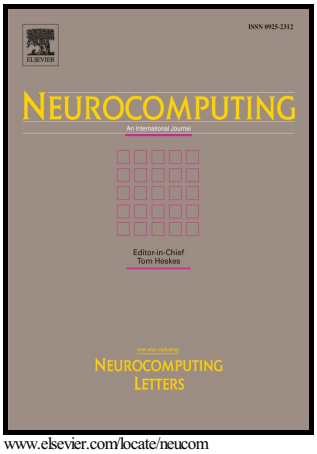


www.elsevier.com/locate/neucom

Path Planning of Multi-Agent Systems in Unknown Environment with Neural Kernel Smoothing and Reinforcement Learning

David Luviano Cruz, Wen Yu

PII: S0925-2312(16)31385-6
DOI: http://dx.doi.org/10.1016/j.neucom.2016.08.108
Reference: NEUCOM17751

To appear in: *Neurocomputing*

Received date: 14 December 2015
Revised date: 3 August 2016
Accepted date: 11 August 2016

Cite this article as: David Luviano Cruz and Wen Yu, Path Planning of Multi-Agent Systems in Unknown Environment with Neural Kernel Smoothing and Reinforcement Learning, *Neurocomputing* http://dx.doi.org/10.1016/j.neucom.2016.08.108

Path Planning of Multi-Agent Systems in Unknown Environment with Neural Kernel Smoothing and Reinforcement Learning

David Luviano Cruz, Wen Yu

Departamento de Control Automático

CINVESTAV-IPN (National Polytechnic Institute)

Mexico City, Mexico.

Abstract

Path planning is a basic task of robot navigation, especially for autonomous robots. It is more complex and difficult for multi-agent systems. The popular reinforcement learning method cannot solve the path planning problem directly in unknown environment.

In this paper, the classical multi-agent reinforcement learning algorithm is modified such that it does not need the unvisited state. The neural networks and kernel smoothing techniques are applied to approximate greedy actions by estimating the unknown environment. Experimental and simulation results show the proposed algorithms can generate paths in unknown environment for multiple agents.

1 Introduction

A multi-agent system includes several intelligent agents in an environment. Each agent has its independent behavior and coordinates with the others [35]. An important benefit of using the multi-agent system is that it can model the cooperation of real life situations. The multi-agent system is also used to learn new behaviors such that the performance of natural systems is predicted [40]. There are many other applications of the multi-agent systems, such as robotics team [13], distributed control [16], resource management [28], collaborative decision [3], and data mining [32].

Path planning generates a path from a starting-point to an ending-point with respect to some restrictions. It is important in some real-life problems, such as mobile robots, logistics, and game design [30]. The path planning problem can be solved by state searching. The single-agent path planning has limit states in an environment. For the multi-agent path planning, many units move simultaneously in the environment, and the dimension of the

configuration space increases exponentially [22]. The path planning of the multi-agent system becomes more difficult [17].

There are several methods of the multi-agent path planning. In [16], the high-gain decentralized control law is designed to solve the consensus problem [41]. In [4] and [12], the grid graph method is applied to design a state-task graph. In [38], the undirected graph is used for multi-agent path planning. In [11], the random tree algorithm is extended to update the paths. Artificial intelligence methods have learning ability, they can solve the path planing problem directly [6]. For the single agent, the genetic algorithm is used in [9], kernel smoothing is applied in [23], fuzzy logic is used in [21], iterative learning approach is applied in [25].

The reinforcement learning (RL) is the most popular method for multi-agent system [18][36]. It is also called multi-agent reinforcement learning (MARL) [7]. The object of MARL is to maximize a reward function defined by the environment and agents, such that the agents can interact with the environment [18]. At each learning step, the agent senses the environment, takes an action, and transits to a new state the environment [2]. The quality of each transition is evaluated by a reward function. The agents know which action has the most reward [33]. In order to generate a good action, RL feedback is needed [10]. The Q-learning algorithm is a popular RL feedback [34]. [39] gives the Q-learning algorithm for the single-agent. [1] proposes the multi-agents Q-learning method. Both of them assumes the environment can be described by a set of states.

In unknown or dynamics environment, if the desired information are available, the supervised learning methods can be applied. In [23], the neural networks are used to estimate the moving obstacles. In [15], the velocity is estimated for the unknown and bounded disturbances. In [13], single-agent path planning is realized by RL and the neural networks. In [19], RL is applied to multi-agent case with fuzzy logic technique, it does not have learning mechanism. However, if the desired information are not available. The above methods do not work.

This paper takes both advantages of the unsupervised learning (kernel smoothing) [27], and the supervised learning (neural networks) [26]. We combine them with RL to solve multi-agent path planning in the unknown environment. This hybrid algorithm includes two training phases:

1. The Win/Learn Fast-Policy Hill Climbing method (WoLF-PHC) [5] is modified. In this stage, the task model consists of the transitions and the reward functions. Each agent does not known the other actions and reward functions. The agents explore in the unknown environment and collect state-action information through the unsupervised kernel smoothing and modified WoLF-PHC.

2. The neural networks are integrated with the RL algorithm. We use the states in Stage

1 as desired values, such that the supervised learning of the neural networks can be applied. Finally, the action (controller) is given by a greedy policy from the state-action Q-table.

In this paper, we also show that after finite iterations, Q-values converge. Our algorithm is tested by two mobile robots Khepera in the dynamic environment. The experimental results show that our algorithms are simple and effective for multi-agent path planning in unknown environment.

2 Reinforcement learning for path planning of multi-agent systems

The current state of the environment which the agent observers is defined as s_t, the action of the agent is defined as a_t.The model of the single-agent reinforcement learning is a Markov decision process. It is defined as:

Definition 1 Single-agent RL process in dynamic environment is

$$\begin{aligned} f_t &: S_t \times A_t \times S_{t-1} \to [0,1] \\ \rho_t &: S_t \times A_t \times S_{t-1} \to \mathbb{R} \end{aligned} \tag{1}$$

where S_t and S_{t-1} are the environment states at t and $t-1$, A_t is the agent actions at time t, f_t is the state transition probability function from the time $t-1$ to the time t, ρ_t is the reward function.

In *dynamic* environment, the reward ρ_t (or penalty) received by the action a_t and the environment s_t, which relate to the previous action a_{t-1} and the environment s_{t-1}. In *static* environment, the reward ρ_t only depends on the action a_t and the environment s_t.

The behavior of the agent is described by the policy π, which specifies how does the agent choose its actions from the given state. The policy may be either static and dynamic. In static case $\pi = S \to A$, the policy is stationary. In dynamic case, $\pi_t : S \times A \to [0,1]$, the policy is time-varying.

The path planning is to find a policy such that the return R is maximized for the state s,

$$R^{\pi} = E\left\{ \sum_{k=0}^{\infty} \gamma^k r_{k+1} \middle| s_0 = s, \pi \right\} \tag{2}$$

where $\gamma \in [0,1)$ is the discount factor, r_{k+1} represents the cooperation goals. For example, if the agent has just grasped the target, $r_{k+1} = 1$; if the agent has reached the final goal position, $r_{k+1} = 10$; in all other situations, $r_{k+1} = 0$.

This form is the probabilistic state transition under the policy π. R also represents the reward accumulated by the agent.

The task of the agent is therefore to maximize its long-term performance. It is obtained by computing the state-action value function, called Q-function, defined by

$$Q^h : S \times A \rightarrow \mathbb{R}$$

It gives an expected return from policy π,

$$Q^{\pi}(s,a) = E\left\{ \sum_{k=0}^{\infty} \gamma^k r_{\,k+1} \middle| s_0 = s, a_0 = a, \pi \right\} \tag{3}$$

The Q-function is defined as

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{4}$$

Once Q^* is available, the policy is the largest Q-value,

$$\pi^* = \arg\max_{a} Q^*(s,a) \tag{5}$$

After Q^* is found, the greedy policy, which maximizes the Q-function, is needed. In multi-agent case, the Markov decision process becomes a game process.

Definition 2 The Multi-agent RL process is a tuple $\langle S, A_1, ..., A_n, f, \rho_1, ..., \rho_n \rangle$,

$$\begin{aligned} &\rho_i : S \times \mathbf{A} \times S \rightarrow \mathbb{R} \\ &f : S \times \mathbf{A} \times S \rightarrow [0,1] \\ &\mathbf{A} = A_1 \times A_2 .... \times A_n \end{aligned} \tag{6}$$

where n is the number of agents, S is the finite set of environment states, A_i $i = 1, ..., n$ are the finite sets of actions of the agents, ρ_i is the reward function of $i_$th agent.

The state transitions are result of the joint action $\mathbf{a}_k$ within all agents, $\mathbf{a}_k = \left[a_{1,k}^T, ..., a_{n,k}^T\right]^T$, $\mathbf{a}_k \in \mathbf{A}$,$a_{i,k} \in A_i$. The joint policy is defined as π. The return of the multi-agent depends on the joint policy

$$R_i^{\pi}(x) = E\left\{ \sum_{k=0}^{\infty} \gamma^k r_{\,i,k+1} \middle| s_0 = s, \pi \right\} \tag{7}$$

The Q-function of each agent depends on the joint action and the join policy

$$\begin{aligned} &Q_i^{\pi} : S \times \mathbf{A} \rightarrow \mathbb{R} \\ &Q_i^{\pi}(s,\mathbf{a}) = E\left\{ \sum_{k=0}^{\infty} \gamma^k r_{\,i,k+1} \middle| s_0 = s, \mathbf{a}_0 = \mathbf{a}, \pi \right\} \end{aligned} \tag{8}$$

Since the agents in multi-agent case act cooperative tasks, the reward functions and the returns of all the agents are the same, *i.e.*, $\rho_1 = \rho_2 = ...\rho_n$, $R_1^{\pi} = R_2^{\pi} = ...R_n^{\pi}$. All agents have the same goal to maximize the common return.

3 Reinforcement learning with kernel smoothing and neural networks

RL algorithm needs all states of the agents. If some states are not available, the MARL discussed above does not work. In this section, we use kernel smoothing and the neural networks to estimate the unavailable states, then send them to the MARL.

3.1 Multi-agent reinforcement learning in unknown environment

In this paper, the reinforcement learning algorithm for multi-agent systems, WoLF-PHC (Win or Learn Fast Policy Hill-Climbing) [5], is modified for unknown environment. This modification assures that any agent has its responses when the other agents remain stationary. This convergence property implies that the agent actions according to the probability distribution of the available actions are similar with the deterministic strategy.

The prior knowledge assumptions, such as the transition functions and the reward functions of the agents, are not needed.

We extend the Q-learning [39]. The agent experience consists of a sequence of distinct stages of episodes. For $n_$th episode, the agent is:

- Observes its current state x_n
- Selects and performs an action a_n
- Observes the subsequent state y_n
- Receives an immediate reward r_n
- Adjust its Q_{n-1} values using a learning factor α_n according to:

$$Q_n(x,a) = \begin{cases} (1-\alpha_n) Q_{n-1}(x,a) + \alpha_n [r_n + \gamma V_{n-1}(y_n)] \\ \text{if } x = x_n \text{ and } a = a_n \\ Q_{n-1}(x,a) \\ \text{otherwise} \end{cases}$$

where $\gamma \epsilon [0,1)$ is the discount factor, and

$$V_{n-1}(y) = \max_b [Q_{n-1}(y,b)] \tag{9}$$

Here $V_{n-1}(y)$ is the best the agent from the state y.

In the early stages of the learning, Q may not accurately reflect the policy. The initial $Q_0(x,a)$ is assumed to be given. The description of $Q_n(x,a)$ is a look-up table.

The policy hill climbing (PHC) algorithm keeps the Q values, and maintains the current mixed policy. The policy is enhanced by increasing the probability of selecting the highest valued according to the learning rate $\delta \in (0, 1]$. When $\delta = 1$, PHC is equivalent the Q-learning. If the greedy policy executes the highest valued with probability 1 in each step, PHC policy becomes the Q-learning. So it is reasonable to play mixed policies for mixed task [5].

In order to use the convergence property without sacrificing the rational property, the learning rate δ is modified as time variant. This modification strategy can give more time to the other agents, and enable all agents to adapt quickly each other. In two learning parameters case, we set $\delta_L > \delta_w$. This policy guarantees current agent is determined by setting current expected value is bigger than the average policy. It also approximates the equilibrium policy for mixed task in dynamic environments.

In order to solve the coordination problem of several agents, the WoLF-PHC algorithm is applied to each one. Any connection is broken, and the coordination is similar with the roles and communication [37]. Each agent has an implicit form of coordination. It learns and decides a good solution which is influenced by the other agents.

The learner converges to a stationary policy when the other agents do not move with their policies. The modified WoLF-PHC algorithm for agent i is expressed as

1. Let $\alpha \in (0, 1]$, the learning rate $\delta_l > \delta_w \in (0, 1]$, initialize

$$Q(s, a) = 0, \quad \pi(s, a) = \frac{1}{|A_i|} \tag{10}$$

   where $\pi(s, a)$ is the probability of the action a in the state s, $|A_i|$ is the cardinality of the set A.

2. Repeat

   - For state s, we select action a according to mixed strategy $\pi(s)$ with suitable exploration. At each step a random action with probability $\varepsilon \in (0, 1)$ is used. Then a greedy action with the greatest Q value in Q-table with probability $(1 - \varepsilon)$ is calculated
   - Obtain reward r and next state $\acute{s}$

$$Q(s, a) = (1 - \alpha)\, Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(\acute{s}, \acute{a}) \right) \tag{11}$$

     where $\acute{a} \in A_i$

   - Update the average $\bar{\pi}$

$$\begin{aligned} C(s) &= C(s) + 1 \\ \bar{\pi}(s, \acute{a}) &= \tfrac{1}{C(s)} \left(\pi(s, \acute{a}) - \bar{\pi}(s, \acute{a}) \right) \end{aligned} \tag{12}$$

- Move π to the policy with respect to Q-table

$$\pi(s,a) = \pi(s,a) + \Delta_{sa} \tag{13}$$

here the constrain and the probability distribution are

$$\begin{cases} \Delta_{sa} = -\delta_{sa} & \text{if } a \neq \arg\max_{\acute{a}} Q(s,\acute{a}) \\ \Delta_{sa} = \sum_{\acute{a}\neq a} \delta_{sa'} & \text{otherwise} \end{cases} \tag{14}$$

with $\delta_{sa} = \min\left(\pi(s,a), \frac{\delta}{|A_i|-1}\right)$ and

$$\begin{cases} \delta = \delta_w & \text{if } \sum_{a'} \pi(s,\acute{a})Q(s,\acute{a}) > \bar{\pi}(s,\acute{a})Q(s,\acute{a}) \\ \delta = \delta_l & \text{otherwise} \end{cases} \tag{15}$$

This modified WoLF-PHC algorithm still has the properties of the normal WoLF-PHC algorithm. It generates a state-action table for each agent. It can find the maximum Q-value for all particular states, and does not need prior knowledge for the learning system.

Q-value converges to certain value after finite training time. This indicates that the learning process can reach its maximum for all known states [37].

If the perceptions of the agents are different, the agents update their own Q-functions in different ways. The consistence of the Q-functions cannot be guaranteed. The dimension of Q-table becomes very large.

3.2 Kernel smoothing learning

The reinforcement learning algorithm can obtain the policy from all Q-values. Since most real applications have large or infinite state space, the modified MARL (10)-(15) cannot work well with limit states. Now we use kernel smoothing to approximate the unknown states for the MARL.

We have the observations s_t, taken at time points $t \in [\tau_1, \tau_2]$. We assume that the data come from the model

$$\hat{s}_{t+1} = \phi(s_t) + \epsilon_t$$

where $\phi(s_t)$ is an unknown smooth mean response curve, and ϵ_t is error. Our goal is to find the kernel smooth estimate $\hat{\phi}$ at some pre-specified time point t.

The kernel smooth is simply a weighted average of all data points

$$\hat{\phi}(s_t) = N^{-1} \sum_{k=\tau_1}^{\tau_2} W_t s_t a(t) \tag{16}$$

where $N = \tau_2 - \tau_1 + 1$, W_k is the weight sequence. The estimated state $\hat{s}_{t+1} \in R^n$.

A simple approach to represent the weight sequence W_t is to describe the shape of the weight function by a density function with a scale parameter, which adjusts the size and the form of the weights near s_t. It is quite common to refer to this shape function as a kernel $K[z]$. The kernel is a continuous, bounded and symmetric real function which integrates to one, $\int K[z]\,dz = 1$.

The weight sequence for kernel smothers is defined by

$$W_t s_t = K_t[s(t)] / \hat{f}_k(s_t)$$

where $\hat{f}(s_t) = N^{-1} \sum_{k=\tau_1}^{\tau_2} K_k[s(t)]$, $K_k[s(t)]$ is a Gaussian function

$$K_t[s(t)] = a \exp\left(\frac{\|s_t - \mathbf{c}_t\|^2}{2\sigma^2}\right), \quad a > 0 \tag{17}$$

Therefore Nadaraya-Watson [29] kernel regression for the data $[a(k), s(t)]$ with $t \in [\tau_1, \tau_2]$ is

$$\hat{\phi}(s(t)) = \frac{\sum_{t=\tau_1}^{\tau_2} \{K_k[s(t)]\, a(t)\}}{\sum_{t=\tau_1}^{\tau_2} K_k[s(t)]} \tag{18}$$

Statistical analysis of the Nadaraya–Watson kernel regression is difficult because it is defined as the ratio of two random variables $\hat{f}_t$ and K_t. In many important applications in signal processing and automatic control, a simpler form of the kernel regression may be used, which is considerably easier to analyze statistically. In these applications, all random variables have a constant probability density function. With these type of random variables, the summations involving the kernel function are equivalent to a Monte Carlo integration and

$$K_t[s(t)] = \frac{N}{\tau_2}$$

The Nadaraya–Watson kernel regression may then be replaced by the Priestley–Chao [31] regression, which is defined as

$$\hat{\phi}(s(t)) = \frac{\tau_2}{N} \sum_{t=\tau_1}^{\tau_2} K_t[s(t)]\, a(t) \tag{19}$$

In this paper, (18) is used to model the unknown environment. A possible way is design a wide kernel function to cover more data, such that these data correspond a similar behavior. A practical method is full width at half maximum [31]. The maximum width is defined as

$$\sigma_{\max} = 2\sqrt{2\ln(2)}\sigma \tag{20}$$

$\sigma_{\max}$ is chosen by the total number of the data, for example $\sigma_{\max} = \frac{N}{10}$, the width parameter of Gaussian function is

$$\sigma = \frac{N}{20\sqrt{2\ln(2)}} \tag{21}$$

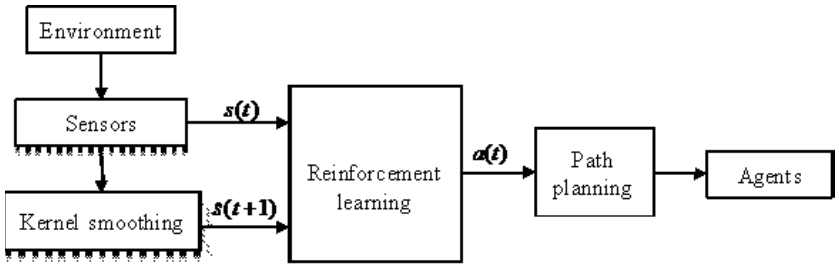


Figure 1: The workflow of this method

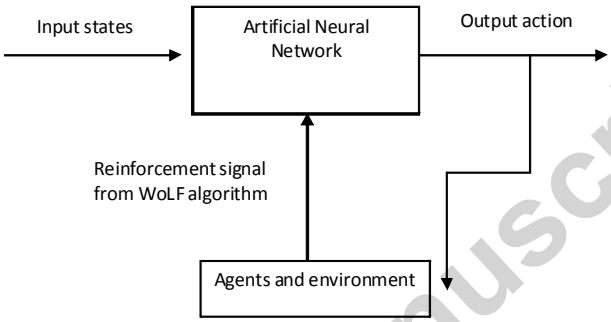


Figure 2: Trainning procedure with neural networks

where N is data length. We use center points of each Gaussian function $\left[x_j^*, y_j^*\right]$, $j = 1 \cdots v$ to construct the membership functions. For the data set $[a(t), s(t)]$, $k \in [1, N]$, we extract fuzzy product rules in the form of (18). x_j^* is center of the Gaussian function $\mu_{A_i^j}$, y_j^* is the center of the Gaussian function μ_{B^j}.

The workflow of this method is shown in Figure 1. Here the sensors obtain the environment information by now, while the kernel smoothing generates the future information such that the reinforcement learning can work.

3.3 Neural network learning

The neural network (NN)model has powerful ability of information fusion, fault tolerance, and learning. Because single layer NN can only solve the linear classified problem, the multilayer NN is used to estimate nonlinear states in the unknown environment . The training process is showed in Figure 2.

In this paper, the unknown states are approximated by the following multilayer neural network with three hidden layer

$$\hat{a}_{t+1} = W\sigma(V[s_t]) \tag{22}$$

where $W \in R^{n\times m}$, $V \in R^{m\times n}$, m is hidden node number which is designed by the user, n

is the number of agents driving in the environment, V is the weight in hidden layer, σ is a neural activation function, here we use Sigmoid function . The input to the NN is s_t, which is obtained from the modified WoLF-PHC in MARL learning phase. The output of the NN is the approximate action $\hat{a}$.

The estimated state $\hat{s}_{t+1} \in R^n$. We use supervised learning method to train the weights of (22). The training error $E(k)$ is between the sensory state and the action pattern in previous phase. With this definition, the output of NN correspond to the action with respect to the training samples.

The error at the output of neuron j is given as

$$e_j(k) = y_j(k) - d_j(k)$$

where $y_j(k)$ is the sensory state, $d_j(k)$ is the action pattern. The instantaneous sum of the squared output errors is given by

$$\varepsilon(k) = \frac{1}{2}\sum_{j=1}^{l} e_j^2(k)$$

where l is the number of neurons of the output layer. Using the gradient decent, the weight connecting neuron i to neuron j is updated as:

$$w_{ij}(k+1) = w_{ij}(k) - \eta\frac{\partial\varepsilon(k)}{\partial w_{ij}(k)}$$

where η is learning rate. The term $\frac{\partial\varepsilon(k)}{\partial w_{ij}(k)}$ can be calculated as

$$\frac{\partial\varepsilon(k)}{\partial w_{ij}(k)} = \frac{\partial\varepsilon(k)}{\partial e_j(k)}\frac{\partial e_j(k)}{\partial y_j(k)}\frac{\partial y_j(k)}{\partial v_j(k)}\frac{\partial v_j(k)}{\partial w_{ij}(k)}$$

The partial derivatives are given by $\frac{\partial\varepsilon(k)}{\partial e_j(k)} = e_j(k)$, $\frac{\partial e_j(k)}{\partial y_j(k)} = 1$, $\frac{\partial y_j(k)}{\partial v_j(k)} = \varphi_j'\left[v_j(k)\right]$, $\frac{\partial v_j(k)}{\partial w_{ji}(k)} = y_i(k)$. Usually, linear function is used for the output layer, so

$$w_{ij}(k+1) = w_{ij}(k) - \eta y_i(k)\, e_j(k) \tag{23}$$

Similar w_{ki} can be updated as

$$w_{ki}(k+1) = w_{ki}(k) - \eta\frac{\partial\varepsilon(k)}{\partial w_{ki}(k)}$$

and

$$\begin{aligned}\frac{\partial\varepsilon(k)}{\partial w_{ki}(k)} &= \frac{\partial\varepsilon(k)}{\partial e_j(k)}\frac{\partial e_j(k)}{\partial y_j(k)}\frac{\partial y_j(k)}{\partial v_j(k)}\frac{\partial v_j(k)}{\partial y_i(k)}\frac{\partial y_i(k)}{\partial v_i(k)}\frac{\partial v_i(k)}{\partial w_{ki}(k)} \\ &= \left\{e_j(k)\,\varphi_j'\left[v_j(k)\right]W_{ij}\right\}\varphi_i'\left[v_i(k)\right]y_k(k) = e_i(k)\,\varphi_i'\left[v_i(k)\right]y_k(k)\end{aligned}$$

So

$$w_{ki}(k+1) = w_{ki}(k) - \eta y_k(k)\, e_i(k)\,\varphi_i'\left[v_i(k)\right] \tag{24}$$

where $e_i(k)\,\varphi_i'\left[v_i(k)\right]$ is error backpropagation.

3.4 Path planning

The states of the agents are observed and referred to an state-action table (Q-table). Not all states are available in the Q-tables caused by the changes in the environment, the kernel smoothing estimator (16) and the neural networks (22) are used to reach the goal.

This integration of Q-table, the kernel smoothing, and the neural networks can improve performance in dynamic environment. When a new state is encountered, the relative state information is fed to the kernel smoothing. The model is trained. The output of the model is the approximated action. In this way we can avoid retraining in the reinforcement learning

The proposed method can be summarized as the following steps:

1. **Catch the initial state of each WoLF-PHC training cycle**: the current state of the agents with respect to the environment are captured through sensors (infrared, sonar).

2. **Limit the States captured** : Limitation of the captured states reduces the whole number of states that the agents are required to complete the task, this conduct to save computing time.

3. **Set the action available for each agent**: In each step time the agents are required to perform an action with a degree of coordination , in that way is required to select in advance the most reliable actions in order to keep minimized the actions space of the task.

4. **Estimate the state-action Q-values for each agent :** The numerical reward for each action is computed and given after it is performed, this calculation is done by WoLF-PHC algorithm then the Q-values obtained are save in a lookup table (Q-table).

5. **Repeat steps 2-4 until the agents achieve the goal:** The training cycle will be finished if the agents' learning trail reach a maximum number of steps , if they collide with other agents or obstacle of the environment or if they achieve the final goal.

6. **Repeat steps 2-5 until Q-values converge :** This happens when the Q-values remain unchanged or they are under a certain bound set in advance.

7. **Final state-action Q-table:** The final the state-action table is set up for each action by locating the maximum Q-value for each state .

8. **Kernel Training phase :** The final Q-table obtained from the WoLF-PHC algorithm is used to train the kernel, each column of the Q-table represents a state which is introduced to the kernel to train it , then the kernel provides an approximate action when the agents are facing new states that were not learning in the former stage.

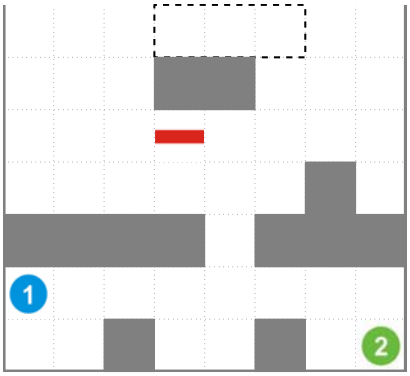


Figure 3: Multi-agent transportation

4 Simulation and experimental results

The objects of the simulations are to check and validate the operation of our proposed method. We also use two mobile robots to compare our algorithms with the other techniques.

4.1 Simulations

There are two agents in an environment girded with 7×8 cells, see Figure 3. The agents need to transport a target (in red color) to the home base (upper line) in minimum time, while to avoid obstacles. It is a typical cooperation task [8].

The initial position of the two agents are shown in the Figure 3. Each time, they can move one cell. They will stop if the target cell is not empty. The first object is the two robot reach the target. The two robots tell the control center if they already grasped the target. The second object is the two robot move together, and reach the home base, see Figure 4.

In this task, we assume the agents do not know the obstacles in the environment. The start point and the home base position are known. We first use MATLAB to simulate the environment.

The state variables for each agent i are: two position coordinates $x \in (1, 2, 3, .., 8)$, $y \in (1, 2, 3, .., 7)$. The relation between the agent and target is $g \in (left, right, free)$. The actions of the agent are $A = \{left, right, up, down, stand\}$. The state vector is $s = [x_1, y_1, g_1, x_2, y_2, g_2]$, with $|S| = (8 \cdot 7 \cdot 3)^2 = 112896$. The Q-tables obtained by WoLF-PHC algorithm has $|S| \times 5 = 564480$ elements, the parameters used in this task are shown in Table 1.

After 24 trainings trial the Q-tables converge to the Q-values, see Figure 5.

The kernel modeling training process is after the WoLF-PHC learning. The Q-values in

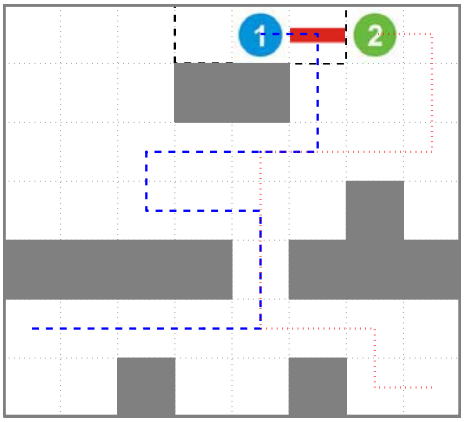


Figure 4: Optimal path after the WoLF-PHC learning phase

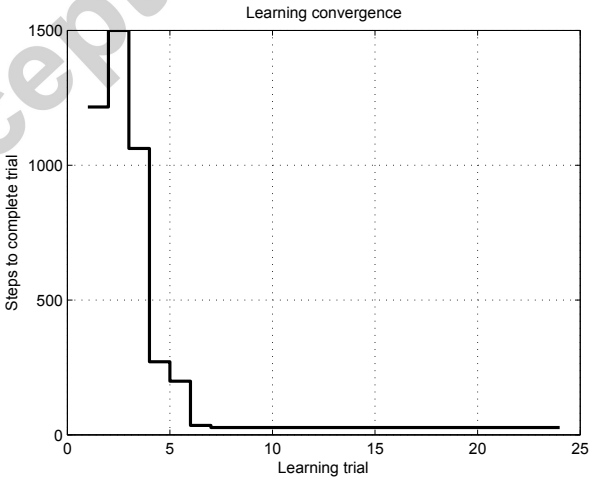


Figure 5: Performance after first learning phase.

Table 1: The parameters of WoLF-PHC algorithm.

Exploration Probability ε	0.7
Exponential Decay Factor λ	0.5
Police step size for WoLF-PHC	$\delta_{win} = 0.1$
Police step size for WoLF-PHC	$\delta_{lose} = 0.35$
Learning rate α	0.1
Discount factor γ	0.98

Table 2: The parameters of the simulation task.

Number of hidden layers	3
Number of Neurons 1^{st} hidden layer	50
Number of Neurons 2^{nd}t hidden layer	20
Number of Neurons 3^{rd} hidden layer	10
Activation Function output layer	Linear function
Activation Function for the hiddens layers	Sigmoid function

the first phase are the training samples. The corresponding actions are trained for 23 epoch. In the second learning phase, the maximum training iterations in each step is 1500.

The inputs are $s = [x_1, y_1, g_1, x_2, y_2, g_2]$. The input layer has 6 neurons , the output layer has 2 neurons which are the actions of the agents. The actions are: 1-left,2-right,3-down,4-up and 5-stand. The specifications of the kernel smoothing is show in Table 1.

The performance of the path planning is proven by Figure 8. The initial position of the mobile robots are random on the bottom line. The final path of the agents are shown in Figure 6.

We can see that the agents successfully reach the target and transport it to the home base in the most of the cases. Both agents do not take the shortest route from random initial position, because of the approximation error of the kernel smoothing. Another reason is there are repetitive states of the agents.

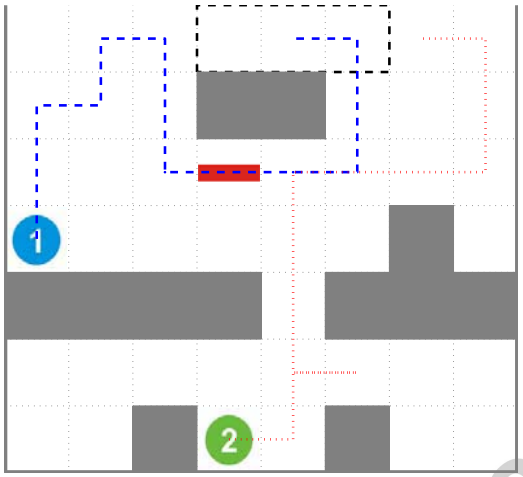


Figure 6: Final path found by the kernel smoothing method from a random initial position in simulation task

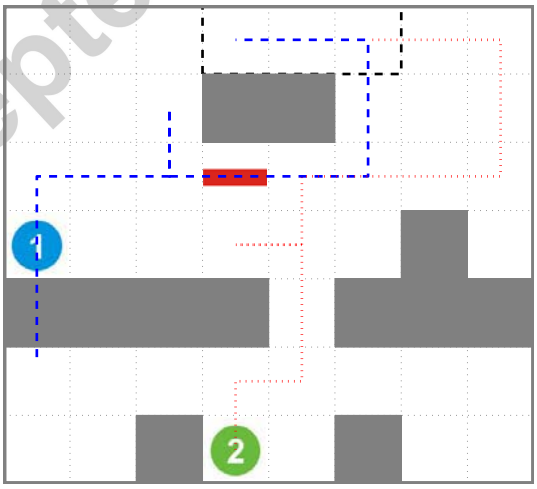


Figure 7: Final path found by the neural network method from a random initial position in simulation task

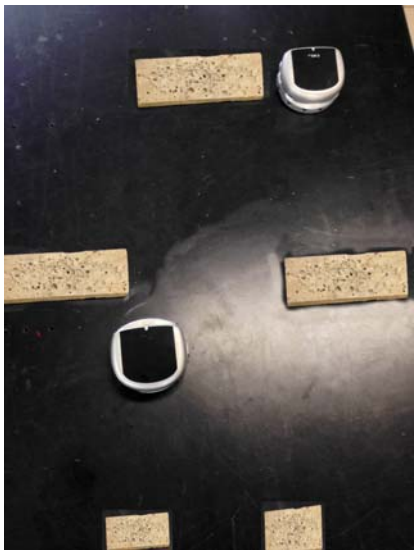


Figure 8: Experiment set-up

4.2 Experimental results

In order to validate the performances of our kernel smoothing based reinforcement learning, we use two mobile robots (Khepera) [20] to accomplish the same goal as the above simulation. We assume that the robots does not have prior knowledge of the obstacles in the working environment.

The start points are randomly selected. The training steps are 2000. If they cannot reach the final goals within 2000, the experiment is restarted. The experiment is inside a room see Figure 8.

The Khepera devices are small-size robot designed for real-world applications. Each Khepera mobile robot has 5 sensors which are placed around the robot as in Figure 9. These ultrasonic sensors are used to reconstruct the odometry and detect natural features in the environment. In the initial training phase the current states are captured with respect to the unknown environment. The state S has 16 elements (8 for each agent) ,

$$S_t = [l_{a,c}, d_a, p_a, g_a] \tag{25}$$

where $a = 1, 2$ and $c = 1, .., 5$, the elements $l_{a,c}$ are the 5 infrared sensors, d_a represents the relative distance between each Khepera robot and the target, p_a is the relative angle from each mobile robot's current forward moving axis to the target, g is the variable indicating if the agent is grasping the target.

There are three values for $l_{a,c}$: 0 indicates the obstacles or agents are near; 1 indicates the obstacles or agents are in medium distance; 2 means obstacles or agents are relatively far.

Table 3: Parameters of the robot

Environment Size	3000 x 1800 mm
Starting points for the 1^{st} test	agent 1=$(337, 115)$ agent 2=$(1462, 115)$
Starting point for the 2^{nd} test	agent 1=$(112, 1500)$ agent 2=$(514, 115)$
Starting point for the 3^{rd} test	agent 1=$(337, 115)$ agent 2= $(562, 1500)$
Target Position	$x \in [675, 900]$ $y \in [2307, 2538]$
Goal position	$x \in [450, 1350]$ $y \in [2760, 3000]$

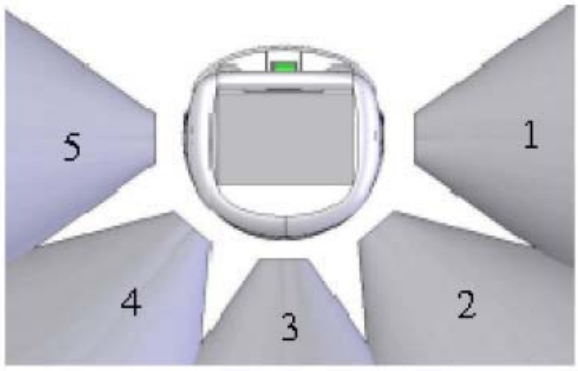


Figure 9: Position of the 5 UltraSonic sensors.

pa (the relative angle to the target or goal) are divided in eight degrees $(0 - 8)$. 0 represents the smallest distance or angle. 8 represents the greatest relative distance or angle from the current Khepera's position to the target or goal.

In order to reduce state number, the learning space should be defined. A large learning space will cause long time computation. A small learning space cannot represent the entire environment and the task [42]. The parameters used in the implementation are shown in Table 3. The ultrasonic sensors are utilized to detect the obstacles in the environment.

In order to achieve the cooperative task, each agent is required to choose one action from the following available actions :

- Move forward 5 *cm*
- Turn 25° in clockwise direction
- Turn 25° in counter clockwise direction
- Stand-Still

Table 4: The parameters used in the experiment.

Exploration Probability ε	0.8
Exponential Decay Factor λ	0.7
Police step size for WoLF-PHC	$\delta_{win} = 0.25$
Police step size for WoLF-PHC	$\delta_{lose} = 0.45$
Learning rate α	0.15
Discount factor γ	0.96

The reinforcement learning algorithm relies strongly on the sensors data. There are noises in the ultrasonic sensors, our learning phase and the Q-table updating are robust with respect to the noises. We stop the sensor reading when the controller is applied, such that the joint action is stable. In the learning process, the robots move slowly in order to avoid the collisions and improve accuracy [14].

The reward for this task is: fully cooperative, grasping the target, and transporting it to the goal position. We define $r_{k+1} = 1$ when the agent is grasped; $r_{k+1} = 10$ when the robot arrives the the goal position; $r_{k+1} = 0$ in all other situations.

The state S_t of the mobile robots with respect to the environment is calculated according to Table 4. The estimation of the Q-values for the state-action pair of each agent is through our modified WoLF-PHC algorithm. The actions of the agents which are obtained by the reinforcement signal

The training cycle is finished if a robot collides with a obstacle or with another agent. In this case the learning process is rebooted. Otherwise, the learning phase continues until the Q-values do not change. The state-action pairs are shown in Figure 10.

The output obtained by the previous learning phase is in the Q-values. Every set of data consists of 16 elements. They are used as the training samples for the kernel smoothing and the neural networks.

Each column of the training matrix represents a state. The neural networks uses the backpropagation, the learning rate is 0.08. The Kernel Smoothing and the neural network aspire to improve the controller to learn from the environment . The parameters of the neural networks are shown in Table 5.

The final state-action table is obtained for each state S_t by locating the maximum Q-values,

$$\pi^* = \arg\max_a Q^*(S, A) \tag{26}$$

In order to compare the performance between the kernel smoothing method and the

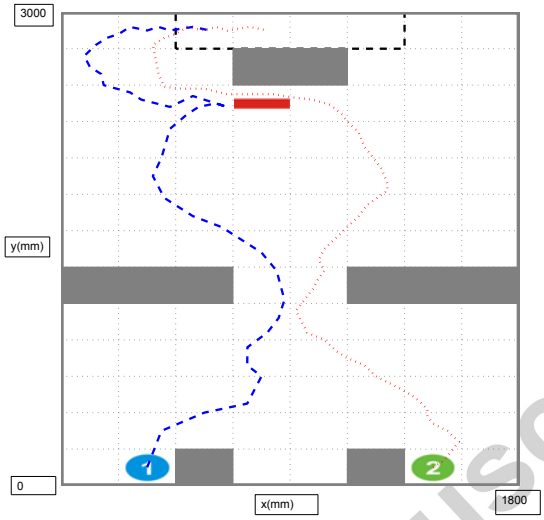
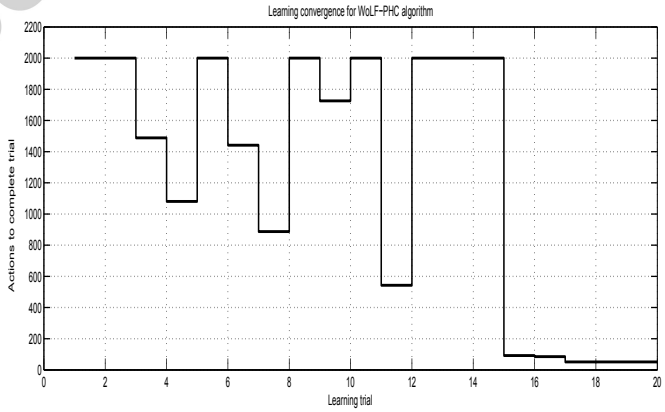


Figure 10: Controller's performance in real time implementation after WoLF-PHC (first learning phase)



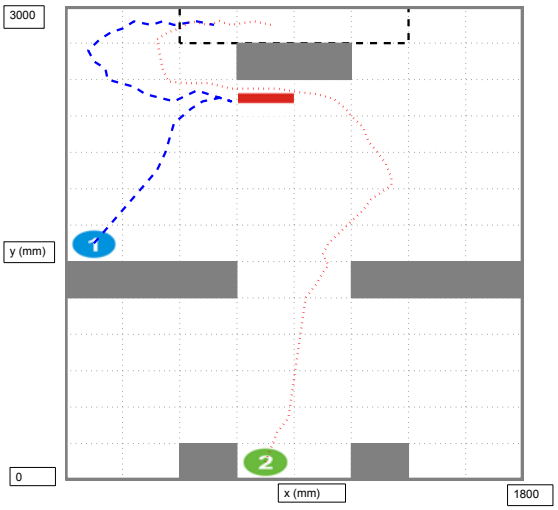


Figure 11: Path found by the Kernel smoothing method from a random initial position.

neural network method , a random initial position is chosen for each agent ,so the robots will face new states that the previous learning phase won't be able to provide the actions to accomplish the final goal,when happens this situation the kernel smoothing and the neural network will provide the actions for each agent , these actions were previously learned using the samples obtained by the WoLF-PHC learning algorithm , as is shown in the figures 11, 12 .

The results obtained from the simulation and the real time implementation with the Khepera robots show that the combinations of the WoLF-PHC algorithm with the neural networks and the kernel smoothing are robust in the path planning, and can overcome the necessity of remaking of the reinforcement learning.

One of principal research tasks of the autonomous mobile robot is to develop a robust cooperative skill in dynamic and unknown environments. This can be done by multiple methods. Using only one method may no obtain better performances. In this paper, kernel smoothing and the neural networks are combined with the reinforcement learning. The results of the experimental task and the simulations verify the feasibility and robustness of our methods.

5 Conclusion

In this paper, we successfully overcome the difficulty of reinforcement learning for unknown environment. Combination of the kernel approximation and neural networks with the WoLF-

Table 5: The parameters of the neural network.

Number of hidden layers	4
Number of Neurons 1^{st} hidden layer	40
Number of Neurons 2^{nd}t hidden layer	25
Number of Neurons 3^{rd} hidden layer	15
Number of Neurons 4^{th} hidden layer	10
Activation Function output layer	Linear function
Activation Function for the hiddens layers	Sigmoid function

PHC algorithm can get over the drawback of the reinforcement learning, such as slow learning speed, time consuming and impossible learning in unknown environments.

The robustness of the changes in the environment depends on the approximation and generalization of the kernel method. The simulations and the experimental results show that the proposed algorithms have learning ability without any prior knowledge on the obstacles and environment. The algorithms are not harshly affected by the fluctuations of the sensors. Experiment results show the proposed algorithm can generate paths in the environment for multiple agents.

These algorithms can be easily extended to robot navigation when there are coordinations and dimensionality problems including dynamic changes in the environment.

References

[1] J. Abdi, B. Moshiri, B. Abdulhai, Emotional temporal difference Q-learning signals in multi-agent system cooperation: real case studies, *IET Intelligent Transport Systems*, Vol.7, No.3, 315 - 326, 2013.

[2] I. Arel, C. Liu, T. Urbanik and A. G. Kohls, Reinforcement learning-based multi-agent system for network traffic signal control, *IET Intelligent Transport Systems*, Vol.4, No.2, 128-135, 2010.

[3] D.Barbucha, Search modes for the cooperative multi-agent system solving the vehicle routing problem, *Neurocomputing* , Volume 88, Pages 13-23 , 2012.

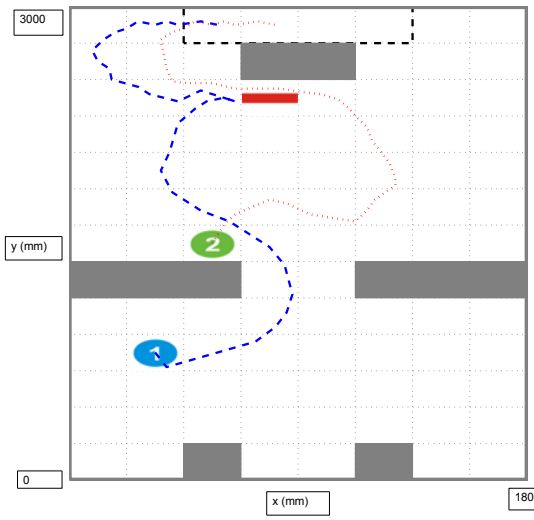


Figure 12: Path found by the neural network method from a random initial position.

[4] S. Bhattacharya, M. Likhachev and V. Kumar, Multi-agent Path Planning with Multiple Tasks and Distance Constraints, *IEEE International Conference on Robotics and Automation*, 953-959, Anchorage, Alaska, USA, 2010

[5] M. Bowling and M. Veloso, Multiagent learning using a variable learning rate, *Artificial Intelligence*, Vol.136, No.2, 215-250, 2002.

[6] S. T. Brassai, B. Iantovicsb, C. Enachescu, Artificial intelligence in the path planning optimization of mobile agent navigation, *Procedia Economics and Finance*, Vol.3, 243-250, 2012

[7] L. Busoniu, R. Babuska and B. De Schutter, A Comprehensive survey of Multiagent Reinforcement Learning , *IEEE Transactions on Systems ,Man and Cybernetics. Part C: Applications and Reviews*, Vol.38, No.2, 164-181, 2008

[8] L.Busoniu , R. Babuska and B. De Schutter *Multi-agetn Reinforcement Learning : An overview.* Innovations in MASs and Applications-1 SCI 310 pp-183-221 Springer Verlag Berlin Heidelberg, 2010.

[9] Z. Cai and Z. Peng, Cooperative Coevolutionary Adaptive Genetic Algorithm in Path Planning of Cooperative Multi-Mobile Robot Systems, *Journal of Intelligent and Robotic Systems,* , Vol.33, No.1, 61-71, 2002

[10] V. Cherkassky and F. Mulier, *Learning from data: Concepts,Theory and Methods,* , Wiley-IEEE Press, Chichester, 1998

[11] V. R. Desaraju and J. P. How, Decentralized Path Planning for Multi-Agent Teams in Complex Environments using Rapidly-exploring Random Trees, *IEEE International Conference on Robotics and Automation*, 4956-4962, Shanghai, China, 2011.

[12] J. Fu and J. Wang, Adaptive Consensus Tracking of High-order Nonlinear Multi-agent Systems with Directed Communication Graphs„ *International Journal of Control, Automation, and Systems*, Vol. 12, No. 5, 919-929, 2014

[13] V. Ganapathy , S. Chin and H. Kusama Joe, Neural Q-learning controller for mobile robot, *IEEE/ASME International Conference on Advanced Intelligent Mechatronics* , 863-868, Singapore, 2009.

[14] V.Ganapathy, S. C. Yun adn W. L. D. Lui *Utilization of webots and Khepera II as a Platform for neural Q-learning controllers* 2009 IEEE Symposium on Industrial Electronics and Applications (ISIEA 2009) Octuber 4-6 Kuala Lumpur , Malaysia.

[15] H. Hu, L. Yu, G. Chen, and G. Xie, Second-Order Consensus of Multi-Agent Systems with Unknown but Bounded Disturbance, *International Journal of Control, Automation, and Systems*, Vol.11, No.12, 258-267, 2013

[16] A. Wei, X. Hu, and Y. Wang, Consensus of Linear Multi-Agent Systems Subject to Actuator Saturation, *International Journal of Control, Automation, and Systems*, Vol.11, No.4, 649-656, 2013

[17] S-H.J i, J-S. Choi, and B-H. Lee, A Computational Interactive Approach to Multi-agent Motion Planning, *International Journal of Control, Automation, and Systems*, Vol.5, No.3, 295-306, 2007

[18] L. P. Kaelbling , M. L. Littman and A. W. Moore, Reinforcement Learning: A Survey, *Journal of Artificial Intelligence Research*, Vol.4, No.1, 237-285, 1996

[19] M. Kaya and R. Alhajj, Modular fuzzy-reinforcement learning approach with internal model capabilities for multiagent systems, *IEEE Transactions on Systems ,Man and Cybernetics, Part B: Cybernetics*, Vol.34, No.2, 1210-1224, 2004.

[20] http://www.k-team.com/, *K-Team Corporation,* 2013

[21] N. Kwak, S. Ji, B. Lee, A knowledge base for dynamic path planning of multi-agents, *16th IFAC World Congress,* 1363-1363, Czech Republic, 2005

[22] S. M. LaValle, *Planning Algorithms,* Cambridge University Press, Cambridge, U.K., 2006.

[23] H. Li, S. X. Yang, M. L. Seto, Neural-Network-Based Path Planning for a Multirobot System With Moving Obstacles, *IEEE Transactions on Systems ,Man and Cybernetics. Part C: Applications and Reviews*, Vol.39, No.4, 410-420, 2009.

[24] C. Li , J. Zhang and Y. Li *Application of artificial neural Network Based on Q-learning for mobile robot path planning* , Proceedings in the 2006 IEEE International Conference on Information Acquisition August 20-23 , 2006 Weihai , Shandong , China

[25] Y. Liu and Y. Jia, Formation Control of Discrete-Time Multi-Agent Systems by Iterative Learning Approach , *International Journal of Control, Automation, and Systems*, Vol.10, No.5, 913-919, 2012

[26] D. Luviano and W. Yu Multi-agent path planning in unknown environment with reinforcement learning and neural network , *International Conference on Systems, Man and Cybernetics (SMC)*, San Diego California, 2014.

[27] D.Luviano and W.Yu, Path Planning in Unknown Environment with Kernel Smoothing and Reinforcement Learning for Multi-Agent Systems, *12th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE'15)*, Mexico City, Mexico, 2015

[28] P.Minga, J.Liu, S.Tan, S.Li, L.Shang, X.Yu, Consensus stabilization in stochastic multi-agent systems with Markovian switching topology, noises and delay, *Neurocomputing,* Volume 200, Pages 1-10, 2016.

[29] E. A. Nadaraya, On Estimating Regression, *Theory of Probability and its Applications,* Vol.9, No.1, 141-142, 1964.

[30] J. Park, J-H. Kim, J-B. Song, Path Planning for a Robot Manipulator based on Probabilistic Roadmap and Reinforcement Learning, *International Journal of Control, Automation, and Systems*, Vol.5, No.6, 674-680, 2007

[31] M. B. Priestley and M. T. Chao, "Non-parametric function fitting," *J. Royal Statistical Soc., Ser. B*, vol. 34, pp. 385–392, 1972.

[32] A.M.Quteishat, C-P.Lim, J.weedale, L.C.Jain, A neural network-based multi-agent classifier system, *Neurocomputing,* Vol.72, pages 1639-1647, 2009

[33] S. Sen and G. Weiss *Learning in multiagent systems. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence,* MIT Press Cambridge ,1999

[34] T. J. Sejnowski and G. Hinton, *Unsupervised Learning :Foundations of Neural Computation*, MIT Press,1999

[35] P. Stone and M. Veloso, Multiagent systems: A survey from machine learning perspective, *Autonomus Robots,* Vol.8, No.3, 345-383, 2000

[36] R. Sutton and A. Barto, *Reinforcement Learning: an introduction*. Cambridge, MA : MIT Press, 1998.

[37] N. Vlassis . *A concise Introduction to Multi Agent Systems amd Distributed Artificial Intelligence .* Synthesis Lectures in Artificial Intelligence and Machine Learning . Morgan & Claypool Publishers 2007.

[38] K-H. Wang and A. Botea, MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees, *Journal of Artificial Intelligence Research,* Vol.42, 55-90, 2011

[39] C. Watkins and P. Dayan, Q Learning: Technical Note, *Machine Learning,* Vol.8, 279-292, 1992

[40] M. Wooldridge, *An Introduction to MultiAgent Systems*, John Wiley & Sons, 2002.

[41] X. Yang and Wang, Finite-gain Lp Consensus of Multi-agent Systems, *International Journal of Control, Automation, and Systems,* Vol.11, No. 4, 666-674, 2013

[42] S. C. Yun, S. Parasuraman and V, Ganapathy *Mobile Robot Navigation: Neural Q-learning* , Advances in Computing & Inf. Technology ,AISC 178, pp 259-268