Learning to control a real micropositioning system in the STM-Q framework

Cédric Adda, Guillaume J. Laurent, Nadine Le Fort-Piat
Laboratoire d'Automatique de Besançon UMR CNRS 6596
24 rue Alain Savary, 25000 Besançon, France
e-mails: (cedric.adda,guillaume.laurent,nadine.piat)@ens2m.fr, url: www.lab.cnrs.fr

Abstract— **This paper presents a new reinforcement learning algorithm named Short Term Model-based learning (STM-Q) able to learn quickly to control stochastic systems. Our objective is to control the positioning of small mechanical parts with a micromanipulation system for microfactory applications. The micropositioning platform is made of three linear actuators that move points to push the object toward a goal position. At the micrometric scale, it is really difficult to model the behaviour of manipulated objects. Moreover, due to the gap between real world and discrete world perceived by the camera, the motion of parts can be seen as a little stochastic. To overcome these problems and design an automated device, we use reinforcement learning methods and develop a new model-based algorithm taking into account the stochastic behaviour of the positioning task.**

Index Terms— **real robot learning, reinforcement learning, model-based algorithm, microrobotics.**

I. Introduction

Positioning is a basic function in industrial production, for example for machining or assembly. For microrobotic applications, some works [1][2] have shown that for micropositionning, pushing an object can be used instead of pick-and-place to reduce mechanical difficulties. Many works use the cantilever of an atomic force microscope to push and position particules [3][4]. Huang [5] presents another solution for fine positioning of macroscopic objects, made of three linear actuators, that use tapping as way of displacement. Zyvex company proposes a similar device using four points, intended for pushing particules under scanning electron microscopes. In most cases, these devices are manually operated.

We are interested in the positioning of small mechanical parts, between a hundred of microns and a few millimeters. For this purpose, we build a manipulation device made of three precise linear actuators that push the object (cf. Fig. 1). The aim of our works is to get a completely automated device. But in small dimensions, the classic mechanical equations, based on the friction forces and on the weight of objects, are not suitable anymore because of the importance taken by the adhesion forces [6]. In fact, the problem which consists in foreseeing the motion of a solid put on a plan and undergoing a punctual push is an already complex problem in larger dimensions [7]. As a consequence, we do not have a reliable model of the displacement of the objects that we want to position. That is why we chose to use a control method based on learning, which do not need a model of the system.

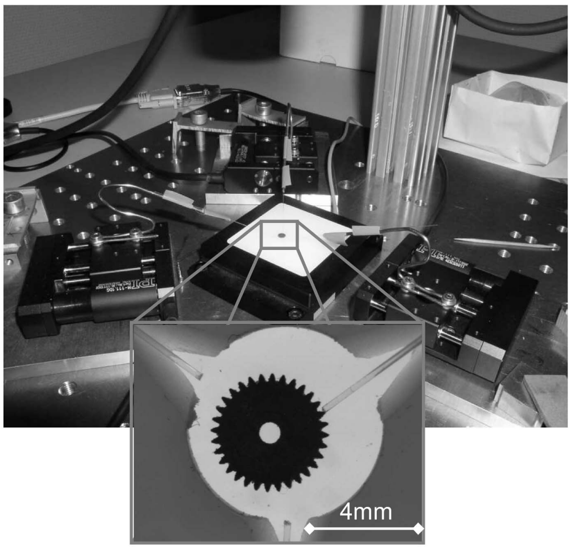

Fig. 1. Micropositioning device

Mahadevan [8] summarizes a number of possible approaches to perform learning on real robots. In our case, the learning must be unsupervised because we don't have examples of good control policy to train the controller. So, we chose to use reinforcement learning.

Mahadevan argues that robot learning is hard because of sensor noise, stochastic actions, real-time response, on-line learning, limited training time and situated representation (partial observations of the state).

In our microrobotics application, the environment is well controlled. Thanks to the CCD camera, the state of the object is well and completely observable. But due to the limited camera resolution, the pushing actions can be seen by the controller as a little stochastic. Our second main problem is the learning time. Basic reinforcement learning algorithms take a long time to find a good control policy. There are a lot of researches to improve their speed. We focus here on works dealing with *real* robots.

To learn quicklier, one idea is to decompose the overall task into several simpler behaviors. Each of these behaviors is learned using a reinforcement learning algorithm. Mahadevan and Connell [9], Asada [10], Laurent and Piat [11] use this principle with success to learn real box-pushing tasks and soccer tasks. In our case, we don't have any idea how to decompose the task.

Atkeson and Schaal [12] (inverted pendulum), Smart [13] (corridor following) use human demonstration to speed up the learning. In our case, the task takes a long time and is dull. The aim is really to get an autonomous and flexible controller that learn without any help from human.

Others solutions use approximation functions (neural networks [14] , locally weighted regression [15]) or model-based approach [16] [17]. The main interest of approximation functions is the generalization capability. Drawbacks are that convergence guarantees toward an optimal control policy are lost and that there are a lot of parameters to tune.

The main interest of model-based algorithms is to do off-line optimization of the control policy. So, the on-line time can be reduced. The drawback is that the state space must be small. In our case, due to the camera resolution, the state space is small but the system is seen as stochastic. Among model-based algorithms, the Sutton's Dyna-Q [18] can't be used because it doesn't work with stochastic systems. Priority sweeping [19] can deal with stochastic systems but require too much memory[1].

For those reasons, we propose here a new reinforcement learning algorithm called Short Term Model-based reinforcement learning (STM-Q). This paper presents this algorithm and the obtained results with both simulated and real manipulation systems.

II. Reinforcement Learning

A. General principle

In the reinforcement learning framework, the controller has a finite set $\mathcal{U}$ of actions it can perform and the system has a finite set $\mathcal{X}$ of states. At each step of learning, the controller takes an action u_t that leads the system from the state x_t to the new state x_{t+1}. Then, it receives a reward r_{t+1} from the system according to x_t, u_t and x_{t+1} (cf. Fig. 2). The objective of the control using learning consists in seeking actions that maximize the long term sum of rewards (*return*) given by:

$$G(t) = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}(x_{t+k}, u_{t+k}, x_{t+k+1}) \qquad (1)$$

A learning is divided into episodes which are a set of steps that end when the system reaches a final state called absorbing state. Absorbing state can be either the state we want the object to reach or a state from where the system can not escape.

[1]About $|\mathcal{X}| \times |\mathcal{U}| \times |\mathcal{X}|$ values if $\mathcal{X}$ is the state space and $\mathcal{U}$ the action space.

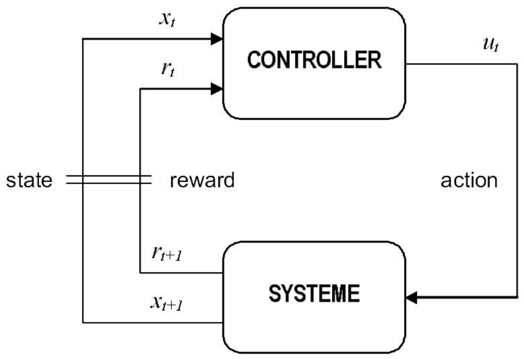

Fig. 2. Sensorimotor loop.

B. Q-Learning

Q-Learning [20], is one of the most used reinforcement learning algorithm. It uses an array to store an evaluation $Q(x, u)$ of the expected return for each state-action couple (x, u). At each step of learning, the evaluation of the state-action couple (x_t, u_t) is updated using this equation:

$$\begin{aligned} Q(x_t, u_t) \leftarrow\, & Q(x_t, u_t) + \\ & + \alpha[r_{t+1} + \gamma \max_v Q(x_{t+1}, v) - Q(x_t, u_t)] \end{aligned} \qquad (2)$$

The α coefficient is a step-size parameter that influences the rate of learning, and the γ coefficient is the discount-rate parameter and determines the importance of future rewards.

According to the expected return of each action, an algorithm chooses the action with an exploration / exploitation criteria. In this paper, we use the ϵ-greedy: with a small probability ϵ, a random action is selected, otherwise, the selected action is the one for which $Q(x, u)$ is maximum. This method allows to easy control the exploration rate.

At the end of the learning, the controller should have learnt the optimal policy. So at each step, according to the state of system, it just has to choose the action that has the best estimation to follow the best control policy.

One of the main problem with Q-Learning is the time needed to learn an optimal policy. To improve this aspect, it is possible to use a model-based algorithm as the Dyna-Q.

C. Dyna-Q

Dyna-Q [18] is an evolution of the Q-Learning algorithm in which a partial model is built during the learning. The model is made of an array in which is saved, for each state-action couple (x, u), the new state and the reward obtained when the controller took the action u in the state x last time.

At each learning step, after having updated the evaluation of the state-action couple $Q(x_t, u_t)$, the Q-values of several other state-action couples are also updated using the model (off-line optimization).

The main improvement of Dyna-Q over Q-Learning is the reduction of the on-line time to learn the optimal policy, which seems very convenient when learning to control a real device.

But in Dyna-Q, the system must be deterministic. The problem is that our system is not deterministic : each time an action is taken in a given state, the system is not sure to always reach the same new state, due to the sampling of the real world made by the camera. So for our purpose, Dyna-Q is not suitable.

D. Short-Term Model-based learning (STM-Q)

To solve both these problems - the very long learning time and the fact that the system is not deterministic - we propose a new algorithm named Short-Term Model—based learning (STM-Q), which is presented in Fig. 3.

In the STM-Q, the memory stores a model M which keeps a historical record of M_{max} *state—reward* couples (x', r) for each *state—action* couple (x, u). These *state—rewards* couples (x', r) correspond to the state and the reward obtained during the last passages where the controller observed the state x and selected the action u. The number M_{max} of couples (x', r) kept for the couples (x, u) is called the memory size.

The model $M(x, u)$ is a FIFO buffer which cardinal is at most M_{max}. If $M_{max} = 4$, we have for example:

$$M(x,u) = \left\{(x'_{t_1}, r_{t_1}), (x'_{t_2}, r_{t_2}), (x'_{t_3}, r_{t_3}), (x'_{t_4}, r_{t_4})\right\} \tag{3}$$

with $t_1 > t_2 > t_3 > t_4$. Each couple (x'_{t_i}, r_{t_i}) corresponds to a past observation of the state and the reward obtained when the controller did the action u while observing the state x. The couples can be different if the system is not deterministic.

The storage of a historical record for each couple allows to calculate an estimate of the transition probability of moving from state x to state x' given that we have applied action u. The maximum likelihood is given by:

$$\widehat{p}(x'|x,u) = \sum_{\forall (y,r) \in M(x,u) | y = x'} \frac{1}{|M(x,u)|} \tag{4}$$

Using this estimation, the action-value function can be optimized without using the learning-rate parameter α, even in case the system is not deterministic. The update equation for a couple (x, u) is then:

$$Q(x,u) \leftarrow \sum_{\forall (y,r) \in M(x,u)} \frac{1}{|M(x,u)|} \left[r + \gamma \max_{v \in \mathcal{U}} Q(y,v) \right] \tag{5}$$

III. Positioning device

A. Positioning small parts

The objective is to fine position small mechanical parts in a fully automatic way. This task is performed by three linear actuators that carries points. Each actuator has only one degree of freedom: it can only go forward and backward in order to push the part to position.

The object to be manipulated is brought to the manipulation area by another robot with a coarse resolution. The objective is that the work of the three points leads to the good and fine positioning of the object (cf. Fig. 4).

To get an automated task we need a feedback to get the position of the part. This feedback is done using vision (cf. Fig. 5).

When the device learns to position the object, the points can put the object in a position from where they can not bring it back. To avoid this problem, we decided to put a barrier around the working area that prevents the object to

```
**Initialization:**
$\forall (x,u) \in \mathcal{X} \times \mathcal{U}$, $Q(x,u) \leftarrow Q_0$, $M(x,u) \leftarrow \emptyset$
$H \leftarrow \emptyset$
*($H$ is the list of the state-action that have been seen at least once)*

**For each episode:**
$x$ ←current state
**While** $x$ is not an absorbing state **do**
    $u \leftarrow \epsilon$-greedy$(Q, x)$
    Take action $u$, observe new state $x'$ and reward $r$

    **If** $M(x,u) = \emptyset$ **then**
        *(the $(x,u)$ state-action couple is met for the first time)*
        $H \leftarrow H \oplus \{(x,u)\}$   *(concatenation)*
    **End if**
    **If** $|M(x,u)| = M_{max}$ **then**
        Suppress the older observation in $M(x,u)$
    **End if**
    $M(x,u) \leftarrow M(x,u) \oplus \{(x',r)\}$   *(concatenation)*
    $x \leftarrow x'$

    **For all** $(y,v) \in H$ **do**
        *(off-line optimization)*
        $Q(y,v) \leftarrow \sum_{\forall (y',r) \in M(y,v)} \frac{1}{|M(y,v)|} \left[ r + \gamma \max_{w \in \mathcal{U}} Q(y',w) \right]$
    **End for**
**End while**
```

Fig. 3. STM-Q algorithm

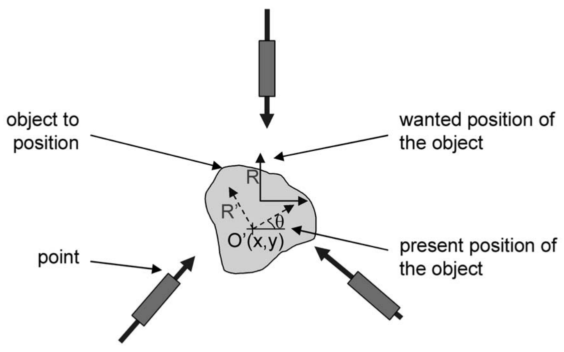

Fig. 4. Top view of the micropositioning device

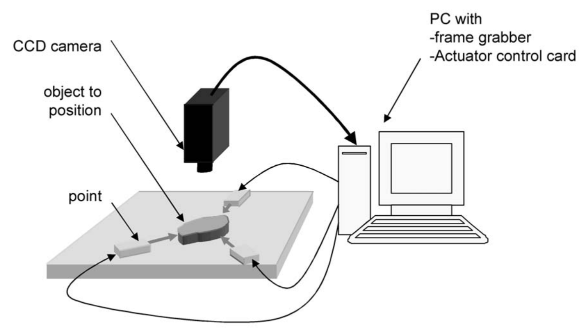

Fig. 5. Large view of the micropositioning device

become unreachable. This barrier is a part of the system, so it can be used by the point to position the object.

B. Device components

The actuators are Physik Instrumente Micro Translation stages with a very high resolution, large range and convenient use.

The points are optical fibers. Optical fibers are rigid enough to push parts and offer interesting features. First one is their flexibility: if a solid point try to push the part when it is blocked by the barrier, it could break itself or spoil the part. Instead, in this situation, the optical fiber will bend. The second interesting feature is the transparency of the optical fiber very useful for the vision feedback.

Indeed, we use a black and white CCD camera whose the image is grabbed using a PC-type computer with a frame grabber card. The scene is back-lighted up, so that the object is well defined on the image. As the optical fibers, the barrier is transparent, which makes them invisible by the camera. That way, the camera only sees the part to position, which is very helpful to analyse the images.

C. Reinforcement learning controller

In order to use our reinforcement learning controller, we need to define the state space, the action space and the rewards.

1) State space: To define states, it is necessary to be able to characterize the position of the object. For that purpose, we use a reference mark $R(O,\vec{i},\vec{j})$ motionless for the camera and a reference mark $R'(O',\vec{i'},\vec{j'})$ fixed to the solid to be positioned (see Fig. 4). x and y, coordinates of O' in R, determine the position of the object. The object we use is circular so we do not take into account its orientation.

We define the various states of the system by sampling the space of the values which x and y can take. In fact, we use the sampling made by the CCD sensor of the camera so that a state of the object correspond to a pixel of the grabbed image. There are about 3000 states in $\mathcal{X}$.

2) Action space: we chose to use twice as many actions as points, which means there are six actions in the $\mathcal{U}$ set. When we execute one of the two actions relative to a point, this one moves forward until it touches the object and then pushes it on a length which was fixed at the beginning. The only difference between the two actions is the pushing length : each point can either perform a small push to precisely move the object or a long push in order to accelerate the positioning task. This kind of actions is rather simple, but it is useful to limit the size of the action space.

3) Reward: The reward function has to give a reward after execution of the action u_t when the system is in the state x_t. This reward can depend only on the new state x_{t+1} or can take into account the initial state x_t and even the action u_t itself.

In our case, we decided to attribute a reward equal to 1 if the new state of the object is the wanted position, 0 if not. That way, we do not influence the learning.

IV. Simulation results

A. Simulated device

To test the controller in a fast and easy way, we developed a simulated device that can be used instead the real one to perform a learning. This simulated device is built using the characteristics of the real device. So it as the same number of states, which are defined in the same way and can perform the same actions. A *simplistic* mechanical model allows to simulate the effect of pushing an object by a point.

The used mechanical model is far from reflecting perfectly the reality, and farther from the reality of the microworld. But it is useful here to allow a first validation of our approach under simulation.

All simulated learning have been done 20 times. The graphics shows the average curve of 20 learning.

B. Results

1) Influence of ϵ parameter: To study the influence of the ϵ parameter over the learning process, we made simulations using several values. The results can be observed on Fig. 6.

We can see that, whatever value of ϵ is taken, the controller manages to learn to position the part. The main difference is that, with a bigger value, once the learning is made, more actions are needed to perform the task. The reason is that the controller explores more the state-command space: it more often chooses actions randomly instead of using what has been learnt.

2) Influence of pushing length: Another parameter we wanted to know the influence is the length L of a long push. We know the short pushing has to stay small enough to be able to reach the wanted position precisely but we

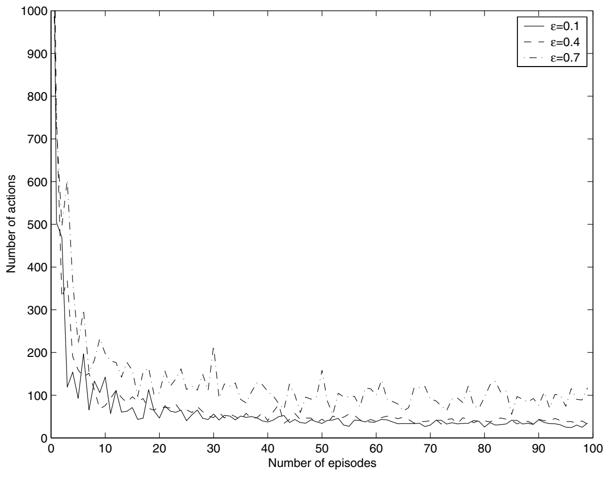

Fig. 6. Effect of ϵ parameter on the simulated manipulation task (with $\gamma = 0.9$)

long push length	number of steps for the first episode	number of step for the 50th episode
0.2	83157	49
0.3	34348	57.3
0.4	1875.8	41.95
...	...	...
1	1551.8	36.35

Fig. 7. Effect of long pushing length on the simulated manipulation task ($\gamma = 0.9$, $\epsilon = 0.1$)

wanted to know what is the real avantage having a long push.

For any value of L from 0.2 to 1 mm, if we wait long enough, the controller will learn to position the device. What change with the value of L is the duration of the first episode. In Fig. 7 we represented the average number of steps of the first learning episode for several values of L. There is no much difference between a value of long pushing length of 0.4 mm or 1 mm. But what is important is that with a value under 0.4 mm, the first episode last really long. In this case, if we observe the motion of the part, we see that it stays in a small area close to the barrier, being pushed alternatively by the three points. In fact, the long push has to be long enough to allow to quickly get out of this deadlock.

3) Comparison between STM, Q-Learning and Dyna-Q: We finally used the simulated scene to compare the efficiency of Q-Learning, Dyna-Q, and STM-Q algorithms.

As you can see on the Fig. 8, only Q-Learning and STM-Q algorithms manage to learn an efficient policy. As we supposed, the Dyna-Q can not deal with the non-determinism of the device and then do not manage to learn. So we can see that the STM-Q algorithm manages to correct this aspect which prevents Dyna-Q to be used on real device.

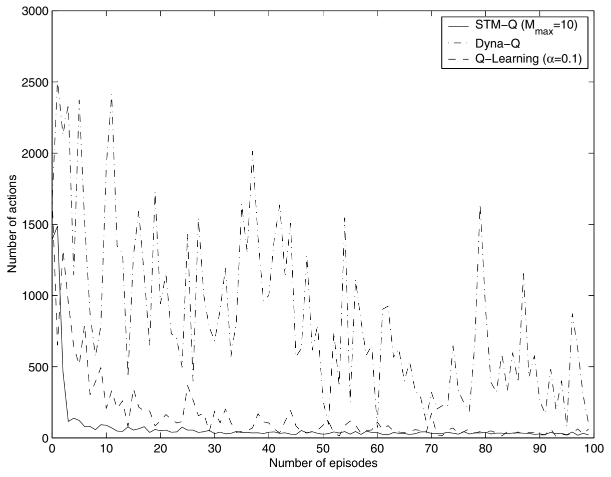

Fig. 8. Comparison between STM-Q, Q-Learning and Dyna-Q on the simulated manipulation task (with $\gamma = 0.9$, $\epsilon = 0.1$)

Finally, we can also see that the STM-Q learning is really faster than the one using Q-Learning, which shows the interest of building a model during the learning.

V. Experimental results

A. First experimental results

At last, we have performed first tests on the real device. You can see in Fig. 9 the curve of one real learning with the same learning parameters as in simulation ($\gamma = 0.9$, $\epsilon = 0.1$) . It has to be noticed that this curve is not directly comparable to the ones shown for the simulated device. As we said above, the simulation curves show the average of 20 learning, while this curve is the result of only one learning. Indeed, the real learning took more than 24 hours. So it is really too long to do average curves. Nevertheless, it is obvious that the results are not as good as in simulation. Yet, the controller managed to improve its control policy, and it is an encouraging result.

B. Further look over the learnt policy

To see more easily what the controller learnt we have drawn on Fig. 10, for each state x, the evaluation $V(x)$ of the learnt policy:

$$V(x) = \max_{v} Q(x, v) \tag{6}$$

We can recognize the position of the three points (the dotted lines on Fig. 10), because it is when the object is on the way of a point that it is easier to position it. The encircled areas correspond to states that have a high value of V. The one encircled by the dotted circle shows the states that are near to the wanted position. If the system is in one of these states, only a few small pushes will be necessary to achieve the positioning task. The three areas encircled by the solid circles are more interesting : they correspond to states that are far from the wanted position, but that can lead the object quickly to this position after,

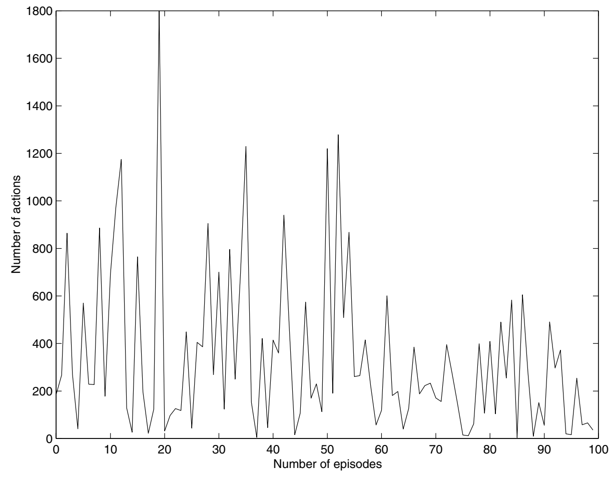

Fig. 9. Learning on the real device (with $\gamma = 0.9$, $\epsilon = 0.1$)

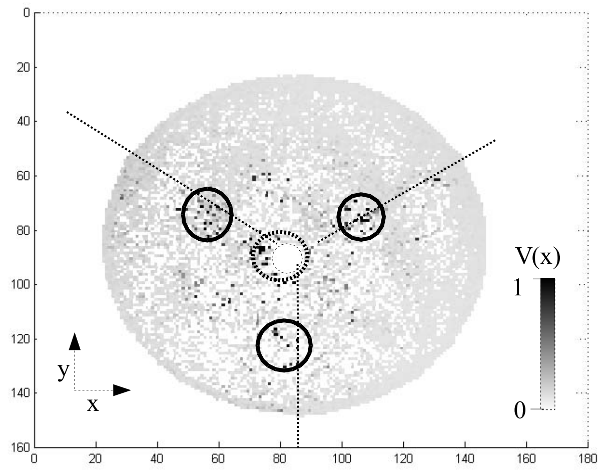

Fig. 10. Evaluation of the policy.

at least, one long push. This validates our choice to give no reward to the state-action couples that bring the object closer to the wanted position. The fastest way to achieve the task is not always to go through the states which are the nearest to the wanted position.

VI. Conclusions and further works

The results obtained on the simulated micropositioning system show that the STM-Q algorithm leads to best results in term of learning time compared to the Q-Learning. Contrary to the Dyna-Q algorithm which does not work with stochastic systems, the STM-Q algorithm converges quickly with few additional memory requirements.

First tests on the real micropositioning system show that, using the STM-Q Learning, the controller achieves to learn a control policy. Further tests will be realized to confirm the good results of the STM-Q algorithm on the real system.

References

[1] Wolfgang Zesch and Ronald S.Fearing. Alignment of microparts using force controlled pushing. In *Proc. of the SPIE Conf. on Microrobotics and Micromanipulation*, volume 3519, pages 148–156, Boston, Massachusetts, november 1998.

[2] C. Baur, A. Bugacov, B.E. Koel, A. Madhukar, N. Montoya, T.R. Ramachandran, A.A.G. Requicha, R. Resch, and P. Will. Nanoparticle manipulation by mechanical pushing : underlying phenomena and real-time monitoring. *Nanotechnology*, 9:360–364, 1998.

[3] Theil L. Hansen, A. Kühle, A.H. Sørensen, J. Bohr, and P.E. Lindelof. A technique for positioning nanoparticles using an atomic force microscope. *Nanotechnology*, 9:337–342, 1998.

[4] R. Resch, D. Lewis, S. Meltzer, N. Montoya, B.E. Koel, A. Madhukar, A.A.G. Requicha, and P. Will. Manipulation of gold nanoparticles in liquid environnements using scanning force microscopy. *Ultramicroscopy*, 82:135–139, 2000.

[5] W.H. Huang. Control strategies for fine positioning via tapping. In *Proc. of the IEEE International Symposium on Assembly and Task Planning*, 2003.

[6] C. Guoliang and H. Xinhan. Research on vacuum micro-gripper of intelligent micromanipulation robots. In *Proc. of the IEEE Int. Conference on Robotics and Biomimetics*, august, 2004.

[7] M.A. Peshkin and A.C. Sanderson. The motion of a pushed, sliding workpiece. *IEEE Journal on Robotics and Automation*, 4(6):569–598, 1988.

[8] Sridhar Mahadevan. Machine learning for robots: A comparison of different paradigms. In *Workshop on Towards Real Autonomy, IEEE/RSJ International Conference on Intelligent Robots and Systems*, Osaka, Japan, 1996.

[9] Sridhar Mahadevan and Jonathan Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, (55):311–365, 1992.

[10] Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23(2–3):279–303, 1996.

[11] Guillaume J. Laurent and Emmanuel Piat. Learning to control real systems with parallel dyna-q. In *Proc. of the Sixth European Workshop On Reinforcement Learning*, pages 45–46, Nancy, France, September 4–5 2003.

[12] Christopher G. Atkeson and Stefan Schaal. Learning tasks from a single demonstration. In *Proc. of the IEEE International Conference on Robotics and Automation*, volume 2, pages 1706–1712, 1997.

[13] William D. Smart. *Making Reinforcement Learning Work on Real Robots*. PhD thesis, Department of Computer Science, Brown University, May 2002.

[14] Lin Long-Ji. Self–improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992.

[15] Stefan Schaal, Chris Atkeson, and Sethu Vijayakumar. Real time robot learning with locally weighted statistical learning. In *Proc. of the IEEE International Conference on Robotics and Automation*, volume 1, pages 288–293, San Francisco, California, 2000.

[16] Bir Bhanu, Pat Leang, Chris Cowden, Yingquiang Lin, and Mark Patterson. Real-time robot learning. In *Proc. of the IEEE International Conference on Robotics and Automation*, Seoul, May 2001.

[17] M. Carreras, P. Ridao, and A. El-Fakdi. Semi-online neural-q learning for real-time robot learning. In *Proc. of the International Conference on Intelligent Robots and Systems*, pages 662–667, Las Vegas, USA, October 2003.

[18] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proc. of the Seventh International Conference on Machine Learning*, pages 216–224, San Mateo, CA, 1990. Morgan Kaufmann.

[19] Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13, 1993.

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