Learning strategy with neural-networks and reinforcement learning for actual manipulator robot

Shingo Nakamura[1] and Shuji Hashimoto[2]

[1,2]Waseda University, Tokyo 169-8555, Japan
(Tel: 81-3-5286-3233, Fax: 81-3-3202-7523)

[1]shingo@shalab.phys.waseda.ac.jp, [2]shuji@waseda.jp

Abstract: When the bottom-up learning approaches are implemented for mechanical systems, we must face a problem including huge number of trials. They take much time and give hard stress to the actual system. Simulator is often used only for evaluation of the learning method. However, it needs simulator modeling process, and never guarantees repeatability for the actual system. In this study, we are considering a construction of simulator directly from the actual robot with neural-networks. Afterward a constructed simulator is used for reinforcement learning to train a task, and the obtained optimal controller is applied to the actual robot. In this work, we picked up a five-linked manipulator robot, and made it track a ball as a training task. Both learning processes make load against the hardware sufficiently smaller, and the objective controller can be obtained faster than using only actual one.

Keywords: manipulator robot, neural-networks, reinforcement learning, simulator construction.

1 INTRODUCTION

The machine learning method is one of the typical techniques that may provide robots a capability to work in unpredictable and variable human life environment. Until now, many kinds of machine learning method have been proposed and they yield very effective results [1][2]. However, if they are directly applied to the actual machine, they face a problem including huge number of trials, which requires much time and gives stresses against the hardware and then makes a practical application difficult. To avoid such problems, a computational simulator is often employed and performed with a learning method since it permits many trials in short time without physical stresses [3]. However, the simulator utilization for the real system also includes some problems in its construction. Normally, the simulator is manually modeled by differential equation according to the physical law. Consequently, it never makes us completely free from exacting works. Moreover, there is no guarantee that the modeled simulator performance matches the actual machine when its construction is complicated. Hence, we propose a simulator construction directly from the actual hardware. In our method, a simulator learns the relationship between command signal and its resultant state of the hardware. Therefore, if the learning proceeds well, a simulator emulates the hardware behavior certainly, and we are relieved from arduous simulator modeling. After simulator construction, the optimal controller for an objective behavior is trained only with the constructed simulator. Therefore, this process never gives harsh load to the hardware and performs faster than directly using the hardware.

In this paper, we describe a novel method of machine learning using both a simulator and an actual hardware. A simulator of the hardware is directly built from the acquired input and output data of the real hardware and it performs with the neural-networks and the back-propagation learning method without any information of kinematics model. Afterward, the objective controller of the hardware is trained only with the built simulator by the reinforcement learning method. Finally, the optimum controller is applied to the actual hardware. In this manner, the optimal controller is generated via hybrid platform: simulator and hardware, without any knowledge in advance. Here we employed a five-linked manipulator robot as an actual hardware platform, and the only treated properties are state and command signals of actuators assembled into the robot. The task for the robot to train is to track a colored ball. Under these conditions, experiments were conducted and the prosed method was evaluated.

2 BASIC STRATEGY

Fig.1 illustrates a basic structure and process of our method. At first, data of the actual hardware behavior is sampled and collected into a buffer. In this case, data means a set of command and state of the actual hardware, and a resultant state corresponding to the command, which represents the relationship between input and output of the hardware. And then the buffer supplies reference data to

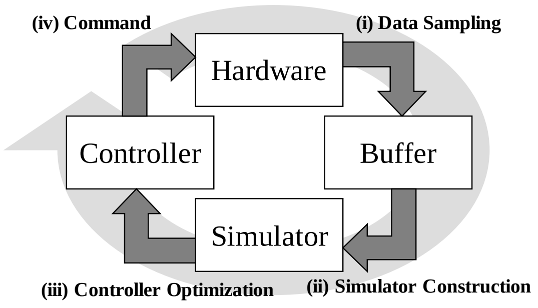

Fig.1. The over view of learning strategy

build a simulator in the appropriate manner. Afterward a controller is trained the optimal control for the objective task using the built simulator. At this time, because of a computational simulator, the hardware does not have to be actually operated and training process performs faster than using the actual hardware. Finally, the optimized controller operates the actual hardware to perform the objective behavior. In this way, the whole process starts from the hardware acting, and the information is returned to the hardware. Based on this structure, we apply to the five-linked manipulator robot to get the optimal task control. The implementation of each process for this problem is explained in the chapter 4.

3 MANIPULATOR ROBOT

In this paper, our method is applied to an actual robot. The robot we employed is a five-linked manipulator robot, composed of five servo motors as shown in Fig.2, and their link relations are illustrated in Fig.3. The motor is digital servo type and performable in serial communication to receive the action command and return the status information. The motor control mode is configured to "speed" mode. Finally the robot hardware consists of such motor, and state and action command of robot mean angle and speed command value for each motor. Specifically, we can just send a speed command value $a = (a_1, a_2, a_3, a_4, a_5)$ to each motor as action command for robot, and observe each motor angle $s = (s_1, s_2, s_3, s_4, s_5)$ as robot state. Some control limitations are given in software level. Every joint is configured to be able to move in range [-π/2, +π/2]. If it is out of the range, it stops until command to move for inside direction comes. Using this platform robot, our method is evaluated by training ball-tracking task.

4 IMPLIMENTATION

As shown in Fig.1, there are four processes in our strategy and their efficient implementation provides a self-learning ability to the system. In this chapter, detailed implementation for the manipulator robot is described.

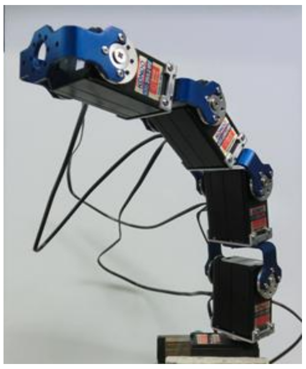

Fig.2. The manipulator robot

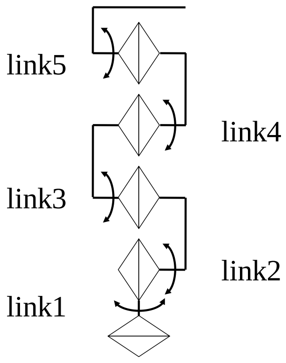

Fig.3. The link structure

4.1 Data Collection and Supply

The data sampled from the actual robot is collected into a buffer. In this work, data to collect is a pair information of (s_t, a_t) and $s_{t+\Delta t}$. These mean the robot state s and command a at time t, and its response state s at time $t+\Delta t$. And it is supplied to a simulator for construction. Therefore, the buffer has to keep and supply the suitable data to represent the actual robot.

In order to supply the suitable data to construct a simulator, the limited parameter space $\mathbf{S} \times \mathbf{A}$ is divided into some sub-spaces in a number of $N_S \times N_A$ in similar way. Each sub-space keeps corresponding data. At simulator construction by neural-networks, one sub-space is chosen at random and one sampled data is provided as learning data.

4.2 Simulator Construction

The Multi-Layered Perceptron (MLP) of the neural-networks which is effective for the approximation of a nonlinear function is employed to simulate the hardware. It directly represents the robot behavior. The MLP consists of three layers, i.e. input, hidden and output layer. The MLP is input 10 parameters, namely the manipulator state s_t and command a_t for each motor, while it outputs next robot state $s_{t+\Delta t}$. Each neuron in the output layer is a nonlinear unit expressed with the arctangent sigmoid function whose output is limited in [-1, 1]. Therefore, the output value s_i [-π/2, +π/2] is normalized into [-1, 1].

A simulator has the MLP network inside. In the simulation process, it follows the MLP. The simulator inputs the data set (s, a) at time t, and obtains the normalized difference values in [-1, 1] as an output signal from the MLP. The de-normalized values s is simply represented as the state at time $t+\Delta t$. Fig.4 illustrates the overview of running process of simulation with the MLP.

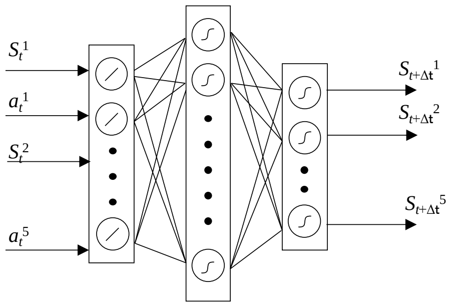

Fig.4. Neural-networks process for robot simulator

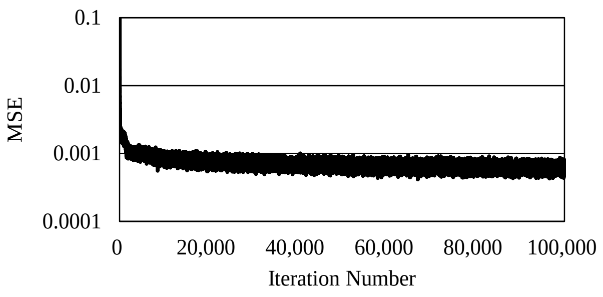

Fig.5. Result of simulator construction learning

And in the process of learning, the teacher data is supplied from the buffer, and the back-propagation method performs.

4.3 Controller Optimization

A controller learns the optimal control using a simulator. Here, we focus on the reinforcement learning [4] to make robot track a ball. In the reinforcement learning, the action is evaluated by the given rewards, and the system learns empirically the optimum action by trials and errors. In this work, we employ the Q-Learning method for the reinforcement learning. This is a typical method of reinforcement learning. When the agent at time t takes the action a_t based on the current state s_t, the state-action value function $Q(s_t, a_t)$ is updated as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (1)$$

where r_t is the reward the agent receives at time t, α is a learning ratio and γ is a discount ratio. In the phase of training, the agent chooses an action randomly. The other hand, it takes the action a where $Q(s_t,a)$ is the max value at state s_t in execution phase. In this way, a controller of the robot explores the optimum action command for each state and then performs tracking the ball.

5 EXPERIMENT AND RESULT

5.1 Data Collection

The target data to train a simulator was sampled from the actual manipulator robot described in the chapter 4. To collect data, an action command a selected in the previously mentioned manner was given with interval Δt = 50 msec. Consequently, the robot moved in the range of motion. The division numbers N_S and N_A for the five-dimensioned state space and action command space were set to 11^5 and 4^5, respectively. This collection process continued for one hour in real world. In other words, 72 thousands sample data were collected.

5.2 Simulator Construction

The learning of the neural-networks was performed by the backpropagation method. The number of neurons in the input, hidden and output layer were 10, 20 and 5, and a target sample data set (s_t, a_t) and $s_{t+\Delta t}$, were randomly took from a divided sub-space, which was selected at uniformly random from whole space. Learning ratio was set to 0.1, and nonlinearity ratio of nonlinear neurons in the MLP was set to 1.0. The number of learning iterations was 10 million times.

This operation was performed just in 10 minutes. Fig.5 shows the mean squared error (MSE) of MLP every one thousand learning. The MSE value decreased until around 1.0×10^{-3}. This means that the accuracy of learning was about 3% in normalized space, namely about 2.6 degree of actual angle.

5.3 Controller Optimization

Learning to track a ball was performed with the obtained simulator in the previous section. The state space **S** and command space **A** was divided into 9 sub-spaces at even intervals and treated as a discrete space for each joint. Hence, whole parameter space **S** × **A** was divided into $5^9\times5^9$ sub-spaces. The 250mm-diameter ball to track was put in the robot-reachable space, and then, task learning processed until the robot touched it or 2,000 steps was done without "touch". Here, "touch" means that the forward point 250 mm from the robot head is inside the ball area. This cycle is defined as "episode" in this experiment. y and z-coordinate value of ball position was fixed, and only x-coordinate value was selected at random for one episode. The controller to optimize was selected, depending on the ball x-coordinate value. Consequently, controllers as many as division number of x-coordinate were prepared. Finally, ball position x was selected at random from [-100, +100]

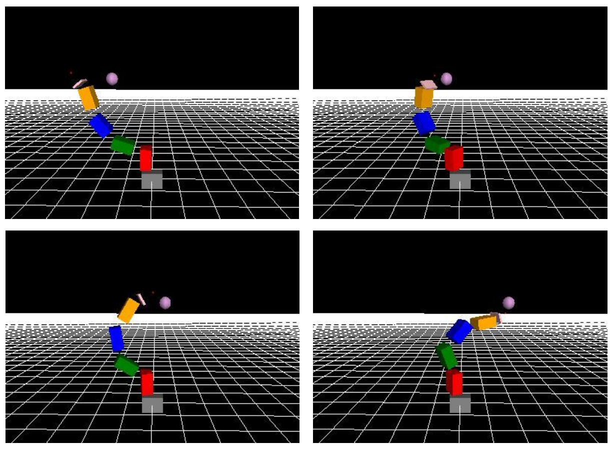
Fig.6. Ball tracking with simulator

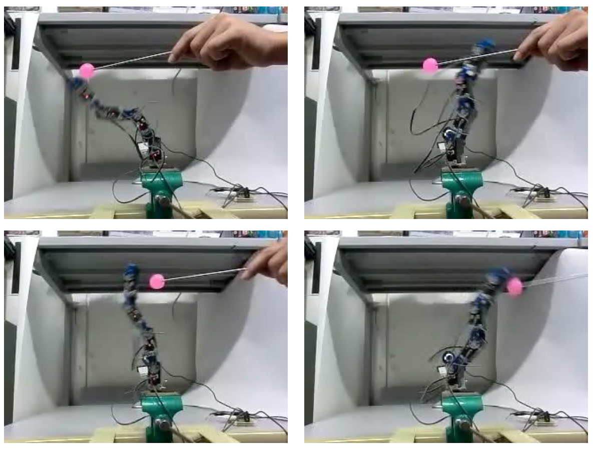
Fig.7. Ball tracking with actual robot

mm, which was divided into 10 areas, *y* and *z* were fixed into -32 mm and +192 mm, respectively.

The episodes to learn were repeated a million times in the simulation world. If this operation is executed in the real world, it would take about 28 thousand hours. The reinforcement learning performed according to the formula (1) with interval 50 ms. The parameters α and γ used for the update, were set to 0.1, and 0.9, respectively. The reward 1.0 was given only at the state "touching".

The learning process finished in twelve hours at most, though it would take about three years in the real world. Fig.6 shows the sequential images of ball tracking by simulator. Depending on the target ball position, the corresponding controller was selected and tried to make the robot touch the ball. The moving ball finally made continuous touching into tracking ball.

5.4 Application to actual robot

Finally, we applied the learning result of the previous section to the actual robot controller. In this experiment, the real ball position was detected by image processing in real time. Two web cameras were put around the robot and detect the red ball. The ball positions in camera image were converted into the real world coordinate by the Direct Linear Transformation method.

Fig.7 shows that the sequential images of the actual robot tracking ball. At the beginning of tracking, the robot tried to track the ball correctly. However, it shook more widely than the simulator robot to hold its state. As the ball moved, the robot tried following it. Nevertheless, the robot went down as if it lost the strength. On investigation, the command for the bottom-positioned motor was too weak to lift up self-body though it attempted. These results raise the possibility that accuracy of simulator by the neural-networks was not enough.

6 CONCLUSION

We proposed a novel method of machine learning associating with an actual hardware and its simulator. The simulator of the manipulator robot was constructed by the neural-networks trained with acquired data from the actual hardware without information of the physical law. Afterward, the ball tracking control for the manipulator robot was learned only through the built simulator by the reinforcement learning method. By using the simulator, the reinforcement learning could be finished much faster than using the real hardware without stress. However, when the controller applied to the actual robot, it could not represent the same way of the simulator completely.

Now we have a plan to construct a simulator after some executions of loop flow illustrated in Fig.1. Sampled data information is carried to the hardware controller and data is sampled from hardware cyclically. This is very adaptable process though physical information of target hardware is changed due to deterioration or component exchange. If this is actualized, our method comes more generalized for various machinery systems.

REFERENCES

[1] J. Bongard, V. Zykov, and H. Lipson (2006), Automated synthesis of body schema using multiple sensor modalities, Proceedings of the Int. Conf. on the Simulation and Synthesis of Living Systems

[2] K. Doya, K Samejima, K. Katagiri, and M. Kawato, (2002), Multiple model-based reinforcement learning, Neural Comput., vol.14, no.6, pp.1347-1369

[3] K. Doya (1996), Efficient Nonlinear Control with Actor-Tutor Architecture, Advances in Neural Information Processing System, pp.1012-1018

[4] Sutton RS, Barto AG (1988), Reinforcement Learning: An Introduction. Cambridge, MA: MIT Press