Proceedings of the 2006 IEEE International Conference on Robotics and Automation
Orlando, Florida - May 2006

Transfer of Learning for Complex Task Domains: A Demonstration Using Multiple Robots

Sameer Singh and Julie A. Adams
Department of Electrical Engineering and Computer Science
Vanderbilt University, Nashville, Tennessee 37235
{sameer.singh, julie.a.adams}@vanderbilt.edu

Abstract— **This paper demonstrates a learning mechanism for complex tasks. Such tasks may be inherently expensive to learn in terms of training time and/or cost of obtaining each training pattern. Learning simple, safe tasks and extending them to more complex tasks can cause faster convergence to the solution. This method has been formalized and demonstrated on a simulated multiple robot (multi-robot) scenario. The objective is to effectively search out and destroy stationary hostile agents present in an unknown urban terrain map. Using the presented method, the robots learn how to effectively map the area, and then improve their learning modules for the complex task. The robots are simple behavioral agents with minimal communication.**

I. Introduction

This paper aims to improve learning for robots faced with complex tasks. Training for such tasks is very tedious and time consuming since each robot has to be trained upon many, varied instances. Creating this large number of instances could be costly in terms of monetary value and time.

It should be more efficient to train the robot on a simpler task, for which creating training instances are cheaper and faster, and then transfer the learning to the more complicated task. The creation of a "simpler" task from the complex task requires considerable task understanding since the number of states and actions must be reduced. This task abstraction must follow a few rules in order to perform efficiently.

Once a simpler task has been determined, the robot learns to optimally perform this task. Then, using prior knowledge regarding the relationship between the simple and the final task, the learned module is extended to include more states and actions. This extended module is suitable for learning the final task. Continued learning with the extended module allows the final task to reach an optimal solution faster, reducing the cost incurred when learning without the intermediate step. This paper describes the method and demonstrates its efficiency on a multiple robot task.

The objective of the multiple robot team is to remove a number of stationary hostile agents in an interior urban scenario. The urban map search is the simple task on which the robots are first trained, after which they are deployed onto the maps containing hostile agents.

The following section discusses the related literature. Section III formalizes the method to extend learning. Section IV provides detailed task descriptions, along with the robot representation. Section V describes the task simplification, the learning methods, and other related issues. Section VI decribes the experiments and presents the results. Section VII contains a discussion regarding the contributions of this work. The conclusion is provided in Section VIII.

II. Related Work

Learning is a heavily studied field. Traditional methods of completely specifying the environment have severe limitations. Real world environments usually consist of unobservable and dynamic characteristics. The incorporation of learning into robotics allows robots to dynamically choose actions based on their previous experiences.

Work in behavior based robotics, both single and multiple robot systems, consists mainly of behaviors competing with each other. The relationship representing which behavior to choose given the current state, is learned via feedback from the previous runs [1], [2]. Matarić [1] applied reinforcement learning, similar to this paper, to a team of foraging robots. It provided continuous reinforcement to various behaviors until an optimal policy was achieved. Parker [2] demonstrated a framework for creating cooperative behavior based systems while introducing a learning element.

The relationship between the behaviors and the state can be learned in isolation [3], [4]. Davesne and Barret [3] presented a set of incrementally learning agents that achieved tasks via behavior reinforcement. Fuentes and VanWie [4] used a simple hill climbing technique to learn the relationship between the states and behaviors. Due to the inter-dependencies within these behaviors, these methods do not work well for larger, tightly coupled tasks. Task decomposition into behaviors has been applied by decomposing them into subtasks and learning them separately. These subtasks are then divided across a set of robots who combine their information to complete the task. Malak and Khosla provided such a framework [5].

The complex task studied in this work is a variation of urban search and rescue. An urban map consists of hostile agents and the robots' objective is to clear all hostile agents. Similar tasks containing hostile agents have been studied, such as multi-robot patrol with reinforcement learning [6]. The simpler task used in this paper is a mapping task. Various aspects of map learning have been studied. An example of coordinated mapping by multiple robots has been demonstrated by Rekleitis et al. [7]. Map Learning has also been implemented using artificial neural networks [8], [9].

0-7803-9505-0/06/$20.00 ©2006 IEEE

III. Extending Learning

The existing methods face a few problems. First, independently learning dependent behaviors may lead to unnecessary training to relearn large portions of the dependent behaviors. Also, the sub-tasks are usually divided according to the physical aspects or functionality, for example by sensors required by the sub-tasks. This decomposition may lead to unnecessarily dividing the task into more sub-tasks. The presented method does not use a number of parallel subtasks during training but instead chooses a *maximal* subtask.

Formally, let T denote the complex task. The optimal policy for T, $\pi_T(s, a)$ provides a utility value for each state s ($s \in S_T$, the set of states for task T) and a corresponding possible action a ($a \in A_T$, the set of actions for task T). If a robot chooses the action with the highest utility value (using the optimal policy) for the given state, it shall function optimally. The learning objective is to obtain an optimal policy for T.

A simpler task (V) consists of a set of states (S_V) and a set of actions (A_V) that are subsets of S_T and A_T, respectively. The optimal policy for V, $\pi_V(s, a)$ is extended to cover the states $S_T - S_V$ and the actions $A_T - A_V$. During this extension, prior knowledge regarding V and T is employed to accelerate convergence, but is not required. This paper demonstrates that learning is achieved faster than learning T from $\pi_T(s, a) = 0, \quad \forall s \in S_T$ and $a \in A_T$.

It should be noted that the defined policy does not apply only to a specific learning algorithm, but can be easily translated to or from any algorithm. If it is viewed as a table of inputs and respective output values, then this policy is a simple functional mapping that is fed into a neural network or decision tree. Conversely, the policy can be extracted from the learning algorithm by providing all the different possible inputs and storing the output values in the policy.

Determining the simple task that lies within a more complex one is a non-intuitive process. There are a few properties that the simpler task should hold:

- It should encompass as much of the final task as possible.
- Training should be inexpensive. For example, if the robot can learn the task via simulations, this is preferred.
- The simple task should contain elements of the final task that are inherently difficult to learn.

These properties reduce the problems with a hierarchical task division. First, since only one subtask is considered, the extra time needed to learn the dependence between various subtasks in other methods is vastly reduced. Also, since the choice of the simpler task is based only on the learning aspect, it may not be interpretable as a subtask (since it contains a set of states and actions for the final task) but shall be one that optimizes the learning time and cost.

IV. System Description

The multi-robot task is to clear stationary hostile agents from an unmapped interior urban (UIU) scenario. Many task details have been abstracted for purposes of demonstrating the learning behavior.

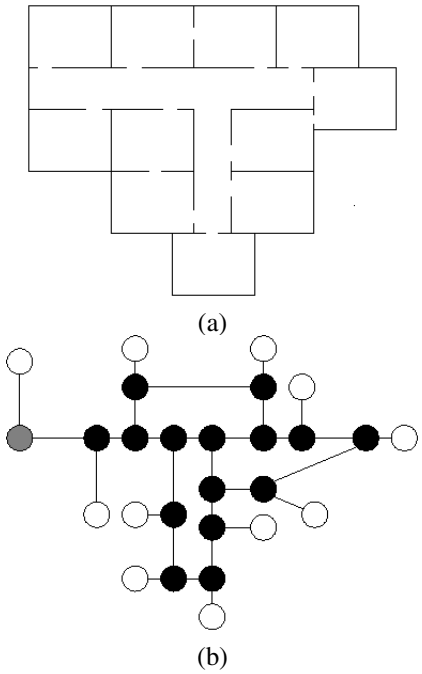


Fig. 1. Creation of a graph (b) from the given map. (a) Dark circles represent intersections, white circles represent rooms, and the gray circle represents a feature.

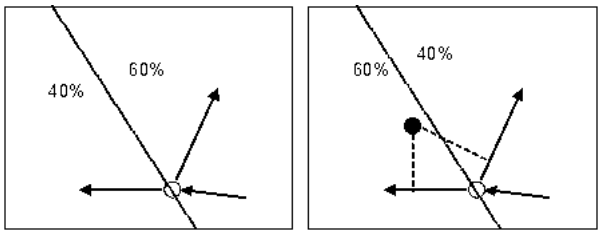
Scenarios consisting of competitive agents in multiple robot domains has recently gained prominence. These problems are very different from traditional multi-robot scenarios and are sufficiently complex to permit demonstration of the described method.

Interior urban scenarios consist of buildings with corridors and rooms. The maps are represented using a graph where edges are straight lines (forming corridors) and nodes represent either a room, an intersection, or a *feature*. A feature can be a bend or a corner in the corridors, or another recurring object, like a door. The edges are weighed by the time taken to traverse them (both length and width). Rooms are weighed by the time required to search them, dependent on factors such as size, the number of obstacles in the room, etc.

Most urban interiors can be represented using such a graph. Fig 1(a) shows an example of a typical urban building consisting of rooms and corridors. Fig 1(b) shows the map represented as a graph. This technique is also used to represent other types of terrain [6] and is often termed a "visibility graph". It is a very effective representation for navigation and path-planning problems. The robots have an empty graph representation at the start of the task execution.

The robot team consists of faster but vulnerable *scout* robots and slower but more powerful *weapon* robots, thus introducing heterogeneity. An individual robot is a mobile robot with localization and communication capabilities. The architecture is fundamentally reactive, but the individual actions include communication and storing the visited edges and nodes.

The stationary hostile agents are placed in the map with their number and positions unknown to the robots. It is assumed that



(a) (b)

Fig. 2. (a) The CovA factor. (b) The GovC factor.

these hostile agents are strictly placed at a feature, for example behind a door. Each hostile agent can only be destroyed by a minimum of *m* weapon robots. If fewer than *m* weapon robots encounter a hostile agent, the team will be destroyed.

The control of the team is very decentralized. There is no communication between the team's robots. The team's state is shared by all team members. When the team approaches an intersection, they split with some following one edge while the others follow another.

The probability that a robot will follow a particular edge j (p_j) depends on the edge's covered area ($CovA_j$), the goal covered ($GolC_j$), and the *coverage area probability* (CA). The *covered area* denotes the area accessible (in a strict geometric sense) compared to the covered area of other edges. It is expressed as a ratio of the total area (see Fig 2(a)). The $CovA$ factor splits the area into two regions according to the angle bisector of the intersection. The *goal covered* denotes the probability that the team's goal lies on this edge. The dark circle in Fig 2(b) represents the goal. The $GovC$ factor shows the distance of the goal from an edge is inversely proportional to the contribution of that edge. Therefore, the $GovC$ of each edge is approximately opposite of the $CovA$ value in Fig 2(a). The *coverage area probability* represents the priority given to the *covered area* over the *goal covered*. Given these three parameters, the probability that a robot will choose path j is:

$$p_j \;=\; CA \times CovA_j + (1 - CA)(GolC_j)$$

This probability is such that $\sum_j p_j = 1$ for each intersection, since $\sum_j CovA_j = 1$ and $\sum_j GovC_j = 1$.

Each scout robot either moves at the slower speed of the weapon robots, or moves at its own speed. This value is represented by a boolean variable, *KeepWeapon*. The robots can stop at anytime during task execution, which is represented by another boolean variable, *Move*.

There are three danger levels. The default is the zero danger level. When a team stops near a feature, the danger level of nearby robots is set to one, denoting that the team suspects there may be a enemy agent at that feature. If any team is attacked, a danger level is set to two, indicating the hostile agent's position which becomes the other robots' *goal*.

Each task state and action is represented as a bit vector. The state corresponds to team conditions and position, as in [1]. The actions are represented as bits that determine the robot's actions. The complete list of states (input to the learning module) and actions (output from the learning module) are shown in Table I. There are 90 input combinations and 44 output conditions resulting in a total of 3960 possible combinations.

TABLE I
INPUTS AND OUTPUTS FOR TASK T

Inputs		**Range**	
DL	Danger Level	0, 1, 2	
GL	Goal	0/1	0 when DL = 0
nWPN	Number of Weapons	0, 1, 2	0 - No weapons 1 - between 1 and m-1 weapons 2 - $\geq$ m weapons
FTR	Feature	1,..,5	1 - when at no feature
Outputs			
MV	Move/Stop	0/1	
WPN	Accept/Deny Weapon	0/1	
CA	Coverage Area Coefficient	0-1.0	At intervals of 0.1

TABLE II
INPUTS AND OUTPUTS FOR TASK V

Inputs		**Range**	
DL	Danger Level	0	
GL	Goal	0	0 when DL = 0
nWPN	Number of Weapons	0, 1, 2	0 - No weapons 1 - between 1 and m-1 weapons 2 - $\geq$ m weapons
FTR	Feature	1,..,5	1 - when at no feature
Outputs			
MV	Move/Stop	0/1	
WPN	Accept/Deny Weapon	0/1	
CA	Coverage Area Coefficient	0-1.0	

V. TASK SIMPLIFICATION AND LEARNING

The UIU hostile agent scenario was selected based on task complexity and the training cost. If the robots were directly trained on this task, they would be frequently destroyed before learning to avoid the destruction. The training time has to be minimized.

The task is "simplified" by training on maps with all enemy agents removed. All inputs and outputs dependent on the hostile agents are removed. Removal of each single variable reduces the total policy size. The abstracted set of inputs and outputs is provided in Table II. The number of possible input combinations is reduced to 15, though the number of output combinations remains 44, thus reducing the total number of possible combinations to 660. This task represents mapping and is a safe problem in which no robots are lost.

TABLE III

REINFORCEMENT SIGNALS FOR THE TASKS

Signal	Type	Description
Basic		
- Inapplicable action	Strongly Negative	Never choose an action not possible for a state
Simplified Task, V		
- if MV = 0 - Time Taken	Negative Positive	If robot does not move Global Reinforcement
Final Task, T		
- if MV = 0 & WPN $\geq$ 2	Negative	Never stop when enough weapon robots
- if MV = 0 & WPN = 0 FTR = EnemyFTR	Positive	Stopped near a correct feature, suspected enemy
- if MV = 0 & FTR $\neq$ EnemyFTR	Negative	Suspected enemy at a different feature

Instead of choosing a complicated teaching method such as neural networks or genetic algorithms, the chosen technique is reinforcement learning. This method consists of explicitly storing the policy in memory, i.e. there exist weight values for each possible input and output combination. This technique was effectively implemented in [1] and is very effective for systems with a small set of inputs and outputs.

Reinforcement learning faces a significant problem of propagating credit to different states and actions for even slightly complex problems. This problem is handled in two ways. First, instantaneous reinforcement signals are included for actions that are definitely negative, such as not moving at any point during mapping. Second, the robots are only allowed to select one output combination per trial. Thus the outputs for every iteration were determined independent of the inputs. The total mapping time was used as the reinforcement signal.

This method ensures that all inputs (for each robot) have all output combinations evaluated and reinforced. Even though it eliminates the reinforcement signal problems, it cannot be used for tightly coupled tasks (like the final task T). Nonetheless, it provides a fairly efficient policy for the simple mapping task.

The learning module extension for training on the final UIU scenario required each input and output that was removed to be returned. As the inputs are inserted, the weight values were copied, assuming that the inputs are independent of one another. For example, when DL = 1 and DL = 2 are inserted, their combinations with the rest of the variables (inputs and outputs) is equal to that of DL = 0. Prior knowledge can be used to modify the weights when the outputs are added, exploiting the previously known qualitative relationship between the inputs and outputs. It is not necessary to do this, and we do not assume such prior knowledge.

This extended learning module is suitable for training in scenarios containing hostile agents. Since the module has already learned a portion of the final policy, it should converge to an optimal policy faster than other methods.

Table III provides the reinforcement signals. The positive

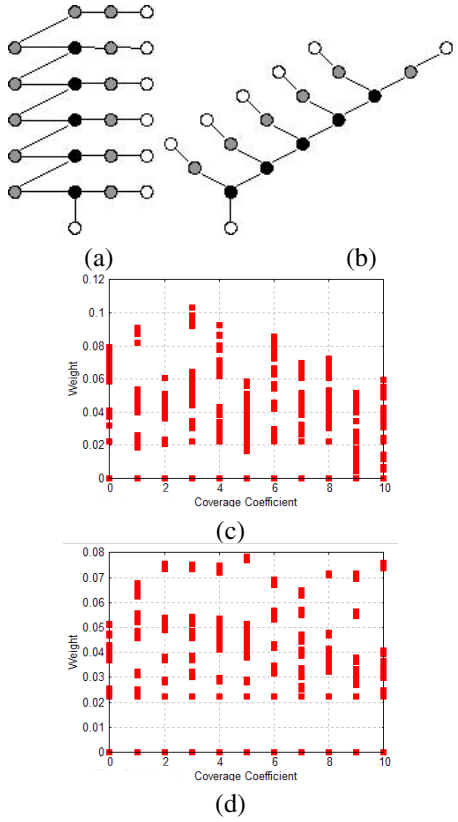


Fig. 3. Demonstrates map learning on different graphs. (c) The Coverage Coefficient graph with the weight values learned for map (a). (d) The weight values learned corresponding to map (b) *(Intersections black, features gray, and rooms white. Start point - center bottom room).*

signals were +0.2 and +0.1 for the mapping (V) and final task (T) respectively, while the negative signals were -0.2 and -0.1. The strongly negative signals were -0.5. The task completion time was normalized and inverted before applying it as a signal.

Three different methods can be used to training. First, for a given state, the action with the highest weight is chosen. This represents an *exploitation* search, where the system primarily relies on previously learned weights [1]. The second method represents the *exploration* method where the actions are chosen with a probability distribution proportional to the weight values. The third method, *pseudo-random sampling*, chooses the highest weight output with the probability q, and samples the actions according to their weight with probability $(1-q)$. If $q = 1.0$, this method corresponds to the exploitation method, and when $q = 0.0$ it corresponds to the *exploration* method. This creates a balance between the two methods.

VI. EXPERIMENTAL RESULTS

All experiments were conducted in simulation on a Pentium Celeron 2.8 Ghz machine, with 192 MB RAM. The simulation time depended on the map size and the complexity, but no runs required more than 5-6 seconds to traverse the entire map, except for cases where the robots could not complete the task.

For all experiments, m (minimum weapons needed to destroy an agent) was set to 2.

Map learning required testing all possible output combinations. Each of the 44 combinations required the module to be trained on 5 runs, resulting in a total of 220 map traversals. After learning, the model was applied to the map by choosing the maximum weight action. The time to search the entire map was very small.

Fig 3 demonstrates the weight changes for different *Coverage Coefficient* values. When trained on Fig 3(a), the *Coverage Coefficient* weights followed a downward trend (Fig 3(c)) indicating a low *Coverage Coefficient* results in better performance. Such a trend was not found for the map in Fig 3(b), where the weights were inconclusive, as shown in Fig 3(d). Thus, the coverage area is not a parameter independent of area, and thus needs to be learned.

Model extension required adding new input states (the outputs remained the same). The actions were randomly chosen with a distribution proportional to their weights. Though choosing *exploitation* might have been a safer option *per iteration*, convergence would have required a large number of iterations.

Exploration results in a more dangerous search, but has a higher probability of reaching the optimal policy faster. This search was applied to various maps with 21 intersections, 25 corridors and 6 rooms. The team consisted of 10 *scout* robots (velocity 1.5) and 5 *weapon* robots (velocity 1.0). The maximum time per trial was 600 steps. Fig 4 compares the learning after simplification with a direct application of learning on one of the maps. Initial weights of the direct application were set equal to one another. All other parameters were identicle. Indirect learning converges faster and to a value of time lower than that achieved with direct learning (Fig 4(a)). The spikes indicate cases where the robots were unable to search the map in a reasonable time. Direct learning has many more spikes than indirect learning and does not converge even after 100 trials. The results were similar in the other maps.

Two other comparisons better confirm the indirect learning performance. Fig 4(b) shows the number of robots left when training was completed. The number of robots lost with indirect learning is much smaller, and the robots quickly learn to avoid destruction. The number of safe robots in direct learning is equal to the initial number of robots (15) in only 11 of 100 trials. Indirect learning resulted in a policy that encouraged team formation. The number of teams remaining after indirect learning was lower than that with direct learning for the same number of robots (Fig 4(c)).

A portion of the final policy learned after the complex task (T) training is provided in Fig 5. The y axis is the feature faced by the robot, where 1 refers to none and 4 to an enemy feature. High weights for moving when faced with no feature, and not moving when faced with the enemy feature can be observed. Also, weights are slightly higher for moving when facing an enemy when the team has m weapon robots, since it need not stop at an enemy feature. This plot demonstrates how the decision *Move* depends on the input feature.

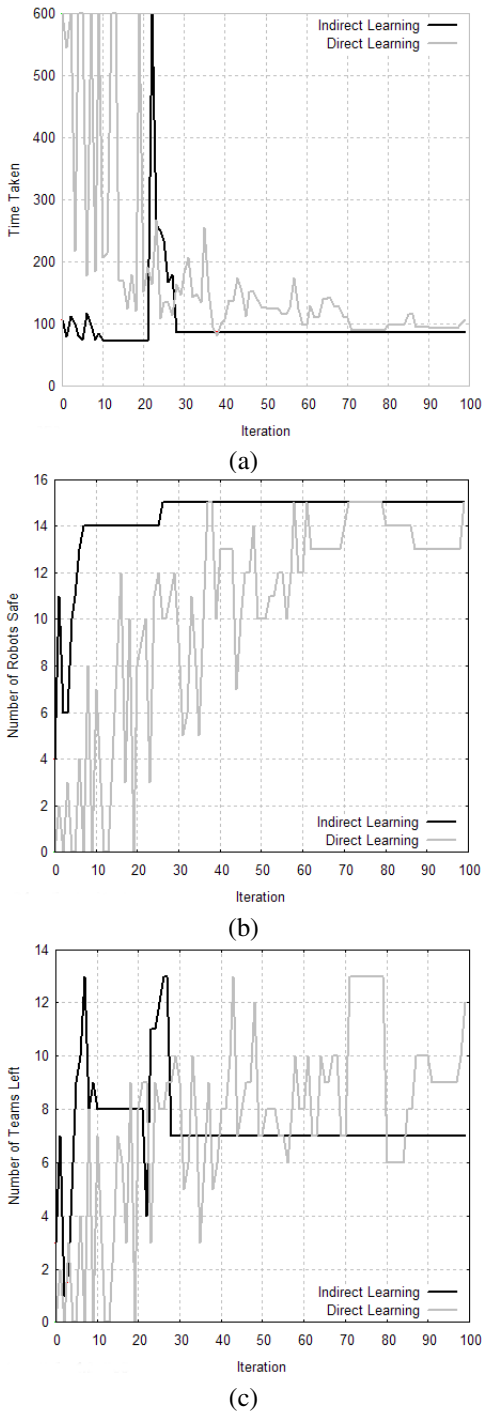


Fig. 4. Comparison of the direct and indirect learning. (a) Training time required. (b) The number of robots left after training. (c) The number of teams left after training.

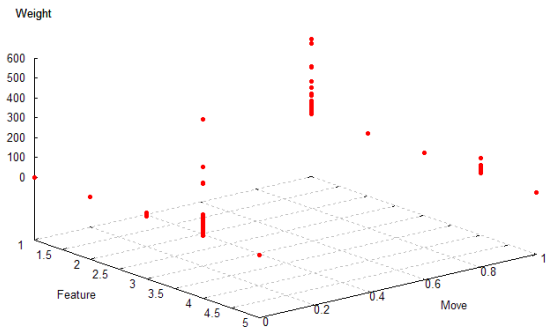


Fig. 5. 3D plot comparing feature weight values and the *Move* decision.

VII. Discussion

The experimental section demonstrates that this method performs better than direct learning. The time required for learning to perform and for performing the task were much lower. The number of robots lost (Fig 4(b)) was much lower and the generated policy was optimal.

This result is not a surprise. If we examine the method, it attempts to remove the redundant correlations at a lower level. For example, consider a scenario where a pair of input/output variables is independent of one another. In other words, for different input values, the distribution of the output weight values inherently remains the same. Suppose one of these input values belongs to the "unsafe" domain and at least one to the "safe" domain. Directly learning the policy requires learning the same weight distribution for each input value, resulting in redundancy. The presented method learns the weight values in the "safe" domain and then copies the values to other ("unsafe") input values thus reducing the amount of learning.

If there is a strong correlation between the variables, either method will spend time learning it. The proposed method uses prior knowledge of any such correlation to change the weights when extending the learning module to the final task. For example, if we know that the "unsafe" value requires the direct opposite of the "safe" value actions, the signs of the weight values can be inverted instead of merely copying them.

Another benefit of the proposed method is the number of reinforcement signals. Creating a set of reinforcement signals for the simplified task is easier since there are fewer states and actions. When extending the method to the final set of states and actions, the included reinforcement needs only be those that *correct* any wrong assumptions made during the extension (inter-independence of variable values). On the other hand, directly learning the task requires an in-depth study of the complicated problem to determine all the reinforcement signals that will effectively converge to the policy.

The final task of clearing a UIU scenario of hostile agents was also fairly complicated and interesting. Since there was minimal communication between the robots, individual robot behaviors were all the learning module could affect. Even this minimal control lead to a seemingly coordinated behavior, hinting at a possible method to team multiple agents without necessarily communicating more than necessary. This is similar to a human team behavior that does not always need a leader for every team and does not require continuous communication of individual position to other team members. The behavior is highly reactive, with the results of actions being reinforced and used for continual learning.

VIII. Conclusion

Directly learning a complicated, unsafe task is a difficult problem. Each iteration is very costly in terms of time and objects, therefore as much learning as possible without unnecessarily using additional trials is required. This paper presented a method to simplify the task and make it safe.

Training in this safe domain is cheap, and thus, many input and output value correlations can be learned. Extending the learned distributions to the remaining states and actions can be achieved either by merely copying the weights or by modifying the weights based on some prior knowledge.

The training continues to the extended module with a new set of reinforcement signals that converge on the optimal policy using fewer *unsafe* iterations. Assigning credit using reinforcement signals is much simpler with this method.

This learning method was applied to a multi-robot problem in which a team of *scout* and *weapon* robots clear an interior urban terrain of hostile agents. The simplified problem was that of mapping an area that does not contain any hostile agents. The method performance was significantly better than direct learning of the complicated task.

Work is needed to demonstrate the method on other scenarios. A number of different tasks have to be investigated to identify the difficulty of creating the simpler, safer task, which cannot be intuitively realized for complex, tightly-coupled problems. Learning techniques other than reinforcement learning also need to be evaluated since a tabular format of storing the policy becomes impractical for complex problems.

References

[1] M. Matarić, "Reinforcement learning in the multi-robot domain," *Autonomous Robots*, vol. 4, no. 1, pp. 73–83, 1997.

[2] L. E. Parker, "Task-oriented multi-robot learning in behavior-based systems," in *IEEE Intl. Conf. on Intelligent Robots and Systems*, 1996, pp. 1478–1487.

[3] F. Davesne and C. Barret, "Reactive navigation of a mobile robot using a hierarchical set of learning agents," in *IEEE Intl. Conf on Intelligent Robots and Systems*, 1999, pp. 482–487.

[4] O. Fuentes, R.P.N. Rao, and M. Van Wie, "Hierarchical learning of reactive behaviors in an autonomous mobile robot," in *IEEE Intl. Conf on Systems, Man and Cybernetics*, 1995, pp. 4691–4695.

[5] R. Malak and P. Khosla, "A framework for the adaptive transfer of robot skill knowledge using reinforcement learning agents," in *IEEE Intl. Conf on Robotics and Automation*, 2001, pp. 1994–2001.

[6] H. Santana, V. Corruble, and B. Ratitch, "Multi-agent patrolling with reinforcement learning," in *Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems*, 2004, pp. 1122–1129.

[7] I. Rekleitis, R. Sim, G. Dudek, and E. Milios, "Collaborative exploration for map construction," in *IEEE Intl. Symp. on Computational Intelligence in Robotics and Automation*, 2001, pp. 296–301.

[8] M. Quinn, "Evolving co-operative homogeneous multi-robot teams," in *IEEE Intl. Conf. on Intelligent Robots and Systems*, 2000, pp. 1798–1803.

[9] F. Wang and E. Mckenzie, "Multifunctional learning of a multiagent based evolutionary artificial neural network with lifetime learning," in *IEEE Intl. Symp. on Computational Intelligence in Robotics and Automation*, 1999, pp. 332–337.