Learning Skill for Enhancing the Capability of Artifitial System

Yutaka Sakaguchi, University of Electro-Communications (sakaguchi@is.uec.ac.jp)

Adaptive Dynamics Matching in Sensory-Motor Fusion System and Its Implementation by Reinforcement Learning

Naoko Ogawa[1], Yutaka Sakaguchi[2], Akio Namiki[1,3], and Masatoshi Ishikawa[1]

[1] Graduate School of Information Science and Technology, University of Tokyo.
[2] Graduate School of Information Systems, University of Electro-Communications.
[3] CREST, JST, Japan.

1 Introduction

Recent progress in hardware devices and computer technologies has substantially contributed to making robotic systems more powerful and flexible. High-torque actuators have facilitated robot motions [1] and high-speed computers have realized natural and intellectual behaviors. At the same time, however, increase in the system's complexity is making it more difficult to design a system whose individual components display their potential abilities. Even with a high-speed actuator, for example, a system cannot utilize its ability if the sensing component is too slow compared to the actuator. In another case, a high-speed sensing component, such as 1ms vision system [2], could not demonstrate its ability unless the processing component cannot finish data processing within a specific time. Therefore, it is essential to balance the temporal characteristics of system components (i.e., sensors, actuators and data processing) in order to draw their potential abilities to the full. In the previous study, the authors pointed out the importance of this problem and proposed a concept of "dynamics matching" or "dynamics adjustment"[3], meaning to maximize the performance of a sensorymotor fusion system under external and physical constraints, through adjusting dynamical characteristics of its components.

Some may think of this concept as a matter of course. It is sure that engineers and designers usually tune the system specification appropriately so as to satisfy the task requirement. In most cases, however, such tuning is carried out in a case-by-case style, and based on engineers' own experience and intuition: Few studies considered this problem in an explicit and systematic manner. Rather, most theories on robotics tended to treat the performance of hardware devices and of processing system, separately. For example, mechanical dynamics is one of the major subjects of robotic research [4, 5], but the processing component has been hardly incorporated in its control theory, and it has not been discussed how the performance of processing system affected the control rule. In the research on higher functions of robotic systems (such as path planning), on the other hand, real-time dynamics is almost ignored. Resource allocation to multiple jobs is an important topic in a parallel processing system [6, 7], but their discussion cannot be applied to robotic systems because of its lack of physical viewpoints.

In general, there are two methodologies to realize the dynamics matching in a robotic system. One is to determine the specification of the components at the stage of designing a system. Although this off-line methodology can guarantee the performance in considered situations, it has some limitations. First, it is difficult in most cases to estimate the system's total performance before running the system in an actual environment. Second, the system cannot adapt to the environment and task changes once the system configuration is fixed at the design stage. The other methodology is to adjust the system configuration in an on-line manner, with monitoring its performance. With this methodology, the system can find an optimal configuration by itself and modify the configuration according to the environment and task changes, though its initial performance is expected to be low. The present discussion will be focused on the latter methodology.

The objective of the present report is to show the way to formulate the dynamics matching problem as an optimization problem, and to describe an algorithm to solve it by on-line

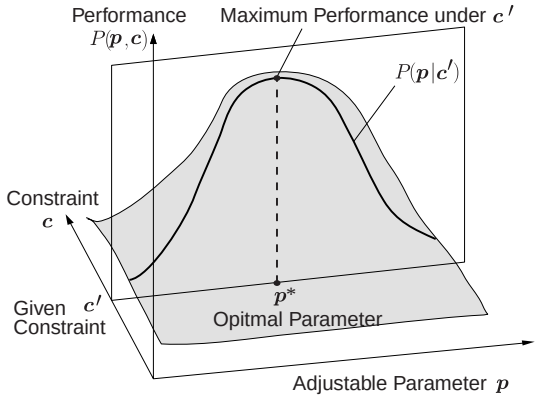


Figure 1: A Scheme of dynamics matching

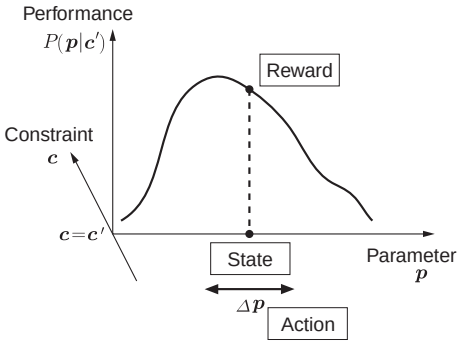


Figure 2: Dynamics matching by reinforcement learning

learning. Specifically, the authors introduce a method for adaptive assignment of computational resource under given hardware constraints and task requirements. Moreover, we implement a sample algorithm based on reinforcement learning and illustrate its behavior by numerical experiments.

2 Dynamics Matching by On-Line Learning

2.1 Dynamics matching as an optimization problem

It is necessary to consider a number of properties in designing a robotic system, as pointed out above. Here, we classify these properties into two groups, those that the system can adjust and those that the system cannot adjust, and name the former "adjustable parameters" $\boldsymbol{p}$ and the latter "constraints" $\boldsymbol{c}$. For example, the hardware specification (e.g., the maximum torque of a motor) and task requirement belong to the "constraints" because they cannot be changed by the system. On the other hand, the sensor parameters (e.g., sampling rate) and processing algorithm are the "parameters" because the system can select in an on-line manner.

Under this preparation, the dynamics matching can be formulated as a problem to find the optimal parameter value under given constraints so as to maximize the system performance.

Figure 1 schematically illustrates the structure of this optimization problem. The bottom plane in the figure represents the space of the system properties, which can be decomposed into the parameter space and the constraint space. The vertical axis represents the system performance P. When we specify the hardware specification and the task, the constraint is fixed to a specific value $\boldsymbol{c}'$. Then, our objective is to find the optimal parameter $\boldsymbol{p}^*$ in the cross-section of the performance surface $P(\boldsymbol{p},\boldsymbol{c})$ and the hyper-plane $\boldsymbol{c}=\boldsymbol{c}'$, that is, to find $\boldsymbol{p}^*$ defined as

$$\boldsymbol{p}^* = \operatorname*{argmax}_{\boldsymbol{p}} P(\boldsymbol{p}|\boldsymbol{c}').$$

Because the system does not know the relation between parameter $\boldsymbol{p}$ and performance $P(\boldsymbol{p}|\boldsymbol{c}')$, this problem cannot be solved by an analytic algorithm, such as the steepest ascent method. Instead, the system has to explore the parameter space and to estimate the performance in order to find the optimal parameter. Among several methods to realize such exploration, here we adopt a trial-and-error method. To be more specific, we will solve this problem using reinforcement learning.

2.2 Dynamics matching by reinforcement learning

Reinforcement learning (RL) is a learning algorithm to find an action-selection policy maximizing the expected reward received from the environment [8]. In order to apply this algorithm to the dynamics matching, we formulate this problem as shown in Figure 2.

First, a value of the adjustable parameter $\boldsymbol{p}$ and the corresponding performance $P(\boldsymbol{p}|\boldsymbol{c}')$ are regarded as a "state" and "reward," respectively. This comes from the fact that the

system tries to find the state (i.e., parameter) maximizing the total reward (i.e., performance). On this assumption, an "action" corresponds to a change in the parameter value. In this framework, therefore, the dynamics matching is realized by repeating the following procedure:

1. Read the current values of parameters (state observation),
2. Refer to the value function $Q(\boldsymbol{p}, \Delta\boldsymbol{p})$ and choose $\Delta\boldsymbol{p}$ according to some action-selection policy (action selection),
3. Change the parameter value according to $\Delta\boldsymbol{p}$ (state transition),
4. Perform the given task with the new parameter value, and evaluate the performance (reward acquisition), and
5. Update the value function V based on the received reward (value function innovation).

Although the general algorithm is very simple, the detailed implementation should be determined dependent on the individual problem.[1] In the next section, the authors take target-tracking as an example, and explain an implementation of the above algorithm.

3 Dynamics Matching in a Target-Tracking System

3.1 Target-tracking task

A target-tracking task is to find a control rule for a moving camera to follow a moving object in a space: The system has a camera mounted on a motor-controlled platform, and controls the motor so that the camera catches a target in its visual field. The authors pick up this task because it includes all of sensing, information processing and control processes and because it has been commonly used in discussing the performance of active vision system. For the sake of simplicity, the authors deal with 2D tracking where the target motion is restricted to a 2D plane (or the system deals only with the direction of the target), as illustrated in Figure 3.

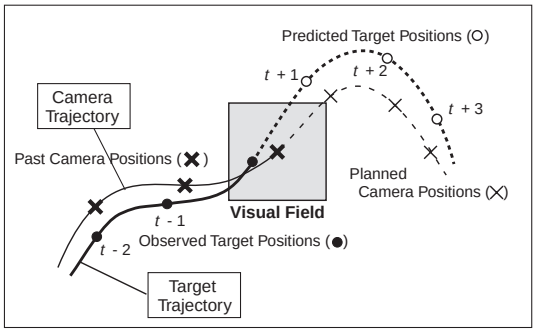


Figure 3: A 2D target-tracking task

Here, we assume that the target motion is so fast that the system cannot track it only by feedback control, and thus, the system utilizes the predictive control using an internal model of the motion. That is, the system predicts the future trajectory of the target and utilizes this prediction for planning to camera control.

3.2 Real-time requirement and trade-off in information processing

In Section 2.1, we classified the system properties into constraints and adjustable parameters. In the target-tracking task, the limitation of camera system (i.e., hardware specification) and the target motion (i.e., task requirement) correspond to the constraints while selection of processing algorithms corresponds to the adjustable parameter. In concrete, we take the visual field size of the camera (sensor's limitation) and the maximum speed of the camera mount (actuator's limitation) as the hardware specification. On the other hand, we think of several kinds of target motion as the task requirement.

As for the processing algorithms, we postulate "prediction of the target position" and "planning of the camera motion." The prediction is performed with an internal model of the target motion, which receives the current target position and velocity and outputs the target's future position. Here, it can be imagined

[1] Some people may suspect why reinforcement learning is used here, rather than a simple exploration method. This suspicion is reasonable because the system need not update the parameter once it reaches the optimal one. Though it may be "over specification" to use reinforcement learning in the current formulation, it is for a future extension treating the case that the system's characteristics change over time.

that more steps of prediction would reduce the risk to lose the target from the visual field in the future because the system can make use of those positions for motor planning. On the other hand, the planning of camera motion is to specify the next camera position based on the predicted target position and the actuator's limitation (i.e., maximum motion speed). It is performed in an iterative manner so that the tracking performance is improved with more steps of iteration.

It is obvious that the system can achieve better performance if it spends more time for both these processes. In actuality, however, the system has to finish these computations within a control cycle for real-time control of the camera movement. This means that there is a trade-off between the resources dedicated to the two processes: Too much prediction deteriorates the tracking performance while too much planning increases the risk to lose the target.

In order to treat this trade-off in a simple manner, we assume that the sum of the steps of prediction and of iteration is fixed to a constant (n), and use the number of prediction steps (k) as the index of the balance between two processes. Thus, k is regarded as the adjustable parameter of the system.

Since our algorithm treats the parameter as the "state" of the RL system, thus, the index k gives the "state." Then, "action" corresponds to an operation to update k. Here, we think of three actions: To increase, to decrease and to keep the value of k (Note that k takes a one-dimensional value). Finally, "reward" is determined by the mean square error between the observed target position $\boldsymbol{x}(t)$ and the predicted target position $\boldsymbol{x}_{\mathrm{p}}(t)$ over N time steps, [2] that is, $P = 1/N \sum_{t=1}^{N} |\boldsymbol{e}(t)|^2$, where $\boldsymbol{e}(t) = \boldsymbol{x}(t) - \boldsymbol{x}_{\mathrm{p}}(t)$. Specifically, the reciprocal of the average error (i.e., $1/P$) is utilized as the reward to the system so that the reward should be greater with less observation error.

The structure of the proposed system is summarized in Figure 4.

[2] Perhaps, it is also possible to measure the performance using the distance between the observed target position and the center of the visual field.

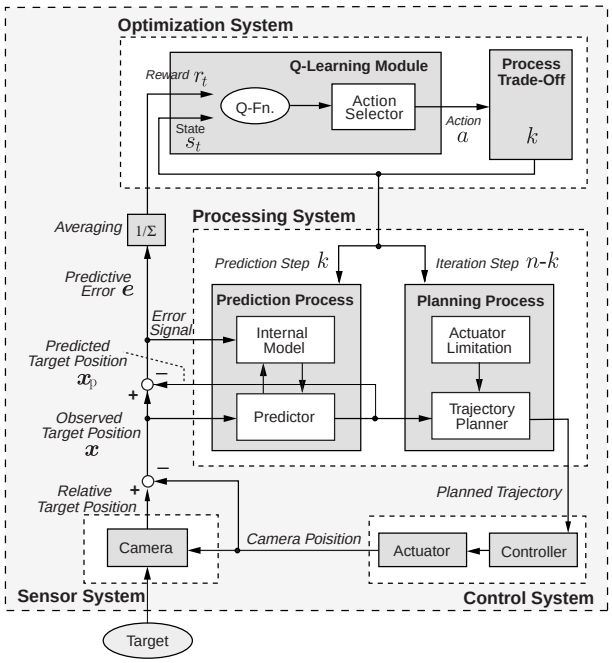


Figure 4: Structure of the proposed system

4 Numerical Experiment

The authors ran some numerical experiments to examine whether the above algorithm worked as expected. Below, we will show some preliminary results.

4.1 Assumptions

Since we would like to focus the discussion on adjusting the balance between prediction and planning, we first postulated that the camera sensor and actuator had ideal characteristics. In concrete, it was assumed that the delay in the camera's sensory processing and observation errors in camera measurement were negligible, and the control system achieved the planned camera position without delay or error.

Second, the authors considered the type of target motion (c), size of the visual field (d) and maximal speed of the camera actuator (v), as the constraints. These values were set by the experimenter, and never changed by the system. In the actual experiments, target motion was chosen from 3 ellipsoidal motions and one Brownian (i.e., completely random) motion. Visual field size d and maximum actuator speed v were chosen from 6 and 9 values, respectively.

Third, as for the internal model of the target motion, a second-order linear model written as

$$\boldsymbol{x} = A\boldsymbol{x} + \boldsymbol{b} \tag{1}$$

was prepared. The system parameters A and $\boldsymbol{b}$ took arbitrary values in the initial state, and were updated by an on-line supervised learning. In trajectory planning, we used an iterative method where the camera position became closer to the predicted position in a stepwise manner. Specifically, the trajectory was first set as a straight line toward the furthest predicted position (i.e., position at time $t+k$), and then updated so that the camera trajectory got close to the predicted target trajectory for the intermediate steps (i.e., positions at $t+1$, $t+2$, $\cdots$, $t+k-1$).

Finally, Q-learning[8] was adopted as the learning algorithm, where the value function was updated by

$$\Delta Q(k_t, a) = \alpha\left(r_t + \gamma \max_{a'} Q(k_{t+1}, a') - Q(k_t, a)\right), \tag{2}$$

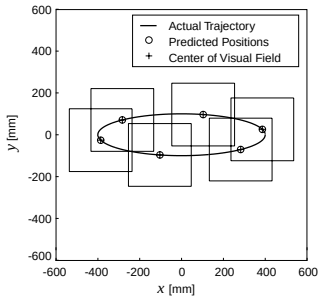
where k was 0, 1, 2, 3 or 4. It should be noted that the value function Q is formally dependent on the constraint $\boldsymbol{c}$. The above equation does not include $\boldsymbol{c}$ just because the constraint was fixed so far as the task and environment were not changed. On the other hand, the action selection probability $\Pr(a)$ was given by

$$\Pr(a) = \frac{\exp(\beta Q(k_t, a))}{\Sigma_{a'} \exp(\beta Q(k_t, a'))}, \tag{3}$$

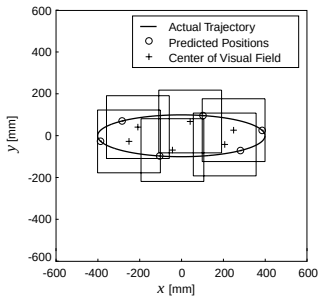
according to the Boltzmann rule. Parameters for Q-learning were $\alpha = 0.01$, $\beta = 0.05 \times$ (# of elapsed episodes), $\gamma = 0.999$, $N = 30$, # of episodes $= 100,000$.

4.2 Results and discussions

First, we show two examples of the system's behavior. Figure 5 presents the target position (circle), the center of the visual field (cross), and the visual field (square), in certain experimental sessions. The upper and lower panels show the results in a slow and a fast ellipsoidal motion conditions, respectively. In the slow-motion condition, the center of the camera field and the target almost coincided, implying



(a) Slow Target Motion

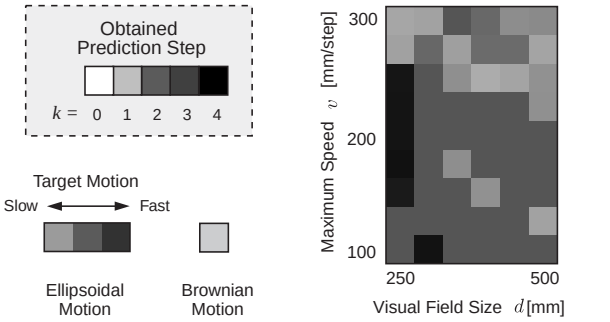


(b) Fast Target Motion

Figure 5: Examples of system behavior

that the system succeeded in tracking the target faithfully. In this case, the system found one prediction step ($k = 1$) as the best solution to the dynamics matching. In the fast-motion condition, in contrast, the camera caught the target in the "peripheral" of the visual field. In this case, the system chose two steps of prediction ($k = 2$), meaning that multi-step prediction was required to track the target: The system compensated the relatively low control ability by making use of multi-step prediction and its visual field.

The authors ran the experiments for a number of combinations of the constraints c, d and v, and examined the prediction steps after enough steps of learning. Figure 6 summarizes the result. Figure 6 (a) shows average prediction steps during the last 2,000 episodes for four types of target motion (i.e., three ellipsoidal motions and a Brownian motion) in the condition $d = 250$ mm and $v = 200$ mm/step. Darkness of each block represents the prediction steps (Darker blocks represent greater prediction steps. Darkness changes continuously though only discrete values are indicated in the scale). We can see that the block is darker for a faster target motion, indicating that the system chose more prediction steps (larger k) for



(a) Effect of Task Requirement (b) Effect of Hardware Specification

Figure 6: Acquired prediction steps

faster motion. On the other hand, the chose a smaller value of k for the Brownian motion. This is also reasonable because it is of no use to predict the target position when it moves completely at random.

Such solutions were dependent not only on the target motion but also the hardware specification: If the specification was different, different values of k should be chosen even for an identical target motion. Figure 6 (b) shows the result for 54 combinations of d and v with an ellipsoidal target motion. Although there are various local fluctuations, we can see a global tendency that the blocks become darker in the left lower direction, that is, for smaller values of d and v. This implies that the system chose a larger value of k for a smaller visual field and for a lower maximum speed, i.e., in the cases that the system had less tracking ability.

These results can be interpreted as follows. When the task requirement is enough low compared to the hardware limitation, it is a good strategy to plan a faithful camera trajectory and catch the target at the center of the visual field, without spending much time for predicting target trajectory. When the task requirement is severe, in contrast, the system should predict the target position for the further future (i.e., to increase k) with sacrificing faithful tracking, in order not to lose the target from the visual field. Above results show that the system found these solutions by learning.

5 Concluding Remarks

In this report, the authors introduced the concept of "dynamics matching" as a design principle of sensory-motor fusion systems and explained a method to realize it by learning. The aim of this report is to point out the importance of the dynamics matching and to show its feasibility by computer simulation. The method described here is just an example; there are many other possibilities to implement this idea in the real system, and the designers could choose appropriate methods according to the task and situation. Various case studies will enrich our methodologies to establish this design principle.

References

[1] W. T. Townsend and J. A. Guertin: "Teleoperator slave-wam design methodology," *Industrial Robot*, vol. 26, no. 3, pp. 167–177, 1999.

[2] M. Ishikawa, K. Ogawa, T. Komuro, and I. Ishii: "A CMOS vision chip with SIMD processing element array for 1ms image processing," *1999 Dig. Tech. Papers of 1999 IEEE Int. Solid-State Circuits Conference (ISSCC'99)*, pp. 206–207, Feb. 1999.

[3] A. Namiki, Y. Nakabo, I. Ishii and M. Ishikawa: "1ms sensory-motor fusion system," *IEEE/ASME Trans. Mech.*, vol. 5, no. 3, pp. 244–252, 2000.

[4] M. T. Mason and J. K. Salisbury: *Robot hands and the mechanics of manipulation*, MIT Press, 1985.

[5] M. W. Spong and M. Vidyasagar: *Robot dynamics and control*, John Wiley & Sons, 1989.

[6] R. Nigam and C. S. G. Lee: "A multiprocessor-based controller for the control of mechanical manipulators," *IEEE J. Robotics and Automation*, vol. RA1-4, pp. 173–182, Dec. 1985.

[7] G. C. Buttazzo: *Hard real-time computing systems — Predictable scheduling algorithm and applications*, Kluwer Academic, 1997.

[8] R. S. Sutton and A. G. Barto: *Reinforcement learning: An introduction*, MIT Press, 1998.