Learning Method for Multi-Controller of Robot Behavior*

Toshio FUKUDA** and Yasuhisa HASEGAWA**

In this paper, we propose a hierarchical behavior controller and a learning algorithm for the behavior controller which consists of several subcontrollers to indicate the desired trajectories for robot actuators. This algorithm selects the subcontroller which is not appropriate and needs to be tuned, by evaluating each subcontroller using multiple regression analysis based on previously obtained evaluation values. This process can reduce the learning iterations by avoiding attempts to tune good subcontrollers. The proposed algorithm is applied to the problem of selecting and tuning subcontrollers at the middle layer in the hierarchical behavior controller in order to compensate imperfect initial controllers. The hierarchical behavior controller is applied to the problem of controlling a seven-link brachiation robot, that moves dynamically from branch to branch like a gibbon, a long-armed ape, swinging its body like a pendulum.

Key Words: Learning Control, Fuzzy Set Theory, Motion Control, Parametric Excitation, Behavior Based Control, Multiple Regression Analysis, Reinforcement Learning, 7-link Brachiation Robot

1. Introduction

Intelligence can be observed to grow and evolve, both through growth in computational power, and through the accumulation of knowledge of how to sense, decide and act in a complex and changing world. There are four system elements of intelligence: sensory processing, world modeling, behavior generation and value judgment. Input to, and output from, intelligent systems are via sensors and actuators(i).

In this paper, we focus on the behavior-generating and the learning method for a behavior controller that is composed of multiple subcontrollers. Behavior is exhibited by a range of continuous actions that are performed by a robot with multiple degrees of freedom. To control such a robot, the controller is required to have the capability for managing multiple inputs and outputs. It would be very hard work to design such a controller, because we have to consider the nonlinear property among all inputs and outputs that are necessary to generate the behavior. Therefore, we set the same number of subcontrollers in the behavior controller as that of actuators which are necessary to generate that behavior. The subcontroller has an output variable and some input variables, and indicate the desired trajectory to an actuator according to the input variables. Therefore, we have to fabricate the subcontrollers one by one.

We consider three ways to design the subcontrollers that cooperatively make a robot perform a behavior. The first is that we design the behavior controller. The second is that any search methods or optimal methods, e.g., genetic algorithm, backpropagation methods, reinforcement learning and so on, search and obtain the behavior controller, thus maximizing a given evaluation function. The third is the combination of the above two ways.

We have some knowledge as to how some actuators of a robot should be controlled in order to make the robot perform a behavior. That knowledge is very useful in the design of behavior controllers; however, for a complicated system it is not enough to
complete the desired behavior. We have to find the controller in a trial and error process, and it requires tremendous iterations to find it. Therefore, the first way is inadequate to obtain the controller of the complicated system and the second way is likewise inappropriate because there is no search method that can find the optimal trajectories from multiple input and output spaces.

The third way is the most applicable for the complicated system. To fix the deficient behavior controller that is made from human knowledge, the search method or the optimal method is adopted. When fixing the behavior controller with multiple subcontrollers, we must determine which subcontroller requires tuning in order to improve the behavior. We use multiple regression analysis to determine the deficient subcontroller.

In the next section, the hierarchical behavior controller is described. In section 3, the proposed algorithm is described. In section 4, a fuzzy controller in the behavior controller is shown and in section 5, its learning algorithm is described. In section 6, a seven-link brachiation robot for computer simulation is explained. We design the initial controller in section 7. In section 8, the proposed method is applied to obtain a behavior controller for the seven-link brachiation robot in order to show its effectiveness.

2. Architecture of Hierarchical Behavior Controller

Most robots have multiple degrees of freedom, that are required to perform various kinds of objective behaviors or tasks. Therefore, multiple sensors and actuators are necessary for a robot to perform these complex behaviors. The robot’s controller, which deals with multiple input and output variables, also becomes complex and hard to design and adjust according to an environmental change. It is beneficial if the complex behaviors are divided into element behaviors and if behavior controllers for each element behavior are designed. In this case, a manager on the top level is necessary to control the behavior controllers. In this way, the hierarchical structure emerges like a human society. In this section, we show a hierarchical behavior controller for complex robot systems.

Fig. 1 Brachiation of a gibbon

Fig. 2 Architecture of hierarchical behavior controller

The hierarchical behavior controller shown in Fig. 2 consists of three layers: a planner, behavior controllers and feedback controllers. The planner in the top layer decides when and which behavior controllers in the middle layer should be used to make the robot perform a desired task. Multiple behavior controllers are in the middle layer. The behavior controller shown in Fig. 3 is a behavior generator that generates a sequence of actions and outputs the desired trajectories to actuators. The behavior controller consists of multiple subcontrollers that have multiple input variables and one output variable, as shown in Fig. 4. Each of the feedback controllers in the bottom layer makes an actuator follow the desired trajectory indicated by the behavior controller, using a linear or nonlinear feedback control method. When the desired trajectories from more than two behavior controllers are input to a feedback controller, the weighted mean value of them becomes the desired trajectory. The environment which is not measurable by the sensors, for example, the center of gravity and the figure of the object, is calculated in the sensor fusion section and then output to the planner and the behavior controller.

Some behavior may be acquired by trial and error learning, but more often it is acquired from a teacher, or from written or programmed instructions. This given knowledge can reduce the searching space and enable us to obtain a desired controller in fewer learning iterations. We prepare the same number of subcontrollers in the behavior controller as that of actuators which are necessary to generate that behavior and then design an initial controller in each subcontroller based on human knowledge. The explanation method of the initial controller is only to
Output a numerical value according to the robot states, for example, a mathematical function, cerebellum model arithmetic computer (CMAC)\(^{(2,3)}\), neural networks and fuzzy map. The fuzzy controller in the subcontroller for compensation of the initial controller is tuned by any learning method. The numerical value forwarded to the feedback controller in the bottom layer is the sum of the values from the initial controller and from the fuzzy controller.

A behavior is generated by the cooperation of these subcontrollers. However, the evaluation of robot behavior is expressed by a numerical value that is calculated with the given evaluation function after the behavior of the robot has finished. When no satisfactory evaluation value is obtained, the behavior controller will be tuned by a learning method. However, it is very difficult to determine which subcontroller in the behavior controller is inappropriate and requires tuning. We propose a method to find this inappropriate subcontroller, using multiple regression analysis in the next section.

3. Selection of Subcontroller

We installed some subcontrollers in the behavior controller, which generate a range of actions. Each of the subcontrollers outputs the desired trajectory of an actuator, according to multiple input variables. The desired behavior is performed by the cooperation of multiple subcontrollers in the behavior controller. When the performance of the robot is not satisfactory, the behavior controller should be tuned. However, because the behavior controller has multiple subcontrollers, the selection of the subcontroller that is inappropriate and that should be tuned is very difficult yet very important. We show an algorithm for selecting the inappropriate subcontroller.

The first step is to evaluate each subcontroller using multiple regression analysis based on the previously obtained evaluation values, and to choose a subcontroller to be tuned in the next trial with the probability of choice in proportion to the evaluation values of the subcontrollers. This can reduce the learning iterations by avoiding attempts to tune good subcontrollers. The method is explained below.

After tuning the subcontroller, \( n \), we calculate the improvement of the performance of the robot owing to its increment. The improvement value, \( f_x \), is

\[
    f_{x + 1} = f_{x} + f_{x + 1} - f_{x}.
\]

When \( f_{x} \) is the performance score calculated by the given evaluation function, \( f_{x}^{'} \) is the performance evaluated in the previous trial and \( x \) is the learning time required to tune the subcontroller. The evaluation value of the subcontroller is
\[ f_{m(x)} = f_{m(x-1)} + f(x) \]  \hspace{1cm} (2)

We assume that the evaluation value, \( f_n \), of the subcontroller is roughly proportional to its learning time, \( x \) (Fig. 5).

\[ f_{n+1} = a_1 x + a_0 \]  \hspace{1cm} (3)

The coefficient values, \( a_1 \) and \( a_0 \), are calculated by multiple regression analysis as follows.

\[ a_1 = \frac{S_{xz}}{S_x^2} \]  \hspace{1cm} (4)

\[ a_0 = \bar{f}_n - a_1 \bar{x} \]  \hspace{1cm} (5)

\[ S_{xz} \] is the covariance between \( x \) and \( f_n \),

\[ S_{xz} = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})(f_i - \bar{f}) \]  \hspace{1cm} (6)

and \( S_x^2 \) is the variance of \( x \).

\[ S_x^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \bar{x})^2 \]  \hspace{1cm} (7)

\( a_1 \) corresponds to the prospect of improvement of system performance when its subcontroller is tuned in the succeeding trial. The variance of \( f_n \), \( S_f^2 \), indicates the reliability of system improvement. The evaluation function of the subcontrollers therefore consists of two terms, \( a_1 \) and \( S_f^2 \), shown below:

\[ S_f^2 = \frac{1}{N-1} \sum_{i=1}^{N} (f_{n(i)} - \bar{f}_{n(i)})^2 \]  \hspace{1cm} (8)

\[ p_c = a_1 - \gamma S_f^2 \]  \hspace{1cm} (9)

where \( \gamma \) is a coefficient > 0.

A subcontroller tuned in the succeeding trial is selected according to the probability, \( g_c \), in the following equation, which is based on the evaluation values, \( p_c \).

\[ g_c = \frac{p_c}{\sum_c p_c} \]  \hspace{1cm} (10)

The flow chart of the proposed algorithm is shown in Fig. 6.

### 4. Fuzzy Logic Controller

In this section, we show the fuzzy logic controller which is for compensation of the initial controller in the subcontroller. The fuzzy logic controller is composed of three units: fuzzy rule, policy and inference.

![Flow chart](image)

**4.1 Fuzzy rule**

We use the simplified fuzzy rules, of which the consequent parts are represented numerically and the membership functions are based on the radial basis functions (RBFs)(a). A simplified fuzzy rule has three consequent values: a center value \( w_c \), a dispersion value \( \sigma_i \), and an internal reinforcement value, \( r_i \). The center value and the dispersion value are output to the policy unit.

The antecedent part is expressed as

\[ \mu_i(t_e) = \exp(-b_i \cdot (x_i(t_e) - a_i)^2) \]  \hspace{1cm} (11)

where \( i \) is the fuzzy rule number, \( j \) is the input number and \( a \) and \( b \) are the coefficients that determine the position and shape of each membership function.

The true value \( \mu_i \) of the \( i \)th rule is

\[ \mu_i(t_e) = \prod \mu_i(t_e) \]  \hspace{1cm} (12)

This true value of each fuzzy rule is output to the inference unit.

**4.2 Policy**

In this part, the consequent values \( w_i \) are determined, and used to infer the control values in the inference unit. In a searching mode, they are random
values determined according to a probability distribution expressed as

\[ g_i = \frac{1}{2\pi \sigma_i} \exp \left( -\frac{(x_i - \mu_{ei})^2}{2\sigma_i^2} \right). \] (13)

The search area depends on the dispersion value. When the dispersion value \( \sigma_i \) is large, a global search about the consequent value \( w_i \) is performed. When it is small, a local search is performed. When the obtained controller controls a robot after the searching mode, the consequent values are equivalent to the center values from the fuzzy rule unit.

4.3 Inference

In this inference unit, control values for a robot are inferred and output based on the true values from the fuzzy rule part and the consequent values from the policy part. The control value \( Y \) of fuzzy reasoning is calculated by

\[ Y(t) = \frac{\sum_i \mu_i(t) \cdot w_i}{\sum_i \mu_i(t)} . \] (14)

5. Self-Scaling Reinforcement Learning

The reinforcement learning method requires many iterations to realize the state–action map. One of its causes is the very low learning rate. If the range of reinforcement variation is known, the learning rate can be adjusted to prevent the weights from overshooting, which impedes system performance. When this is not the case, however, a very low learning rate is used, which results in a slow learning process. When a learning algorithm obtains much larger reinforcement than that in previous trials, it might be approaching the goal and should change its search strategy to a local search. Therefore, we propose a new algorithm to change the learning rate according to the reinforcement values obtained. This self-scaling method prevents the weights from overshooting and finds the optimal solution with fewer iterations.

After a robot executes a range of objective tasks or motions, the results are evaluated by a given evaluation function. The evaluated value is assumed to be a value of \( f \). We calculate the past performance \( \rho \) (Eq. (15)), which is the weighted mean of the past evaluation values. Furthermore, we compute an inner reinforcement value \( r' \) using Eq. (16) below.

\[ P(s) = \frac{\sum_i k^{s-i} f(i)}{\sum_i k^{s-i}}, \] (15)

\[ r' = f(s) - r(s-1), \] (16)

where \( s \) is the trial number and \( k \) is the positive weight \(<1\).

The range of reinforcement value \( r' \) is determined by the evaluation function which is arranged by the designer. Therefore, it is transformed into \( r \) with its range in \([-1, 1]\).

\[ r(s) = \frac{1 - \exp \left( -\frac{r'(s)}{\beta} \right)}{1 + \exp \left( -\frac{r'(s)}{\beta} \right)}, \] (17)

where \( \beta \) is a positive coefficient.

The reinforcement value is given after a robot executes a set of objective tasks or motions. Related works use fuzzy–neural networks or CMAC to compute the internal reinforcements, which also require tuning. Therefore, we assign an internal reinforcement to each of the fuzzy rule in the fuzzy rule unit explained in section 4.1. The internal reinforcement, \( r_i \), assigned to fuzzy rule, \( i \), is updated as follows.

\[ r_i \leftarrow r_i \left( 1 - \max_i \mu_i(t) \right) + \lambda (1 - r_i) \max_i \mu_i(t) \] (18)

where \( \lambda \) is a positive constant \(<1\), \( T_i \) is the time when the true value (Eq. (12)) is a maximum, and \( T^* \) is the time when the reinforcement is received.

The center values and the dispersion values of the consequent parts in the fuzzy rules used during robot motion are updated according to the internal reinforcement calculated by the above Eq. (18). The equations are

\[ w_i \leftarrow w_i + r_i \left( w_i - w_0 \right) \text{ if } r \geq 0, \] (19)

\[ \sigma_i \leftarrow \sigma_i (1 - r_i) \text{ if } r \geq 0. \] (20)

Only with the above two increment equations, would this algorithm converge into the local minimum around an initial position. Therefore, when no evaluation better than the average, \( \rho \), in Eq. (15), is received, that is, the reinforcement, \( r \), is less than zero, the dispersion values of the consequent parts are increased by

\[ \sigma_i \leftarrow \sigma_i + \alpha \text{ if } r < 0, \] (21)

where \( \alpha \) is positive. If an evaluation much larger than the average, \( \rho \), is obtained, the center values of the consequent values are updated to almost the same values as those used in this trial and the dispersion values become very small. In this case, a local search is performed until the received reinforcement becomes smaller than the average. On the other hand, if the obtained reinforcement is smaller than the average, the search range is widened.

In order to obtain a controller for a complicated real robot, we should make the robot perform the trials rather than make a modeled robot in computer simulations perform them. It is required to obtain a controller for the real robot in fewer iterations, because computer simulations can easily repeat the trials until a solution is found, but in the case of the real robot, durability and time (cost) are limited. In such a case, this algorithm shows the utility of finding feasible solutions that satisfy one order in fewer
iterations, instead of finding the best solutions.

6. 7-Link Brachiation Robot

Figure 7 shows the brachiation mobile robot (BMR) of a seven-link model used for computer simulation in this study. This robot is a mobile robot, which dynamically moves from branch to branch like a gibbon (Fig. 1), namely a long-armed ape, swinging its body like a pendulum. It has two arms, a body and a leg. The robot has seven degrees of freedom and six control inputs to actuators; elbows and shoulders of both arms, hip and knee. It has a redundant degree of freedom to move from branch to branch. On the tips of the two arms, the grips are set to catch horizontal parallel bars. In this study, the motion of the robot is assumed to be within a vertical plane. The equation of motion below is solved from Lagrange's equation of motion:

\[ r = \sum_{k=1}^{7} \left( J_a + m_s s_k + \sum_{l=1}^{7} m_l(l) \dot{\theta}_k + g M_k \sin \theta_k \right) \]
\[ + \sum_{k=1}^{7} \left( \sum_{j=1}^{7} P_{d,j} - \sum_{j=1}^{7} P_{a,j} \dot{\theta}_j \right) + \tau_{\text{friction}} \]

(22)

Table 1 lists the physical parameters used in the simulation, where \( l \) is the length of a link, \( s \) is the center of gravity, \( m \) is the mass and \( J \) is the moment of inertia. Table 2 lists the joint parameters, where \( D \) is the coefficient of viscous friction, \( |\max |\tau |\) is a limit

<table>
<thead>
<tr>
<th>Link</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4, 5, 6, 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l ) [m]</td>
<td>0.287</td>
<td>0.195</td>
<td>0.190</td>
<td>0.295</td>
</tr>
<tr>
<td>( s ) [m]</td>
<td>0.107</td>
<td>0.098</td>
<td>0.159</td>
<td>0.148</td>
</tr>
<tr>
<td>( m ) [kg]</td>
<td>4.0</td>
<td>1.5</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td>( \dot{a} ) [kg m²]</td>
<td>5</td>
<td>0.3</td>
<td>0.5</td>
<td>0.15</td>
</tr>
</tbody>
</table>

Table 2 Joint parameters of seven-link brachiation robot

<table>
<thead>
<tr>
<th>Joint</th>
<th>05</th>
<th>12</th>
<th>23</th>
<th>14, 16</th>
<th>45, 67</th>
</tr>
</thead>
<tbody>
<tr>
<td>( D ) [N m s rad⁻¹]</td>
<td>0.5</td>
<td>2.0</td>
<td>0.8</td>
<td>0.5</td>
<td>0.6</td>
</tr>
<tr>
<td>( \max</td>
<td>\tau</td>
<td>) [N m]</td>
<td>0</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>( \max d_{\theta,j} ) [deg]</td>
<td>-</td>
<td>150</td>
<td>0</td>
<td>-</td>
<td>150</td>
</tr>
<tr>
<td>( \min d_{\theta,j} ) [deg]</td>
<td>-</td>
<td>0</td>
<td>-150</td>
<td>-</td>
<td>-30</td>
</tr>
</tbody>
</table>

Fig. 7 Seven-link brachiation robot

---

**Fig. 8 Control architecture for seven-link brachiation robot**
of control torque for the driving motor, max|dc/dt| is an increase or a decrease limit of control torque, and maxθ and minθ are ranges of the angle between a vertical line and its link.

7. Design of the Initial Controller

The seven-link brachiation robot has eight actuators in which two actuators are for grips to hold or release the bar. The controller should cooperate with the eight actuators so that the robot can move from branch to branch. We prepare a planner, four behavior controllers and eight feedback controllers, as shown in Fig 8. The feedback controllers are PD controllers with respect to the desired angle and angular velocity from the behavior controllers.

7.1 Behavior controller

We assume that four behavior controllers are needed for the robot to move from branch to branch: amplitude control behavior, approach behavior, feedback behavior and hold and release behavior.

7.1.1 Amplitude control behavior This behavior controls the amplitude of the oscillation based on the parametric excitation theory, by changing the length of the pendulum according to the phase angle of the robot’s center of gravity. We use three strategies.

1. The elbow joint of the arm holding a branch is controlled such that it is kept vertical.
2. Link 6 is oscillated at the same cycle as that of the robot’s center of gravity.
3. Links 2, 3 and 7 are oscillated at half-cycle of that of the robot’s center of gravity so that the length of the pendulum should be changed.

7.1.2 Approach behavior Links 6 and 7 are controlled so that the grip of the free arm should come near the next branch.

7.1.3 Feedback behavior When the grip is near the branch, links 6 and 7 are controlled based on the PD control as follows in order to catch the branch:

1. τg is adjusted to control the angle between the grip and the target branch.
2. τg is adjusted to decrease the distance between the grip and the target branch.
3. The feedback gains are increased according to the distance between the grip and the target branch.

7.2 Planner

We designed a planner which decides when and which behavior controllers explained above should be used to make the robot move from branch to branch. The planner has four modes:

1. The amplitude control behavior for all links is activated until the phase amplitude of the robot’s center of gravity reaches a desired value.

Fig. 9 Stick diagram of seven-link robot locomotion (Circles show center of gravity of robot, 16 ms interval)

Fig. 10 Control Torque (Horizontal axis shows time [second]. Vertical axis shows actuator torque [N·m].)
8. Simulations and Results

The hierarchical behavior controller shown in Fig. 8 is applied to the motion control of the seven-link brachiation robot. We designed the initial controllers of body swing, of approach to a target bar and of feedback to a target bar based on the parametric excitation method and geometric analysis. The initial controller could not make the robot catch the target bar in one trial but could do so after a few swing motions as shown in Figs. 9, 10.

The evaluation functions, Eqs (23) and (24), are set to learn the behavior control of body swing by reinforcement learning. The motions which are obtained in 316 learning iterations are shown in Figs. 11, 12.

\[ f_a = 5 + a_1 \frac{1}{E} a_2 \theta s, \quad \text{if robot catches the bar,} \]

(b) Control Torque (Horizontal axis shows time [second]. Vertical axis shows actuator torque [N·m].)
Fig. 13 Learning iterations of each subcontroller

\[ f_m = \frac{1}{0.2 + \min|\alpha^n|} + \alpha_1 E - \alpha_2 \theta_n, \quad \text{otherwise.} \]  

The robot was able to catch the target bar in one trial. Each of the learning iterations is shown in Table 3 and Fig. 13. The number of iterations for the shoulder of holding arm is a maximum and for the elbow of the free arm, it is a minimum. The proposed algorithm reduced the number of learning iterations by about 18 percent of that of learning iterations when all subcontrollers are tuned one by one.

9. Conclusions

We proposed the architecture of a hierarchical behavior controller and the learning algorithm to refine a behavior controller. Using these methods, we obtained an appropriate behavior controller that can perform the desired behavior: the locomotion of a brachiation robot from branch and branch.

References