Proceedings of the 2006 IEEE/RSJ
International Conference on Intelligent Robots and Systems
October 9 - 15, 2006, Beijing, China

# Multi-robot Box-pushing: Single-Agent Q-Learning vs. Team Q-Learning

Ying Wang
Department of Mechanical Engineering
The University of British Columbia
Vancouver, BC, Canada, V6T 1Z4
ywang@mech.ubc.ca

Clarence W. de Silva, *Fellow IEEE*
Department of Mechanical Engineering
The University of British Columbia
Vancouver, BC, Canada, V6T 1Z4
desilva@mech.ubc.ca

***Abstract* - In this paper, two types of multi-agent reinforcement learning algorithms are employed in a task of multi-robot box-pushing. The first one is a direct extension of the single-agent Q-learning, which does not have a solid theoretical foundation because it violates the static environment assumption of the Q-learning algorithm. The second one is the Team Q-learning algorithm, which is a multi-agent reinforcement learning algorithm, and is proved to converge to the optimal policy. The states, actions, and reward function of the algorithms are presented in the paper. Based on the two Q-learning algorithms, a fully distributed multi-robot system is developed. Computer simulations are carried out using the developed system. The simulation results show that the two algorithms are effective in a simple environment. It is shown, however, that the single-agent Q-learning algorithm does a better job than the Team Q-learning algorithm in a complicated and unknown environment with many obstacles.**

***Index Terms* - *Multi-robot Systems, Team Q-learning, Multiagent Reinforcement Learning, Box-pushing.***

## I. INTRODUCTION

This paper addresses the multi-robot box-pushing problem employing machine learning. In a multi-robot box-pushing task, multiple autonomous robots cooperatively push an box to a goal location in a dynamic or static environment with obstacles. The multi-robot box-pushing problem has been of value in both theoretical research and practical application. On the one hand, it is a good platform to study multi-robot/multi-agent architectures and motion coordination approaches. On the other hand, in the physical world, multiple robots often need to cooperatively transport a very big or heavy box, which is usually difficult for a single robot to handle.

Several approaches have been developed to solve the problem of multi-robot box-pushing or multi-robot object transportation. Parker has suggested a distributed behavior-based architecture (L-ALLIANCE) and applied it to a multi-robot box-pushing task [1]. Wang et al. have employed the concept of "object closure" and a leader-follower formation control scheme to move a box with three mobile robots [2]. Huntsberger et al. have developed a behavior-based multi-robot architecture (CAMPOUT) and applied it to a NASA project of robotic work crew [3]. Yamada et al. have designed a rule-based action selection method for an adaptive multi-robot box-pushing task [4]. Miyata et al. also have addressed a dynamic task planning architecture for a multi-robot object transportation system in an unknown static environment [5].

All these approaches share a common disadvantage, however: the robots are not equipped with learning capabilities and they cannot improve their skills through experience. However, in an unknown dynamic environment, because it is difficult to predict all potential cases in advance in the design stage, a robot team with stationery skills alone will be doomed to be weak and not robust.

Some researchers have attempted to employ the reinforcement learning technology in multi-robot box-pushing tasks to cope with the difficulties, which result from unknown dynamic environments [6]-[8]. They have directly extended the single-agent Q-learning algorithm to the multi-robot domain and obtained some positive results. However, such extensions do not have a solid theoretical foundation because they violate the static environment assumption in single-agent Q-learning. In the multi-robot domain, the environment is no longer static because the multiple robots affect it simultaneously [9].

On the other hand, several multi-agent reinforcement algorithms have been developed in the multi-agent community. They include MiniMax Q-learning algorithm, Nash Q-learning algorithm, and Friend-or-Foe Q-learning algorithms [9]. In particular, for purely cooperative games, Littman suggests the Team Q-learning algorithm, which is a simplified version of the Nash Q-learning algorithm [10]. All these algorithms assume the dynamic environment and allow each robot to observe the actions of its peers before it makes decisions. Under some conditions, these algorithms are proved to converge to the optimal policy [9].

The main disadvantage of the multi-agent reinforcement learning algorithms is that they result in a tricky and extensive learning space (state/action space) because each robot needs to monitor not only its own actions but also the actions of its peers. When the number of robots increases, the resulting extensive learning space will make the algorithm worthless. However, when the number of robots increases, the single-agent Q-learning algorithm still can work because its learning space is fixed.

In this paper, the Team Q-learning algorithm is extended to a multi-robot box-pushing task, and its results are compared with the single-agent Q-learning algorithm. Based on the

1-4244-0259-X/06/$20.00 ©2006 IEEE

simulation results, we will show that both algorithms are effective, and will study differences between them.

## II. MDPs and Single-Agent Q-learning

Q-learning is commonly used as a reinforcement learning algorithm for its simplicity [10]. It finds the optimal policy in the Markov Decision Processes (MDPs) problem. The MDPs can be defined by a tuple $<S, A, T, R, \beta>$, where

- $S=\{s_1, s_2, \ldots, s_n\}$, is a set of states of the environment
- $A=\{a_1, a_2, \ldots, a_m\}$, is a set of actions available to each agent
- $T: S \times A = \prod(S)$, is a transition function, which decides the next environment state $s'$ when an agent selects an action $a_i$ under the current state $s$. $\prod(s)$ is the probability distribution over the states.
- $R: S \times A \rightarrow \Re$, is a reward function, which gives the immediate reward when the agent takes an action $a_i$ under current state $s$.

In the MDPs problem, an agent should find the optimal policy to map its interaction history of current action choice so that it will maximize the sum of discounted rewards $\sum_{i=0}^{\infty} \beta^i r_{t+i}$, where $r_{t+i}$ is the reward received in the $i_{\text{th}}$ step in the future and $0 \le \beta < 1$ is the discount rate. The optimal policy can be found through the value iteration method.

However, in the physical world, the transition function $T$ and the reward function $R$ are usually unknown. Therefore, the value iteration method cannot be directly used to find the optimal action-selection policy for the agent. In this case, Q-Learning can be used to find the optimal policy without knowing the transition and reward functions in advance. A typical Q-learning algorithm is presented as follows:

- For each state $s_i$ and action $a_j$, initialize the table entry $\hat{Q}(s_i, a_j)$ to zero, and initialize $\tau = 0.99$
- Observe the current state $s$
- Do repeatedly the following:
  - Probabilistically select an action $a_k$ with probability $P(a_k) = \frac{e^{\hat{Q}(s,a_k)/\tau}}{\sum_{l=1}^{m} e^{\hat{Q}(s,a_l)/\tau}}$, and execute it
  - Receive an immediate reward $r$
  - Observe the new state $s'$
  - Update the table entry for $\hat{Q}(s,a_k)$ as follows:

    $\hat{Q}(s,a_k) = (1-\varepsilon)\hat{Q}(s,a_k) + \varepsilon(r + \beta \max_{a'} \hat{Q}[s', a'])$,

    where, $0 \le \varepsilon < 1$ is the learning rate
  - $s \leftarrow s'$, $\tau \leftarrow \tau * 0.999$

After a sufficient number of iterations, the $\hat{Q}(s_i, a_j)$ will converge to the optimal value.

## III. Stochastic Games and Team Q-learning Algorithm

Most multi-agent reinforcement learning algorithms are based on the framework of Stochastic Games (SGs) [9]. A learning framework of SGs is defined by a tuple $<S, A_1, \ldots, A_n, T, R_1, \ldots, R_n, \beta>$, where

- $S$ is a finite set of states and $n$ is the agent number.
- $A_1, \ldots, A_n$ are the corresponding finite sets of the actions available to each agent.
- $T: S \times A_1 \times A_2 \times \ldots \times A_n \rightarrow S$, is a transition function which maps the current state to the next state.
- $R_i: S \times A_1 \times A_2 \times \ldots \times A_n \rightarrow \Re$, is a reward function which describes the reward received by the $i_{\text{th}}$ agent.
- $0 < \beta < 1$ is the discount factor.

In a SGs framework, the joint actions of the agents determine the next state and the rewards received by each agent. Based on the interaction history, each agent tries to find the optimal action-selection policy to maximize the expected sum of the discount rewards in the future. For a purely cooperative game in which each agent shares the same goal and receives the same payoff with its peers, Littman suggests the Team Q-learning algorithm, which is a simplified version of the Nash Q-learning algorithm, to find the optimal policy [10]. The Team Q-learning algorithm is described as follows.

- For each state $s_i \in S$ and joint action $(a_1, \ldots, a_n) \in (A_1, \ldots, A_n)$, initialize the table entry $\hat{Q}(s_i, a_1, \ldots, a_n)$ to zero
- Observe the current state $s$
- Do repeatedly the following:
  - Let $(a_1^*, \ldots, a_n^*) = \arg\max_{(a_1, \ldots, a_n)} Q(s, a_1, \ldots, a_n)$.

    Execute the joint action $(a_1^*, \ldots, a_n^*)$.
  - Receive an immediate reward $r$
  - Observe the new state $s'$
  - Update the table entry for $\hat{Q}(s, a_1^*, \ldots, a_n^*)$ as follows:

$$\hat{Q}(s, a_1^*, \ldots, a_n^*) = (1-\varepsilon)\hat{Q}(s, a_1^*, \ldots, a_n^*) + \varepsilon(r + \beta \max_{a_1, \ldots, a_n} \hat{Q}[s', a_1, \ldots, a_n])$$

    where, $0 \le \varepsilon < 1$ is the learning rate
  - $s \leftarrow s'$

After a sufficient number of iterations, the $\hat{Q}(s_i, a_1, \ldots, a_n)$

will converge to the optimal value. In a purely cooperative game, compared with the Nash Q-learning algorithm, the Team Q-learning algorithm requires less computational resources, and has the same performance [10].

The above algorithm uses the greedy policy to choose actions. However, this policy may not explore an adequate number of states to converge to the optimal policy. Therefore, in this paper, the GLIE policy (Greedy in the Limit with Infinite Exploration) is used to guarantee convergence to optimum [10].

## IV. Multi-robot Box-pushing with Reinforcement Learning

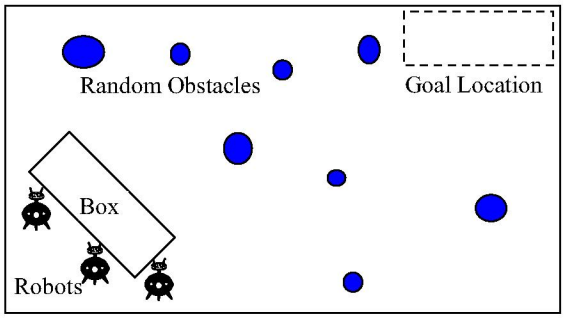



Fig. 1 A multi-robot box-pushing task.

Multi-robot box-pushing is a long-standing yet difficult problem in the multi-robot domain. In this task, multiple autonomous robots have to cooperatively push an box to a goal location and orientation in an unknown dynamic environment. The box is sufficiently big and heavy so that no single robot can complete the task without the help of other robots. In addition, each robot only possesses local sensing capability and the obstacles may occur and disappear randomly in the path, so the robots are unable to plan the trajectory in advance. In such an unknown dynamic environment with continuous space, any static planning technology will fail. Although dynamic planning [5] may be useful to complete the task, it will fail if the environment changes quickly and the computation speed is not fast enough. Therefore, the multi-robot box-pushing problem is a good benchmark for evaluating various multi-robot architectures and understanding how cooperative behaviors emerge among robot members in a multi-robot system. At the same time, it also has many practical applications in the physical world.

Because we think that the learning-based reactive architecture is a better solution to this challenging problem than a static/dynamic planning architecture, in this paper, both single-agent Q-learning algorithm and Team Q-learning algorithm are employed for the multi-robot box-pushing problem. Each robot is equipped with the Q-learning algorithms. When the robots move the box, each robot identifies the current environmental state, and then independently employs the Q-learning algorithm as given before to select the "optimal" action to push the box. After the action is taken, the reward is assessed and assigned to the robots. The robots then update their own Q-tables with the reward information and continue to push the box. Therefore, it is a fully distributed learning-based decision-making system. In other words, the multi-robot box-pushing problem is solved with a purely learning-based reactive architecture instead of complicated static or dynamic planning approaches.

### *A. The environment states*

Because the environmental space in the multi-robot box-pushing problem is continuous, in order to employ the Q-learning algorithm, it is converted into a set of finite states. Each environmental state is coded into a 13-bit binary code, as shown in Fig.2.

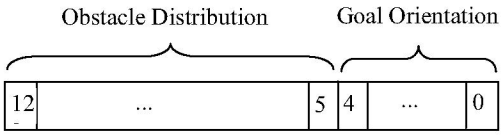



Fig. 2 The binary coding of the environmental states.

In Fig.2, the higher 8 bits (from bit 5 to bit 12) in the code represent the current obstacle distribution around the box within the detection radius. The lower 5 bits (from bit 0 to bit 4) represent the orientation of the goal relative to the current box pose. The coding rule is explained in Fig.3.

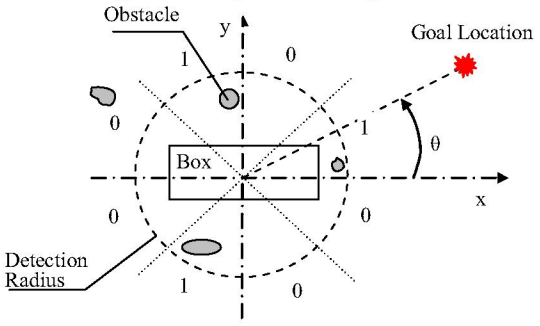



Fig. 3 Coding rule of the environmental state.

In Fig.3, the robots possess local sensing ability only. Consequently, only those obstacles within the detection radius can be detected and coded into the higher-8-bits of the environmental state. In order to describe the obstacle distribution in Fig.3, the detection circle is divided into 8 equal sectors. Each sector corresponds to one of the higher-8-bits of the environmental state. If at least one obstacle is detected in this sector, the corresponding bit value becomes "1". Otherwise, it becomes "0". For example, in Fig.3, the higher-8-bits of the environmental state are "10100100", which indicates the current obstacle distribution around the box within the detection radius.

The lower-5-bits of the environmental state indicate the goal orientation relative to the current box pose. The goal orientation angle $\theta$ in Fig.3 is used to calculate the lower-5-bits of the state using the following formula:

$$State_{bit(0\sim4)} = INT(\theta/(360.0/32)) \quad (1)$$

where, *INT()* is a function to covert a float value into an

integer.

*B. Robot Actions*

In the physical world, a robot usually can push a box at any contact point. It means that there are infinite possible ways that the robots are able to push the box. In this paper, the Reinforcement Learning is extended to the multi-robot domain. Therefore, the action space is simplified by assuming that only six actions are available for each robot. The available actions $(a_1 \sim a_6)$ are represented in Fig.4.

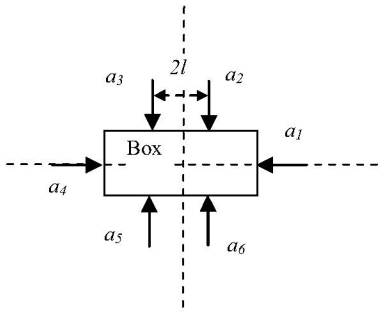



Fig.4 Pushing actions available to each robot.

*C. Reward function*

In this paper, a common payoff reward strategy is employed. Specifically, each robot will get the same reward after it pushes the box for a short period of time $\Delta t$. The reward consists of three parts, as follows:

(1) The reward for pushing the box toward the goal location

$$R_{distace} = (Dist_{old} - Dist_{new})c_d \quad (2)$$

where, $Dist_{old}$ and $Dist_{new}$ are the distances between the goal location and the box center before and after taking the pushing actions, and $c_d$ is a coefficient.

(2) The reward for the rotation behavior of the box

$$R_{rotation} = \cos(\alpha_2 - \alpha_1) - 0.9 \quad (3)$$

where, $(\alpha_2 - \alpha_1)$ is the rotational angle of the box when it is pushed by the robots in the time period $\Delta t$. While a small rotation is encouraged, a large rotation is punished because it will make handling the box by the robots very difficult.

(3) The reward for obstacle avoidance

$$R_{obstacle} = \begin{cases} = 1.0 & if\ no\ obstacle \\ = 1.5(\sin(\psi/2) - 0.3) \end{cases} \quad (4)$$

where, $\psi$ is the angle between the direction of the net force and the direction of the obstacle closest to the net force.

Finally, the reward is calculated as follows:

$$R = w_1 R_{distance} + w_2 R_{rotation} + w_3 R_{obstacle} \quad (5)$$

where, $w_1 \sim w_3$ are the weights, and $w_1 + w_2 + w_3 = 1.0$.

## V. Simulations

A simulation system has been developed using the Java language to simulate a multi-robot box-pushing system with the Reinforcement Learning algorithm presented in the previous sections. The environment dimensions are $950 \times 700$ and box dimensions are $120 \times 80$. There are three robots, denoted by circles with a radius of 2 in the subsequent figures, which push the box to the goal location. The box is sufficiently big and heavy so that no individual robot can handle it without the help of the peer robots. The Java multi-thread programming technology is used to implement the parallel operations of the robots.

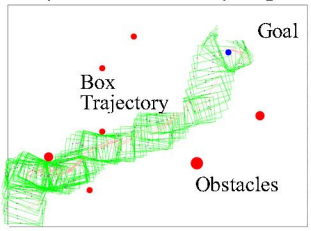
When the simulation system is started, based on the current environmental state, the robots independently select their actions with the single-agent Q-Learning or Team Q-learning algorithm. Then they begin to push the box after synchronization. The box is moved for a short period of time $\Delta t$, and the reward is collected and distributed equally among the robots. This period is called a "step" in the box-pushing process. After the robots receive the reward, they begin to start the next step of pushing until the goal location is reached or the total number of steps exceeds the maximum value. In this paper, the time period of push an box from its initial location to the goal location is called a "round" of box-pushing.

Before the robots can work, they need to train themselves to learn the knowledge of box-pushing. In the training stage, when the box reaches the goal location or the total number of steps in one round exceeds the maximum number of steps, this round of box-pushing ends and a new round of box-pushing is started. Before the next round of box-pushing is started, the locations of the box, the obstacles, and the goal are selected randomly. However, the Q-table of each robot is inherited from the last round. When the total number of rounds reaches the desired number, the training process ends and the Q-tables are saved into a file. The Q-tables represent the box-pushing policy learned by the robots.

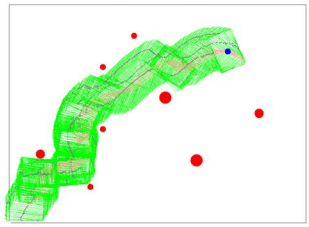
Some parameter values of the simulation system are given below: $\beta$ =0.9, $\tau = 0.9$, $\varepsilon = 0.8$, $c_f = 0.2, c_t = 0.02, c_d = 0.5, w_1 = 0.7, w_2 = 0.05, w_3 = 0.25, l = 10$, the maximum step number per round is 5,000, and the total number of rounds in the training stage is 10,000.

*A. Simulation results without training*

Some simulation results are presented in Fig.5. Note that the robots in Fig.5 do not receive any training before they begin to push the box. Accordingly, all the robots are novice and they do not have any experience in box pushing.




(a1)



(a2)

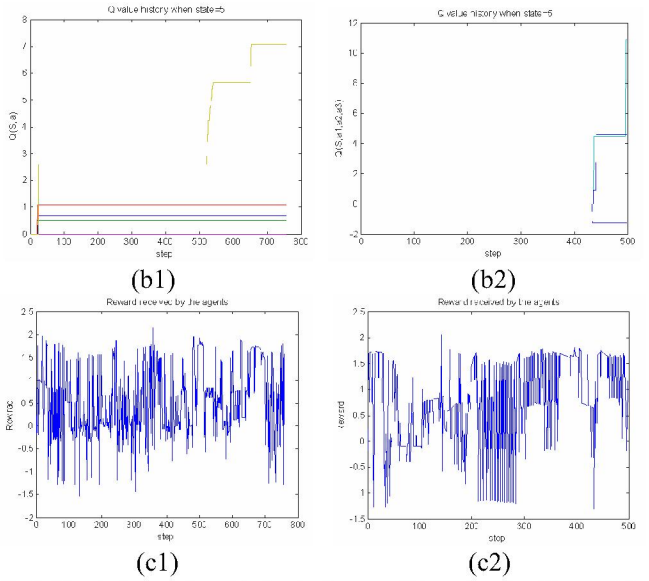
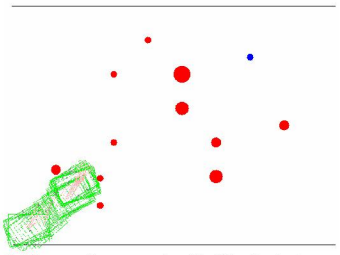



Fig.5. Simulation results with the single-agent Q-learning algorithm and the Team Q-learning algorithm: (a1) The box-pushing process with the single-agent Q-learning algorithm; (a2) The box-pushing process with the Team Q-learning algorithm; (b1) The Q value history (state=5) in the single-agent Q-learning algorithm; (b2) The Q value history (state=5) in the Team Q-learning algorithm; (c1) The received reward in the single-agent Q-learning algorithm; (c2) The received reward in the Team Q-learning algorithm.

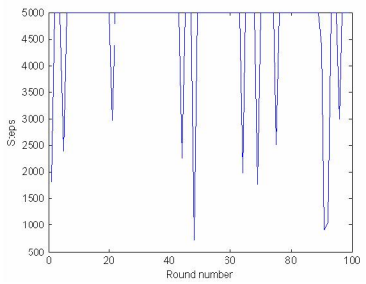
From Fig.5 (a1) and (b1), it is seen that both Q-learning algorithms can successfully push the box to the goal location in a simple unknown environment. Because each robot independently makes decisions and chooses its actions, it is a fully distributed multi-robot system. From Fig.5 (a2) and (b2), it is seen that the $Q$ values converge to fixed values. In Fig.5 (a3) and (b3), it is observed that the rewards received by the robots tend to fluctuate. This fluctuation results from hitting the unknown obstacles in the pushing process or disturbance from the actions of its peers. However, most of the time, the robots receive positive rewards.

*B. Comparison between two Q-learning algorithms*

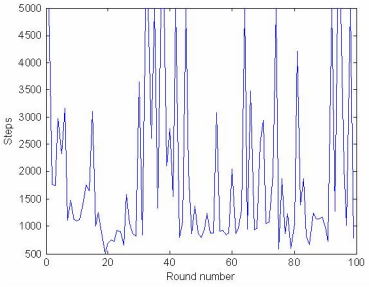
In Fig.6, the performance of the single Q-learning algorithm before and after training is compared. Here, the performance index is reciprocal of the total steps per round. All simulations are based on the same tricky environment with eleven obstacles, which is shown in Fig.6 (a).



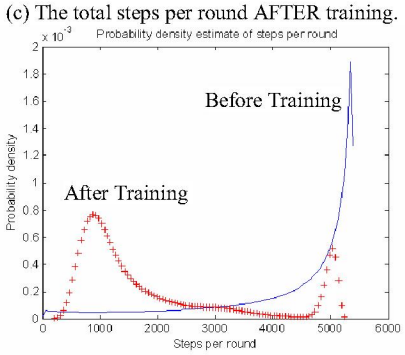
(a) The environment with 11 obstacles.




(b) The total steps per round BEFORE training.




(c) The total steps per round AFTER training.

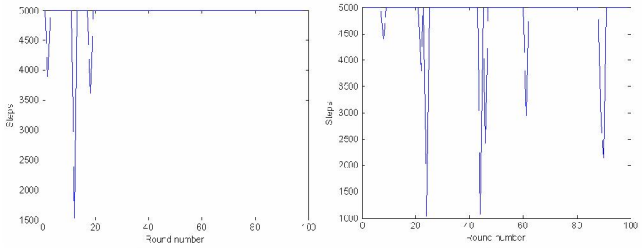


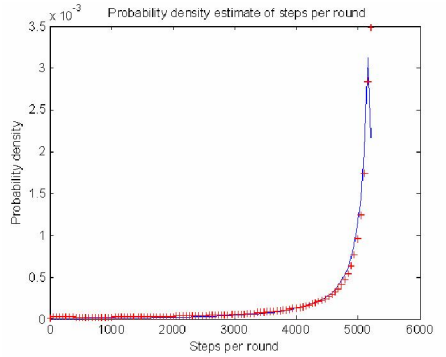

(d ) Probability density estimate of steps per round

Fig.6. Performance index comparison of the single-agent Q-learning algorithm before and after the training stage.

From Fig.6, it is seen that the performance index is significantly improved in the training stage. After training, the total number of steps per round is decreased from 5,000 to 1,000. Because the maximum number of steps is set at 5,000 for a round of box-pushing, a total number of steps of 5,000 means that the robots fail to push the box to the goal location within the desired number of steps in that round. Fig.6 indicates that the success rate of box pushing greatly increases after training because the robots learn and improve the box-pushing policy through experience.

Similarly, the performance of the team Q-learning algorithm before and after training is compared in Fig.7. The simulations are based on the same environment as in Fig.6 (a).






(c)

Fig.7. Performance index comparison of the Team Q-learning algorithm before and after the training stage. (a) The total number of steps per round BEFORE training. (b) The total number of steps per round AFTER training. (c ) Probability density estimate of steps per round before and after training.

From Fig.7, the following conclusion is drawn: Training is not required in the Team Q-learning algorithm. The robots do not learn any useful policy for pushing the box. It appears that the Team Q-learning algorithm does not help the robots to improve their skills using the experience received during the training stage. The total number of steps per round has a high probability of 5,000 before and after training. It means that the robots always fail to push the box to the goal location in the desired steps even after training.

By comparing Fig.6 (d) with Fig.7 (c), one observes a further interesting result: The single-agent Q-learning algorithm seems to do a better job than the Team Q-learning algorithm even though the latter has a much stronger theoretical foundation.

A possible explanation for this phenomenon is that some random actions are necessary to solve the local maximum problem in a complicated unknown environment for a multi-robot box-pushing task. Although the Team Q-learning algorithm with GLIE policy does take random actions, the probability of random action is still too low. Therefore, the Team Q-learning algorithm can be easily trapped in a local maximum.

The single-agent Q-learning algorithm is different since each robot/agent does not observe its peer's actions when it makes decisions. The peer's actions are regarded as a type of random disturbance or random "noise" in the environment. When this type of random "noise" is large, it is harmful to the team performance. However, if the random "noise" is small, it can help the robot team to escape from a local maximum. That is why the single-agent Q-learning algorithm does a better job than the Team Q-learning algorithms in the present simulations.

There are two possible ways to improve performance of the Team Q-learning algorithm in an unknown environment with complicated obstacles. First, increase probability of random actions to overcome the local maximum problem. Second, increase the training numbers so that the robot can explore enough states and converge to the optimal policy.

## VI. Conclusions

In this paper, the performance of two types of Q-learning algorithms is comparatively studied in the context of multi-robot box pushing. First, the single-agent Q-learning algorithm was directly extended into the multi-robot domain. Such extensions do not possess a solid theoretical foundation because they violate the static environment assumption of the algorithm. Second, the Team Q-learning algorithm was introduced into the multi-robot box-pushing field, which was specifically designed to solve the purely cooperative stochastic game problem. This algorithm was shown to converge to the optimal policy if an adequate number of explorations are executed. Both algorithms were implemented in a multi-robot box-pushing system. The simulation results showed that both algorithms were effective in a simple environment without obstacles or with a very few obstacles. However, in a complicated environment with many obstacles, the simulation results showed that the single-agent Q-learning algorithm did a better job than the Team Q-learning algorithm. It is believed that more random disturbances in the former help it to escape from local maxima.

## Acknowledgment

Funding for the work reported in this paper has come from the National Science and Engineering Research Council (NSERC) of Canada, Canada Foundation for Innovation (CFI), and the K.C.WONG Education Foundation of Hong Kong.

## References

[1] L. E. Parker, "Lifelong adaptation in heterogeneous multi-robot teams: response to continual variation in individual robot performance," *Autonomous Robots*, Vol.2000, No.8, pp.239-267, 2000.

[2] Z. Wang, Y. Hirata, and K. Kosuge, "An Algorithm for testing object caging condition by multiple mobile robots," *Proc. of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Edmonton, Alberta, Canada, pp.2664-2669, 2005.

[3] T. Huntsberger, P. Pirjanian, and A. Trebi-Ollennu, "Campout: A control architecture for tightly coupled coordination of multi-robot systems for planetary surface exploration," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol.33, No.5, pp.550-559, September 2003.

[4] S. Yamada and J. Saito, "Adaptive action selection without explicit communication for multi-robot box-pushing," *IEEE Trans. on Systems, Man, and Cybernetics,* Vol.31, No.3, pp.398-404, August 2001.

[5] N. Miyata, J. Ota, T. Arai, and H. Asama, "Cooperative transportation by multiple mobile robots in unknown static environment associated with real-time task assignment," *IEEE Trans. on Robotics and Automation,* Vol.18, pp.769-780, October 2002.

[6] E. Martison and R. C. Arkin, "Learning to role-switch in multi-robot systems," *Proc. of the 2003 IEEE International Conference on Robotics & Automation,* Taipei, Taiwan, 2003, pp.2727-2734.

[7] M. J. Mataric, "Reinforcement learning in the multi-robot domain," *Autonomous Robots,* Vol.1997(4), pp.73-83, 1997.

[8] Y. Wang and C. W. de Silva, "An object transportation system with multiple robots and machine learning," *Proc. of the 2005 American Control Conference (ACC 2005)*, Portland, OR, 2005, pp.1371-1376.

[9] E. Yang and D. Gu, "Multiagent reinforcement learning for multi-robot systems: A survey," *http://citeseer.ist.psu.edu/yang04multiagent.html.*

[10] M. L. Littman, "Value-function reinforcement learning in Markov games," *Journal of Cognitive Systems Research*, Vol.2002(1), pp.1-12, 2001.