experiment. The trace of both robots with RVQ method is shown in Fig. 6 (b).

Fig. 7 is the trend of the two robots learning to pass the gate cooperatively. We can see at first 100 trials there still needs more than 500 steps to find the way to pass through the gate. Then when the number of training trials is greater than 100 times, the average of the steps of robots to reach the goal successfully is down to about 200 steps. And after the 450th trials, we can see that the robots have found an optimal solution and take about 10 steps to cooperate to pass through the gate. In the last demonstration, the length of the stick held by the robots increase one and half times after the robots are trained for the previous tasks. Fig. 7 shows the result trace of robots in cooperation through a gate and the steps of the master robot with RVQ take to get through a gate and reach the target point along the number of learning epochs.

V. Conclusion

In this paper we propose a method that can generate a continuous action and it is based on Q-learning. We compare the performance of the proposed RVQ with SRV and implement this algorithm to the multi-agent.

This algorithm uses three parameters to generate the action. In SRV learning algorithm, the critic network may be trapped into local maximum since it applies the LMS rule to update the weight. To overcome this problem, the stochastic recording real-valued unit is proposed to record the optimal reward. The result shows that it converges more quickly.

However, when it applies to the simulation, there are still some problems. If we train the Q-learning and the RVQ at the same time, the Q-table will be updated wrong often especially when the robot is near the obstacle. This make the training become inefficient. One solution is to check the state, when the state is actually maps to the action that Q-learning made, the Q-table then will be updated. But it decreases the chance to explore. In multi-agent cooperation, the problem is that the master doesn't know the existence of the slave one. It only knows how to pass through the obstacle and reaches the goal, but doesn't know how to cooperate with the slave. So this cooperation may fail at some other cases. The other problem is that the training must begin with Q-learning, when the Q-learning converges; the RVQ unit then finds the optimal solution. When the master robot's motivation is fixed, the slave one then can cooperate with the master robot and converge. This is not ideal at the learning.

The continuous state problem is always an issue in reinforcement learning. We can try another approach to improve the performance. Try converting the Cartesian coordinates into polar coordinate; this will be more general than the origin, or try tracing the dynamic targets. Maybe convert the weights into local and global information is not a bad idea. The structure in communication between Q-learning and RVQ maybe can be improved, such as considering the sojourn time [10], or elsewhere.

References

[1] K. S. Hwang and Y. B. Hsu, "A Self-Improving Fuzzy Cerebellar Model Articulation Controller with Stochastic Action Generation," *Cybernetics and Systems, An International Journal,* Vol. 33, No. 2, pp 101-128, 2002.

[2] C. J. C. H. Watkins. *Learning from Delayed Rewards.* PhD paper, Psychology Departments, Cambridge University, 1989.

[3] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279-292, 1992.

[4] Leslie Kaebling, Michael L. Littman, and Andrew W. Moore. "Reinforcement learning:A survey." *Journal of Artificial Intelligence Research,* 4:237-285, May 1996.

[5] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning:An Introduction*. MIT Press/Bradford Books, March 1998.

[6] A.G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Trans. Syst., Man, Cybern.,* Vol. SMC-13, pp. 834-846,1983.

[7] A.I. El-Osery and M. Jamshidi,"A stochastic learning automaton based autonomous control of robotic agents," *IEEE International Conference on Systems, Man and Cybernetics*, *Hammamet, Tunisia*, October 6-9, 2002.

[8] V. Gullapali, "A stochastic reinforcement learning algorithm for learning real-valued functions," Neural Net., Vol. 3, pp. 671-692, 1990.

[9] Albus, J. S. 1975a. A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Trans. ASME. J. Dynamic Syst. Meas.Control* 97 (September):220– 227./

[10] Tapas K. Das, Abhijit Gosavi, Sridhar Mahadevan, Nicholas Marchalleck, "Solving Semi-Markov Decision Problems Using Average Reward Reinforcement Learning." *Management Science,* Vol. 45, No. 4, April 1999.