Control of planar micro-manipulator systems using reinforcement learning

Guillaume 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

This paper presents an application of reinforcement learning to the control of a planar micro-manipulator system. The manipulation station we used is able to push small objects on a glass slide under a camera. The objective is to automate complex tasks of manipulations. Our approach is based on reinforcement learning algorithm (Q-Learning) because the models of the manipulator and of the dynamics of objects are unknown. The system is too complex for a classic algorithm, so we propose an original architecture which realizes several parallel learning processes. This method produces an almost optimal policy whatever the number of manipulated objects may be. Some simulations allowed us to optimize every parameter of the learning process. The experimental tests show that the controller learns its task very quickly.

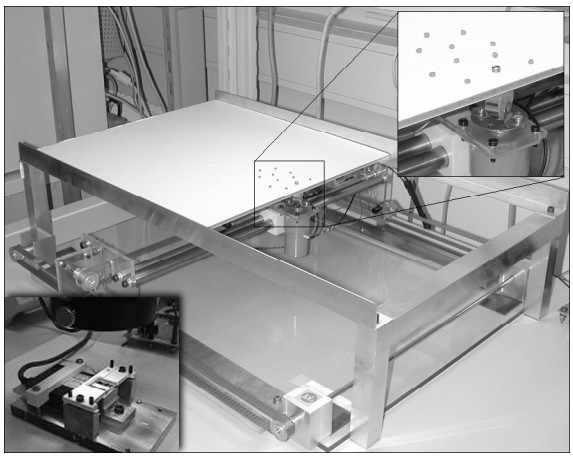

Figure 1: Micro and milli-manipulator systems.

1 Introduction

Our objective is to automate the control of a micro-manipulation station able to push micro-objects on a glass slide under a microscope. The station is constituted by a ferro-magnetic tool moved by a permanent magnet set under the glass slide (cf. figure 1 and 2). The magnet has two degrees of freedom : up/down and left/right. Objects are set on the glass slide. For this experiment, we used 3 mm plastic balls but the future application would be able to move biological cells (about 10 μm). Another research team works on the miniaturization of this manipulation station [1]. Above the manipulator system, a video camera catch the complete scene. The magnet is moved by two electric motors. The motors are open loop controlled. There isn't any system to control the position of the magnet and the system is very difficult to drive : great hysteresis between the magnet and the tool, non-linearity, ...

Our purpose is to automate a pushing task. For example, we wish the controller would be able to move objects from a point to another one by pushing them with the tool.

The dynamics of the block-pushing is not coarse and it is difficult to model it [2]. The behaviors of the manipulator system and of the micro-objects are unknown : no-linearity, hysteresis, adhesiveness, ... The modeling of these particular phenomena due to the microscopic scale is even more difficult. So, it would be interesting to use a controller which doesn't require any model. Furthermore, during a micro-manipulation process, the only available information comes from the microscope. So, we wish the controller would be able to work with those inputs. Finally, the controller must manage successive movements of objects and optimize a global function as the total time of manipulation for n objects.

The algorithms of reinforcement learning developed in the 80s by Sutton [3] and Watkins [4] offer interesting characteristics for the control of our manipulation station. First, they allow to approach optimal control without knowing the model of the system. The models of the block-pushing manipulation, of the manipulator system, and of the manipulated micro-objects don't need to be known. Reinforcement learning provides a way of programming by reward and punishment without needing to specify how the task is to be achieved. Furthermore, they guarantee the convergence towards the global optimal command in Bellman's sense. This point is particularly interesting to minimize the manipulation time.

Although reinforcement learning fits very nicely to every system, it is a slow learning process. The main reason is the size of the space of states to visit [5]. If

the dimension of the space is big, the time of learning is very important. Quickly, the learning process takes such a long time that an on-line learning is not possible.

The space of states of our application is large. In fact, every object moves in a two dimensional space. If n objects are considered, the space of states of the system has $2n$ dimensions. A learning is very difficult in a so big space. So, we propose an original architecture to reduce this complexity.

This article contains three parts. The first is dedicated to the description of the architecture of our controller. The second presents simulations made to optimize the algorithm. The last part describes experimental results obtained with the real manipulation station.

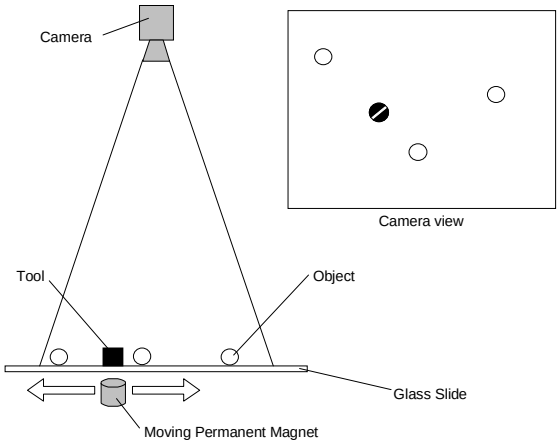

Figure 2: Working plan.

2 Architecture of the controller

2.1 Q-Learning

Q-Learning is a reinforcement learning algorithm introduced in 1989 by Watkins [4]. It is also described in the Sutton's book [6] and in the Kaebling's paper [7].

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 s of the system. Then, the controller chooses an action a. This action changes the state of the system and the controller receives a reward $r_{ss'}$ according to the new state s'. The controller's job is to find a policy π, mapping states to actions, that maximizes the long-run sum of rewards. The convergence of this algorithm towards the optimal policy π^* was proved in 1992 by Watkins [8].

The choice of an action is based on the past experience of the controller. A function $Q(s,a)$ is used to memorize the expected reward for the action a and the state s. This function Q is called the action-value function. In our case, the action-value function is a look-up table.

On each step of interaction, the action-value function is updated with equation (1) :

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r_{ss'} + \gamma \max_b Q(s',b) - Q(s,a) \right] \quad (1)$$

with :

- s the old state of the system,
- s' the new state of the system,
- a the chosen action in the state s,
- α a learning rate parameter,
- γ a discount rate parameter.

To choose an action in a state s, the controller uses the action-value estimations $Q(s,a)$ and an exploration/exploitation strategy called σ. The selected action is then :

$$a = \sigma(s, Q)$$

In our case, σ is the ϵ-greedy strategy : most of the time, the greedy action is selected (the action for which $Q(s,a)$ is maximum) and sometimes, a random action is selected with a small probability ϵ, independently of the action-value estimations. This strategy allows us to control exactly the exploration rate. To maintain a constant exploration rate, we chose $\epsilon = 0.1$ during the tests. This parameter can be lowered when learning is ended.

The policy π of the controller, mapping states to actions, is fully defined by the action-value function Q in addition with the exploration/exploitation strategy σ (i.e. $\pi(s) = \sigma(s, Q)$).

The Q-Learning algorithm suits very well to all small state spaces. If the controller had to manipulate only one object the state space would have only two dimensions, so the learning would be possible. But, the controller has to manage several objects. Every object moves in a two dimensional space. It is necessary to take into account all of possible configurations of all objects set on the glass slide. So, for N objects, the state space has $2N$ dimensions and the traditional Q-learning algorithm can't work with a so large state space.

2.2 Our approach

The elementary hypothesis of our approach is to assume that all the objects are the same. Each object moves alike on the plane defined by the glass slide. So, if it was alone, each object would generate the same action-value function q. For this reason, we use the same action-value function for all the objects. Several learning processes are done at the same time with the same look-up table q.

This method allows to reduce the complexity of the system. Furthermore, the learning process may be sped up because more backups will be done. As a general

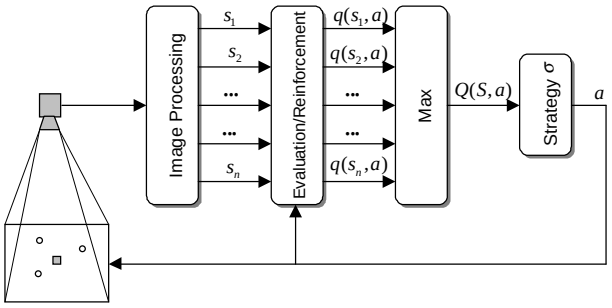

Figure 3: Architecture of the controller.

rule, our approach allows to reduce the state space complexity when several variables move in the same vectorial space. For example, in our application, every object moves in the same vectorial space defined by the glass slide plane.

Every state s_i of an object i is defined by its position in the plane. On each step of interaction, the action-value function is updated using the Q-Learning equation for every object :

$$q(s_i, a) \leftarrow q(s_i, a) + \alpha \left[r_{s_i s'_i} + \gamma \max_b q(s'_i, b) - q(s_i, a) \right]$$

Then, we define the global action-value function for the global state $S = \{s_i, \forall i\}$ of the system by :

$$Q(S, a) = \max_i q(s_i, a)$$

This point will be discussed in the next section.

The strategy σ uses the global action-value function Q to choose the action to perform.

The algorithm architecture is summarized on the figure 3. Every object is independently processed using the same function q. At the end, the local action-values are compared in order to evaluate the global action-value function. The figure 4 presents the algorithm of our controller.

Initialize the first state $S = \{s_i\}$
Repeat
  Choose a using $a \leftarrow \sigma(S, Q)$
  Take action a
  Observe the new state $S' = \{s'_i\}$
  For all object i do
    $a^* \leftarrow \arg\max_b q(s'_i, b)$
    $q(s_i, a) \leftarrow q(s_i, a) + \alpha[r_{s_i s'_i} + \gamma q(s'_i, a^*) - q(s_i, a)]$
  $S \leftarrow S'$

Figure 4: Controller algorithm.

2.3 Theoretical approach

First case, the state space is $\mathcal{S}$ and all the rewards equal zero except the reward $r > 0$ associated to the transitions leading to the state x_{goal} (cf. figure 5).

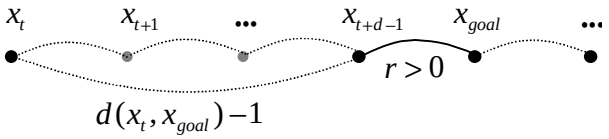

Figure 5: A path example in the state space $\mathcal{S}$.

If x_t is the state of the system at the time t, the optimal action-value function is defined by :

$$q^*(x_t, a) = \max_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{x_{t+k} x_{t+k+1}} \right]$$

where $\max_\pi$ means that the policy which maximizes the sum is chosen for the evaluation.

The system will reach the state x_{goal}, so the optimal action-value function can be written as :

$$q^*(x_t, a) = \max_\pi \left[\gamma^0 r_{x_t x_{t+1}} + \gamma^1 r_{x_{t+1} x_{t+2}} + ... + \gamma^{d(x_t, x_{goal})-1} r_{x_{t+d-1} x_{goal}} + \sum_{k=d(x_t, x_{goal})}^{\infty} \gamma^k r_{x_{t+k} x_{t+k+1}} \right]$$

We called $d(x_t, x_{goal})$ the number of transitions to go from the state x_t to the state x_{goal}. For all states except x_{goal}, the reward equals zero, so :

$$q^*(x_t, a) = \max_\pi \left[\gamma^{d(x_t, x_{goal})-1} r_{x_{t+d-1} x_{goal}} + \gamma^{d(x_t, x_{goal})} \sum_{k=0}^{\infty} \gamma^k r_{x_{t+d+k} x_{t+d+k+1}} \right]$$

Using the Bellman's principle, we get :

$$q^*(x_t, a) = \gamma^{d^*(x_t, x_{goal})-1} \left(r_{x_{t+d^*-1} x_{goal}} + \gamma \max_\pi \sum_{k=0}^{\infty} \gamma^k r_{x_{t+d^*+k} x_{t+d^*+k+1}} \right)$$

i.e. :

$$q^*(x_t, a) = \gamma^{d^*(x_t, x_{goal})-1} \big(r + \gamma q^*(x_{goal}, \pi^*(x_{goal})) \big) \quad (2)$$

where $d^*(x_t, x_{goal})$ is the minimum value of $d(x_t, x_{goal})$.

Second case, the state space is $\mathcal{S}^N$ and several goal states provide a positive reward r (cf. figure 6). The new action-value function is noted Q and a state in $\mathcal{S}^N$ can be written as :

$$X = (x_0, x_1, ..., x_n) \mid \forall i, x_i \in \mathcal{S}$$

The goal states are defined by :

$$X_{goal_i} = (x_0, x_1, ..., x_n) \mid \exists i, x_i = x_{goal}$$

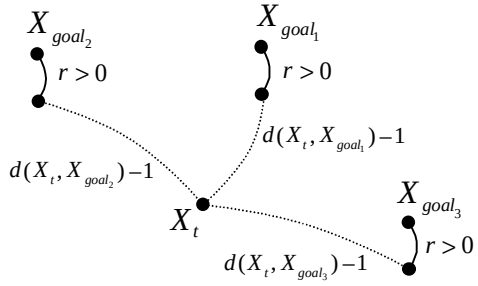

Figure 6: Example of a $\mathcal{S}^3$ state space.

In any case, the controller will reach a first state X_{goal_i}, so the optimal action-value can be written as :

$$Q^*(X_t, a) = \max_i \Big[\gamma^{d^*(X_t, X_{goal_i})-1} \big(r + \gamma Q^*(X_{goal_i}, \pi^*(X_{goal_i})) \big) \Big]$$

$Q^*(X_{goal_i}, \pi^*(X_{goal_i}))$ is difficult to estimate because it depends on each state x_i. To continue, we assume that :

$$Q^*(X_{goal_i}, \pi^*(X_{goal_i})) = q^*(x_{goal}, \pi^*(x_{goal}))$$

i.e. :

$$Q^*(X_t, a) = \max_i \Big[\gamma^{d^*(X_t, X_{goal_i})-1} \big(r + \gamma q^*(x_{goal}, \pi^*(x_{goal})) \big) \Big]$$

Using the equation (2), we get :

$$Q^*(X_t, a) = \max_i q^*(x_{i_t}, a)$$

This hypothesis is strong. The policy of a learning using this method will not converge toward the global optimal policy. The optimization will be local and calculated as for an isolated state x_i. But, if the states x_i are far away from each other, the hypothesis is nearly true.

3 Simulations

3.1 The test-bed

The objective of this simulation is to test the performances of the learning algorithm. The objects and the tool are modeled by circles (cf. figure 7). Eight actions are possible : up, down, left, right and the four diagonals which deterministically cause the tool to move in the corresponding direction. All dimensions and velocities are near the reality. The hysteresis between the magnet and the tool is modeled by a dead area. The mechanical interactions between the objects and the tool

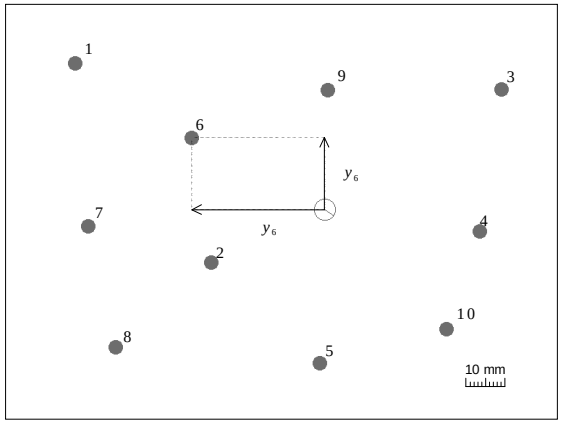

Figure 7: Snapshot of the simulation.

are also modeled : if the tool bumps into an object, it pushes it.

When an object reaches an edge of the plane, it is put back to a random position. Every state s_i of an object is defined by its coordinates (x_i, y_i) calculated with regard to the robot. This state definition is important to reduce the data to the bare necessity. Each object can stay in 110 495 different positions.

We want the manipulator system to move all the objects towards the right edge of the screen. More precisely, the tool has to successively push each object towards the right. So, the controller is rewarded when an object is moved towards the right.

3.2 Choice of the parameters

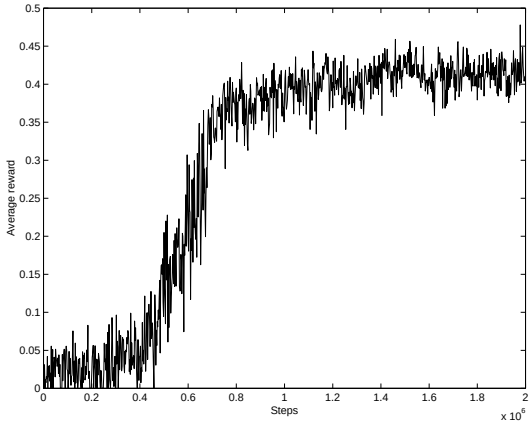

Figure 8: Average reward according to the number of algorithm steps with $\alpha = 0.1$ and $\gamma = 0.5$.

The controller and the algorithm behave very well. During the simulations, we calculate the average reward to study the behaviour of the algorithm. This average reward is the ratio between the sum of rewards and the number of algorithm steps. We obtain the characteristic graph described in the figure 8. This graph shows the evolution of the average reward during a learning pro-

cess. The average reward tends towards a limit value which determines a measure of the performance of the learning. In the case described by the figure 8, the algorithm reaches a performance about 0.4. So, more than one action out of three is a good pushing action.

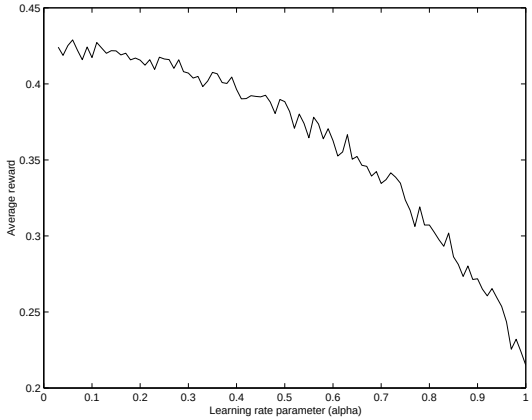

Figure 9: Performance of the algorithm according to α with $\gamma = 0.5$.

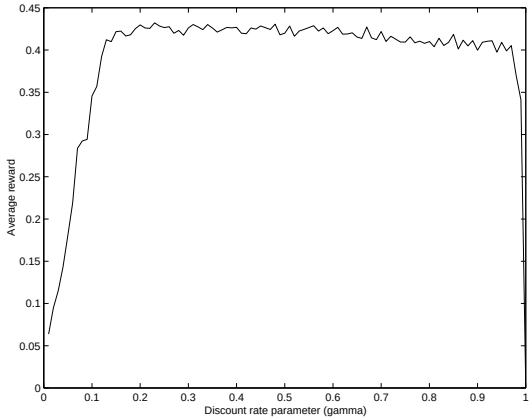

Figure 10: Performance of the algorithm according to γ with $\alpha = 0.1$.

We studied the performance of the learning according to the α and γ parameters. The figures 9 and 10 show the results of simulation experiments with ten objects. They represent the average reward after convergence according to α and γ. Each of these graphs has got an optimum. The performance of the algorithm is optimal with $\alpha \simeq 0.1$ and $\gamma \simeq 0.5$. We choose these values for the next experimental tests.

The figure 11 shows that the convergence speed depends on the number of objects. The more objects there are, the faster the convergence is. In fact, when there are more objects, the controller can do several backups at the same time, so it learns faster. This result is very interesting to reduce the learning time. With this method, the high number of objects becomes an advantage for the controller.

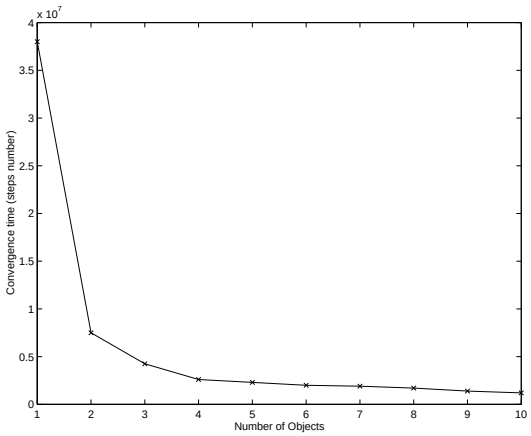

Figure 11: Convergence time according to the number of objects with $\alpha = 0.1$ and $\gamma = 0.5$.

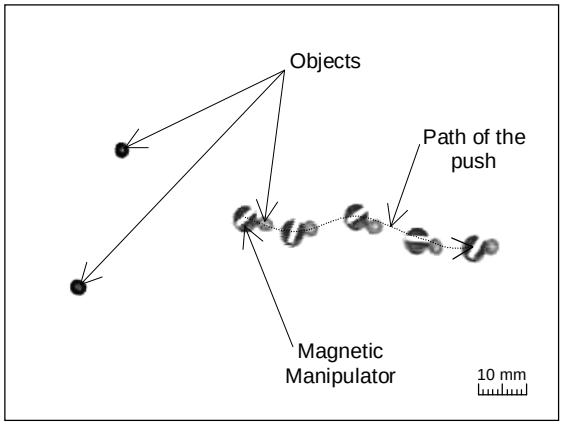

Figure 12: Video sequence of the push of a first object.

4 Experimental results

After a simulation learning process, the controller is connected to the real manipulator system. The controller adapts itself to the reality in a short time, because the general policy is the same. There is only a few number of states around the objects to update.

A pushing sequence is always the same : the tool moves towards the nearest object, bypasses it and pushes it to the right side of the screen. The figure 12 shows a video sequence in which the tool pushes a first object. After this first task, it begins to look for another object and pushes it, etc. (cf. figure 13).

When the tool pushes an object, it tries to follow a straight line because it is the optimal path. During the search of an object, the tool bypasses the object very near not to wasp time. It follows also the optimal path. So, for one object, the controller always optimizes the path of the push.

From a global point of view, the controller begin with the nearest object. This choice is not necessarily optimal. Nevertheless, as for the travelling salesman problem, this choice seems to be a good heuristics in much cases.

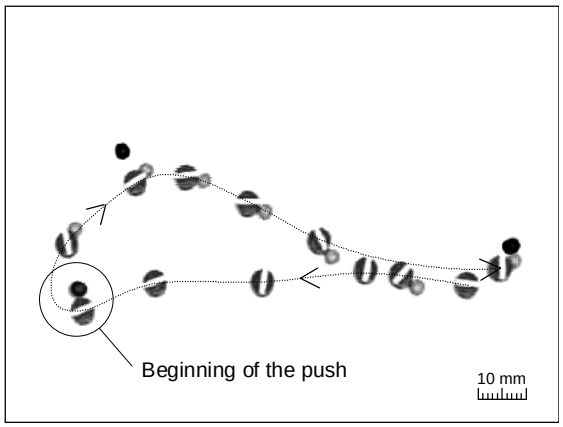

Figure 13: Video sequence of the search and the push of a second object.

5 Conclusion

The controller is based on a parallel approach to the Q-Learning algorithm. It has to perform block-pushing tasks using a micro-manipulation station with no-linearity and hysteresis features.

Simulations allowed us to choose the optimal parameters of the algorithm. From a local point of view, the controller optimizes perfectly its paths. From a global point of view, the controller chooses the nearest object. This global behavior is close to the optimal policy.

This architecture shows good performances. The main interest is that the controller turns the high number of objects to good account : due to the multitude of objects, it learns faster. Nevertheless, we know that the processing of the video image is fundamental, because the complexity is reduced during the image processing. Afterward, we would like to simplify this process and to connect the algorithm direct to the pixels of the camera.

References

[1] Michaël Gauthier, Emmanuel Piat, and Alain Boujault. Force study for micro-objects manipulation in an aqueous medium with a magnetic micro-manipulator. *Submitted to Mecatronics 3rd European-Asian Congress*, October 2001.

[2] Ben-Shahar Ohad and Rivlin Ehud. To push or not to push : On the rearrangement of movable objects by a mobile robot. *IEEE Transactions on systems, man and cybernetics, part B : cybernetics*, 28(5):667–679, October 1998.

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

[4] Watkins Christopher J.C.H. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.

[5] Brooks Rodney A. and Mataric Maja J. Real robots, real learning problem. *Robot Learning*, pages 193–214, 1993.

[6] Sutton Richard S. and Barto Andrew G. *Reinforcement Learning : An Introduction*. The MIT Press, 1998.

[7] Kaelbling Leslie Pack and Moore Andrew W. Reinforcement learning : A survey. *Artificial Intelligence Research*, 4:237–285, 1996.

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