Learning Mixed Behaviours with Parallel Q-Learning

Guillaume J. Laurent, Emmanuel Piat

Laboratoire d'Automatique de Besançon - UMR CNRS 6596
25, rue Alain Savary, 25000 Besançon, France
E-mail: epiat@ens2m.fr, website: http://www.lab.ens2m.fr

Abstract

This paper presents a reinforcement learning algorithm based on a parallel approach of the Watkins's Q-Learning. This algorithm is used to control a two axis micro-manipulator system. The aim is to learn complex behaviours as reaching target positions and avoiding obstacles at the same time. The simulations and the tests with the real manipulator show that this algorithm is able to learn simultaneously opposite behaviours and that it generates interesting action policies with regard to the global path optimization.

1 Introduction

Our objective is to automate the control of a micro-manipulator system called WIMS (WIreless Micro-manipulation System) which is now being developed in our lab [1]. This two degrees of freedom micro-manipulator system will be able to push micro-objects like biological cells (about 10 μm) between two glass slides under a microscope. The actuation principle uses a magnetic field to indirectly move a micro-tool in the biological solution (cf. figure 1).

In this paper, we focus on the control of this manipulator. The purpose is to design a controller able to move the tool between many objects without hitting them. More precisely, the tool has to reach a target position defined by a human operator, while avoiding objects considered as obstacles.

The number and the position of the obstacles are not defined and can evolve with time. In the same way, there may be several targets and their positions are not fixed either. Moreover, the behaviours of the micro-tool and of the micro-objects are difficult to predict because of adhesive forces, frictions, hysteresis... The modeling of these particular phenomena due to the microscopic scale is difficult. For these reasons, we chose an approach using learning features. The algorithms of reinforcement learning introduced in the 80's by Sutton [2] and Watkins [3] offer interesting characteristics for the control of our manipulator system. Reinforcement learning provides a way of programming by rewards and punishments without needing to specify how the task is to be achieved. It was demonstrated that some algorithms like the Watkins's Q-Learning converge towards the optimal control policy with some conditions [4].

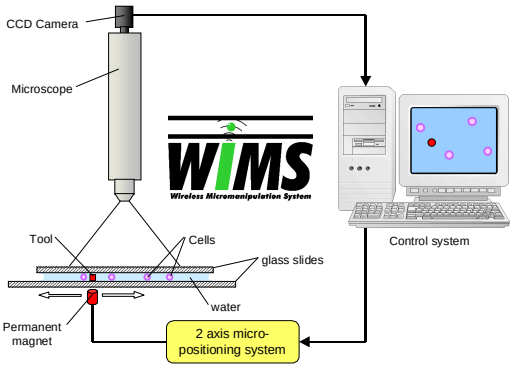

Figure 1: *The WIreless Micromanipulation System (WIMS).*

Nevertheless, these algorithms are rather adapted to the control of simple systems. To learn more complex behaviours, hierarchical algorithms were developed. But these algorithms require to define either a hierarchical fixed architecture (Gated behaviour [5] [6] [7], Feudal Q-Learning [8]) or a division of the problem into sub-objectives (Compositional Q-Learning [9]). Furthermore, these algorithms can't adapt themselves to a changing number of targets or obstacles.

Our own approach was presented in a previous paper [10]: it is a new learning algorithm called parallel Q-Learning. This algorithm has two advantages. It adapts itself immediately to any situation whatever the number of targets may be. Moreover, the higher the number of targets is, the faster it learns. So, it is well adapted to reach many targets which number and position may change. But this algorithm is limited because it does not allow to mix opposite behaviours as targets searching and obstacles avoiding.

To allow to learn these more complex behaviours, we enriched two of the functions of the previous algorithm.

This paper contains five parts. The both first are dedicated to the descriptions of Q-Learning and of the architecture of our controller. The third presents the manipulation system and the frame of the tests. The performances of the algorithm are described in the last two parts. The simple case where all the objects are the same, the problem of opposite behaviours and the experimental results obtained with the real manipulation system are presented in these parts.

2 Q-Learning

Q-Learning is a reinforcement learning algorithm introduced in 1989 by Watkins [3]. The principle of the reinforcement learning is based on learning by trials and errors. On each step of interaction, the controller receives an input that provides indications of the current state X_t of the system. Then, the controller chooses an action u_t. This action changes the state of the system and the controller receives a reward r_t according to the new state X_{t+1}. The controller's job is to find a policy, mapping actions to states, that maximizes the long-run sum of rewards.

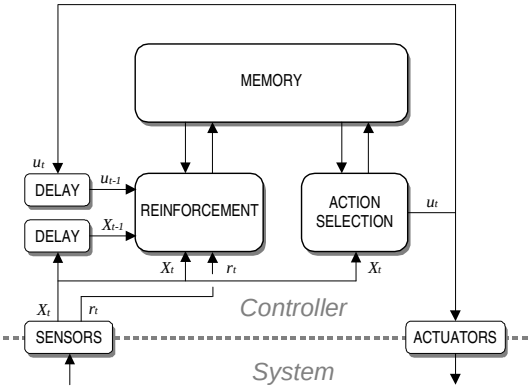

Figure 2: *Q-Learning operating cycle.*

The Q-Learning operating cycle is illustrated by the figure 2. The memory stores an estimation of the expected return[1] in a real value $q(X, u)$ called action-value for every observed state-action couple (X, u).

On the left, the reinforcement function updates the action-value function q on each step of interaction, with the following equation:

$$q(X_{t-1}, u_{t-1}) \leftarrow q(X_{t-1}, u_{t-1}) + \\ +\alpha[r_t + \gamma \max_v q(X_t, v) - q(X_{t-1}, u_{t-1})] \quad (1)$$

On the right, the action is selected by two stages. First, the expected return of every action is estimated using the action-value $q(X, u)$. Then, a policy chooses the action according to an exploration / exploitation criteria. In this paper, we use the ϵ-greedy[2] policy because of its simplicity. Other policies could be used as well. ϵ is set to 0.1

3 Our approach

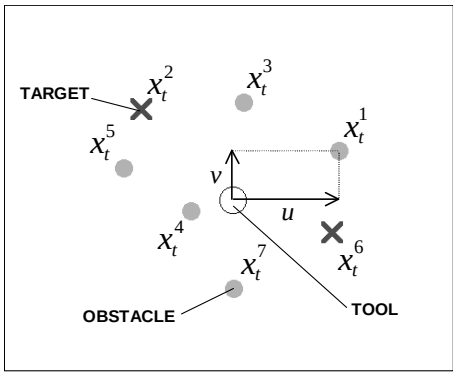

Figure 3: *State example.*

Our approach takes place in a particular case. The controller receives on each step of interaction a state X_t made of a *changeable* number N_t of elementary perceptions x_t^i. All these elementary perceptions x_t^i belong to the same vectorial space $\mathcal{X}$.

$$X_t = \{x_t^i \,|\, 1 \leq i \leq N_t,\ x_t^i \in \mathcal{X}\} \quad (2)$$

In our application, X_t contains the set of targets and obstacles. In the example shown by the figure 3, the state is:

$$X_t = \{x_t^1, x_t^2, x_t^3, x_t^4, x_t^5, x_t^6\} \quad (3)$$

Every target or obstacle constitutes an elementary perception x_t^i. These elementary perceptions are defined by their type (target or obstacle) and their coordinates in the reference mark bound to the tool (cf. figure 3), i.e.:

$$x_t^i = (type, u, v) \quad (4)$$

[1] summation of future expected rewards for a given control policy

[2] With a small probability ϵ, a random action is selected independently of the action-value $q(X_t, u)$, otherwise, the selected action is the one for which $q(X_t, u)$ is maximum

The elementary hypothesis of our approach is to assume that the perceptions of X_t are similar and independent each other. Intuitively, the presence of one or several targets doesn't change their own behaviour. It means that a perception has a single behaviour. So, the controller has to learn how a perception behaves to be able to predict the behaviour of all states made of the entire set of perceptions.

In this way, the algorithm proceeds to a learning like Q-Learning for every perception but always using the same action-value function q. So, the action-value function $q(x_t^i, u)$ associates an estimation of the expected rewards to every perception x_t^i and not to a state X_t. Then, the action selection function allows to generalize this learning to a complete state.

3.1 Reinforcement function

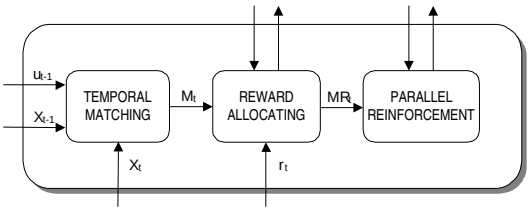

Figure 4: *The new reinforcement function.*

The aim of this function is to update the memory with the equation 1 using every perception as if it were the state of the system. To do that, the evolution of every perception must be reconstructed. But, in the two states X_{t-1} and X_t, the elementary perceptions are not ordered, x_t^i is not the temporal successor of x_{t-1}^i. So, we must create a tool to match in twos the perceptions of the states X_{t-1} and X_t. On the other hand, in the classic frame of reinforcement learning, the reward signal is a scalar. In our case, we assume that perceptions are independent. So, a reward can be connected with a single perception. The question is to know which is this perception.

To solve these problems, the reinforcement box is enriched by new functions (cf. figure 4). A temporal matching function connects the perceptions of X_{t-1} with their temporal successor of X_t. A reward allocating function determines the perception connected with the last obtained reward. Finally, a parallel reinforcement function updates the memory with all collected data .

Temporal matching function. The purpose of the matching function is to match in twos the perceptions of the states X_{t-1} and X_t according to a defined criterion. In practice, this function has to reconstruct the spatial evolution of every perception. For example, it has to reconstruct the motion of every target when the robot moves.

The output of this function is a set M_t of triplets called transitions. Every transition is made of an elementary perception of X_{t-1}, of its supposed successor belonging to X_t and of the last action u_{t-1}. If f is the matching function, M_t can be written as:

$$M_t = \{(x, u_{t-1}, y) \,|\, x \in X_{t-1},\, y \in X_t,\, y = f(x)\} \tag{5}$$

It remains to choose f in the set $\mathcal{F}$ of the functions mapping X_{t-1} to X_t. In our case, we chose the criterion of the minimization of the sum of distances between two elementary perceptions. So, f must verify, for any function g in $\mathcal{F}$:

$$\sum_{x \in X_{t-1}} \|x - f(x)\| \leq \sum_{x \in X_{t-1}, g \in \mathcal{F}} \|x - g(x)\| \tag{6}$$

To obtain a good solution in a weak calculation time, we opted to use a heuristic method. Perceptions are matched with their closest neighbour in a random order. This heuristic method gives good results if the perceptions are far enough from each other (like in our application).

Finally, if a new object enters or goes out of the field of vision, the states X_{t-1} and X_t won't have the same number of perceptions ($N_{t-1} \neq N_t$). In that case, these new or last perceptions are not used by the matching function during one step of interaction.

Reward allocating function. The aim of this function is to find the elementary perception connected with the last obtained reward.

For example, if the robot reaches a target, the reward is equal to 1. The perception of the target which caused this reward is the one which is at the same position as the robot. But this information comes from our global knowledge of the system. The controller has to learn this link. It must find the perception which caused the reward with the only information it has.

Just before reaching a target, every perception x of the state X_t of the robot predicted a value of the expected return given by the action-value function $q(x, u)$. The perception of this target had to predict 1 if the controller already met this situation. The other targets which are not reachable in one step probably predicted values lower than 1. So, the perception connected with the reward 1 is the one which predicted this reward at best.

So, our method allocates the reward to the perception of the previous state which action-value is the closest to this reward. The others get zero. This algorithm means that only one elementary perception is connected with the reward.

If the reward is positive, the perception x^* which receives this reward is the one for which $q(x,u)$ is maximum for all the triplets (x,u,x') in M_t, i.e.:

$$x^* \leftarrow \underset{\forall x,(x,u,x')\in M_t}{\arg\max} \; q(x,u) \tag{7}$$

If the reward is negative, we have to find the perception which predicted the worst expected return. This worst expected return is not estimated by q. So, we use a new function called w. $w(x,u)$ represents an estimation of the worst expected return for the perception x taking the action u. w is updated in the same way as q but by using the operator *min* rather than *max* (see next section).

The perception which receives the negative reward is the one for which $w(x,u)$ is minimum, i.e.:

$$x^* \leftarrow \underset{\forall x,(x,u,x')\in M_t}{\arg\min} \; w(x,u) \tag{8}$$

The output of this function is a set called MR_t of transitions with rewards. These transitions with rewards are made of the three members of a transition and of the reward allocated to this transition.

Parallel reinforcement function. The parallel reinforcement function updates the memory with the information supplied by the reward allocating function in the following way:

For all $(x,u,x',r) \in MR_t$ do
$q(x,u) \leftarrow q(x,u) + \alpha[r + \gamma \max_v q(x,v) - q(x,u)]$
$w(x,u) \leftarrow w(x,u) + \alpha[r + \gamma \min_v w(x,v) - w(x,u)]$

3.2 Action selection function

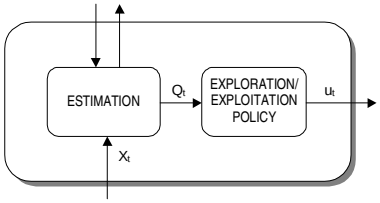

Figure 5: *The new action selection function.*

This box selects the action which is sent to the actuators. It contains two parts (cf. figure 5). The left part calculates for every action u of $\mathcal{U}$ an estimation of its expected return $Q_t(u)$ for the current state X_t. In fact, the estimation function is a fusion operator of the expected returns generated by every elementary perception [10]. The right function uses the Q_t values and a exploration / exploitation policy in the same way as Q-Learning.

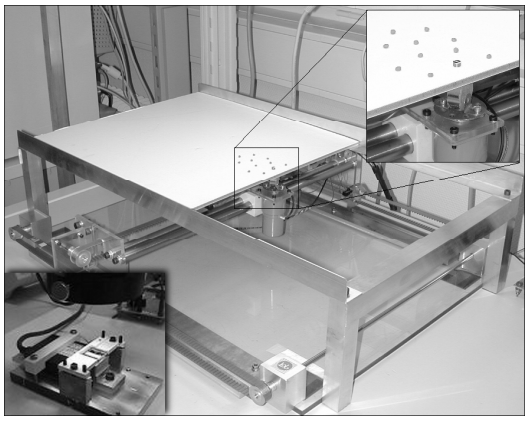

Figure 6: *The manipulator system and the WIMS (on the left).*

In [10], we presented the results obtained with the *max* fusion operator . Although successful, this operator does not allow to mix opposite behaviours because negative expected returns are forget. For this reason, we chose the *sum* operator. The expected return of every action is the sum of the expected returns of every elementary perception, i.e.:

$$\forall u,\, u \in \mathcal{U},\, Q_t = \sum_{x \in X_t} q(x,u) \tag{9}$$

We will see farther that this operator produces better optimization results.

4 Test-bed

The manipulator system we used for the experiments of this article is able to push millimeter size objects. It is only a test-bed for our controller. Next, our results will be adapted to the future WIMS.

This manipulator system is made of a ferro-magnetic tool moved by a permanent magnet set under the glass slide (cf. figure 1). The tool is a 5×5 mm steel cylinder. The magnet and the tool have two degrees of freedom: up/down and left/right. The magnet is moved by two electric motors. The system is very difficult to drive: great hysteresis between the magnet and the tool, non-linearity,... Above the manipulation area, a video CCD camera allows to calculate the position of every object and to generate the state X_t of the system. The figure 6 shows a photo of the real manipulator system and the WIMS.

Aim. The controller has to learn to move the tool towards a target and to avoid obstacles at the same time. Targets are virtual objects placed by a human operator according to the wanted task. Obstacles are 2 mm plastic balls set on the glass slide. The perception space $\mathcal{X}$ is divided into 47×35 boxes for

every kind of object, that is to say 3290 different perceptions.

Four discrete actions generate a displacement of the tool: up, down, right, left. If the tool reaches a target, the controller receives a positive reward equal to 1. Then, the target disappears. If the tool touches an obstacle, the controller receives a negative reward equal to -1. The obstacles are motionless and fixed.

To test the performances of the learning algorithm, we use a simulation of this system.

5 Learning to mix similar behaviours

In this part, we study the behaviour of the algorithm with states made of similar perceptions (i.e. without obstacles)

5.1 Previous work

The preliminary stage consists in determining the optimal values of the learning parameters (see [10]). Using the simulation, we obtain $\alpha = 0.5$, $\gamma = 0.5$.

The analysis of the learning convergence time is interesting. The conclusion was already presented in the previous article. It is about the influence of the number of targets on the learning speed. The larger the number of targets is, the faster the learning is (the learning is ten times faster with ten targets with regard to one target).

5.2 Resulting paths

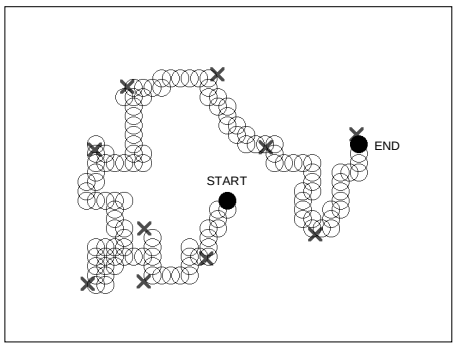

Figure 7: *Path of the tool with ten targets.*

The figure 7 shows the path in the simulated system after a sufficient learning time. The analysis of this path is interesting in two levels. First, the actions done to reach a target are almost optimal. The controller found the optimal policy with the exception of exploration action (about one action for ten actions because $\epsilon = 0.1$). This result normally reached by the Q-Learning algorithm was not changed with the parallel process. The second point is even more interesting. From a more global point of view, the general path is optimized. The controller always begins to reach targets belonging to the densest zone. Then, it goes for a collecting tour exactly like an optimal travelling salesman.

6 Learning to mix opposite behaviours

In this part, we study the behaviour of the algorithm when there are obstacles.

6.1 Learning speed

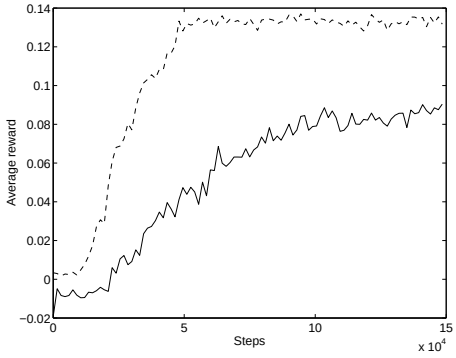

Figure 8: *Average reward with ten targets without obstacles (broken line) and with five obstacles (continuous line) according to the number of algorithm steps.*

The figure 8 compares the performances with obstacles and without obstacles. If there are obstacles, the controller doesn't get as much positive rewards. Indeed, when the tool by-passes an obstacle, it has to do a longer way to reach the target. This longer way causes a fall in the received average rewards. Furthermore, at the beginning of the learning, the received rewards are often negative because the controller has not learnt to avoid obstacles yet. Finally, the learning time is longer because the algorithm has to learn to sort the targets and the obstacles. So, the reward allocating function has to learn to differentiate the obstacles from the targets.

On the other hand, if the obstacles are introduced after the controller learnt to reach targets, the results are different. The figure 9 shows the results when obstacles are added during the test. The loss of efficiency is smaller. The controller receives more rewards than in the previous test. Furthermore, the learning of the obstacle avoiding behaviour is almost immediate.

These results show that the way of learning is important to speed the learning and to reach better performances. This gradual learning method helps the controller to sort the obstacles and the targets.

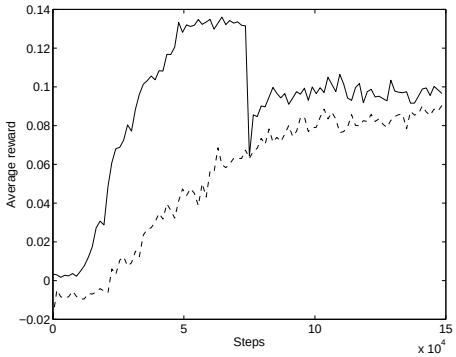

Figure 9: *Average reward when the five obstacles are added during the test (continuous line) with regard to the previous test (broken line).*

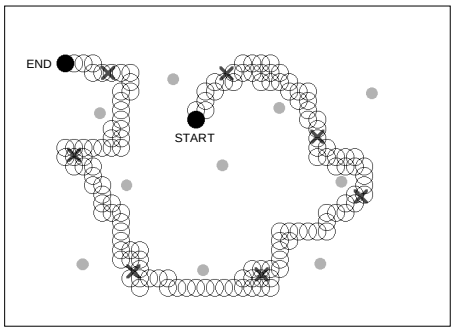

Figure 10: *Path of the tool with seven targets and ten obstacles.*

6.2 Resulting paths

The figure 10 shows that the controller find good paths. Indeed, the path between every target is optimized and is very close of obstacles. In the global point of view, the obstacles don't change the optimization. The mixing of behaviours is very satisfactory.

6.3 Test with the real manipulator system

After a learning process with the simulation, the controller is connected to the real manipulator system. The figure 11 shows the path of the tool. The path is more chaotic because the real system is not as perfect as the simulation. Especially, the large hysteresis between the magnet and the tool forces the controller to step aside when it changes its way.

7 Conclusion

This paper presents the parallel Q-Learning algorithm. The tests show that this algorithm is able to learn simultaneously opposite behaviours. For example, it is able to learn to search targets while avoiding obstacles. Beyond this behaviours mixing capacity,

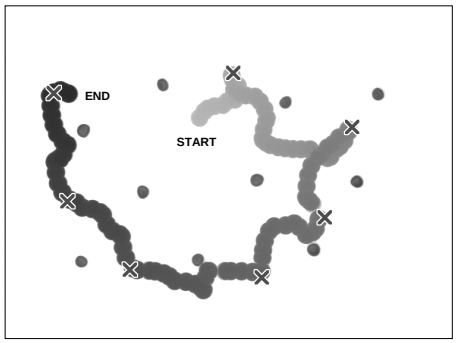

Figure 11: *Path of the real manipulator system with five targets and ten obstacles.*

the action policy of the algorithm are specially interesting with regard to the global path optimization. The controller always begins to reach targets belonging to the densest zone. Then, it goes for a collecting tour exactly like an optimal travelling salesman.

We work now to adapt this algorithm to more complex control policies like the control of systems with dynamics elements (for example a block-pushing problem with obstacles avoidance).

References

[1] M. Gauthier and E. Piat. Behavior of a magnetic manipulator of biological objects. In *International Conference on Robotics and Automation*, Washington D.C., 2002.

[2] R.S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, Amherst, MA, 1984.

[3] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, 1989.

[4] C. Watkins and Peter Dayan. Technical note : Q–learning. *Machine Learning*, 8:279–292, 1992.

[5] P. Maes and R. Brooks. Learning to coordinate behaviors. In *Eighth National Conference on Artificial Intelligence*, pages 796–802, 1990.

[6] S. Mahadevan and J. Connell. Automatic programming of behaviorbased robots using reinforcement learning. In *Ninth National Conference on Artificial Intelligence*, Anaheim, CA, 1991.

[7] L. Lin. Hierachical learning of robot skills by reinforcement. In *International Conference on Neural Networks*, 1993.

[8] P. Dayan and G.E. Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems 5*, San Mateo, CA, 1993.

[9] S.P. Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8(3):323–340, 1992.

[10] G. Laurent and E. Piat. Parallel q-learning for a block-pushing problem. In *International Conference on Intelligent Robots and Systems*, Maui, USA, 2001.