Applying homeostatic neural controller to multi-legged robot and adaptivity to novel disruptions

Hiroyuki Iizuka, Hiroki Nakai and Masahito Yamamoto

Department of Information Science and Technology, Hokkaido University
iizuka@complex.ist.hokudai.ac.jp

Abstract

A robot implemented with a homeostatic neural controller can adapt to disruptions that have not been experienced before. When novel changes occur, the homeostatic neural controller autonomously detects the disruptions from their behaviors and recreates new motions to achieve desired behaviors. In previous studies, the homeostatic neural controller has been only applied to a robot with a simple structure, i.e. wheeled robots which are controlled by adjusting the outputs of wheel motors. It is not clear whether the homeostatic neural controller can work properly when it is applied to a robot with a more complex structure. Therefore, we implement the homeostatic neural controller onto the multi-legged robot and investigate adaptivity. As a result, the robot with the homeostatic neural controller recreates new motion patterns to achieve a desired behavior after novel disruptions that break a leg.

Introduction

Our human and animals have an ability to adapt to various changes that occur internally and externally. The adaptivity is necessary to survive in the dynamic environment. The adaptation works towards not only the periodic or seasonal changes but also the changes that have not been experienced before. For example, people can adapt to the inverted glasses. Despite the fact that there was no chance to experience the inverted views in the long history of the human evolution, people can recover sensory motor coordination after they wear the inverted glasses for about a week and can catch an object properly or ride a bicycle (e.g., Miyauchi et al., 2004). If the adaptivity can be realized in the robots or software agents, the robots can behave more adaptively in the dynamic environment while achieving their objectives.

In the evolutionary robotics, artificial neural networks are usually used to generate adaptive and robust behavior. In order to obtain proper weight connections for adaptive behaviors evolutionary algorithms are used. It means that the desired behaviors are somehow encoded directly into the static weight connections. The obtained neural networks show robustness toward perturbations using generalization ability of the neural network. However, it is difficult to adapt to the relatively large perturbations like wearing inverted glasses described above because of the direct encoding of desired behaviors into the static weight connections.

A homeostatic neural controller proposed by Di Paolo (2000) is one of the possible implementations that make a robot to adapt to such a kind of disruptions. In his work, the homeostatic neural controller is realized with certain plastic rules and a fitness function to evolve the network to realize generating a desired behavior and maintaining homeostasis at the same time. The weight connections change only when it is required. The study shows that a wheeled robot can adapt to disruptions of exchanging sensor positions like the inverted glasses that have not been experienced before.

Because the homeostatic neural controller can show such an adaptation, there are some extended works to improve the adaptivity (Hoinville and Henaff, 2004; Williams, 2004, Williams, 2005; iizuka et al., 2013) or to show more cognitive behavior (Iizuka and Di Paolo, 2007; Wood and Di Paolo, 2007). In the most cases of the previous studies, a wheeled robot with simple light sensors is used as an embodied agent and a desired behavior is set to phototaxis or catching an object in order to test adaptivity of the homeostatic neural controller. However, the diversity of possible behaviors becomes very small when the sensors, motors and tasks are very simple. For example, in the case of wheeled robots, the behaviors can be roughly classified into three, i.e., approaching, avoiding or rotating around a light source. For this, any network weight connections result in one of those behaviors. The solution spaces of the neural network for approaching and avoiding are almost same and very large. More simply, when the sensors on the left and right are connected to right and left wheel motors, respectively, the robot can approach a light source. The robot avoids the light source when the sensors and motors on the same sides are strongly connected. Therefore, it is not very difficult to adapt to the disruption of exchanged sensors. It can be overcome by exchanging the connections from sensors to motors somehow. These settings are intentionally kept simple to evaluate adaptivity however it makes unclear if it is applicable to a more complex robot.

There is a study where the homeostatic neural controller is applied to a legged robot (Hoinville and Henaff, 2004). However, the robot only has a single leg. In the similar manner, the adaptation to the disruptions can be realized because of the simplicity of the robot. Therefore, it remains unclear whether the homeostatic neural controller can work properly when it is applied to a robot with a more complex structure. In this paper, the homeostatic neural controller is applied to a multi-legged robot and we investigate how the robot adapts to the different levels of disruptions.

The following sections are organized as follows. First, we describe the homeostatic neural controller and multi-legged
robot that we used. Next, the evolutionary algorithm to obtain the parameters of the network is explained. The adaptivity of the multi-legged robot with the homeostatic neural controller is shown in the simulation results.

**Homeostatic Adaptation Model**

The idea behind homeostatic adaptation is based on the ultrastable system proposed by Ashby (1960). The ultrastable system is a system—open to interaction with the environment—that plastically changes its configuration whenever stability is lost until it finds new internal dynamics which make the system stable under current conditions. Inspired by the idea of ultrastability, the homeostatic neural controller has local plastic mechanisms that change the incoming synaptic weights only when neural activations move out of a bounded region (called homeostatic region) defined in advance. Plasticity continues to operate until the neural activations return to the homeostatic region, resulting in a “stable” configuration in the sense that the network weights do not change further as long as the neural activation remains bounded. When this mechanism is implemented in a simulated robot evolved with a fitness function that simultaneously reward the desired behavior and the maintenance of neural activations within the homeostatic region, the neural controller give rise to the right sensorimotor coordination within a given environmental situation without any synaptic weight changes and the robot behave as desired. If the situation changes, such as an inversion of the visual field or breakdown of body parts, this causes a breakdown of coordination. Under these circumstances, the neural homeostasis of evolved robots also break down. When this happens, the local adaptive mechanism is activated until it find a new synaptic configuration that sustains activations within the homeostatic region. Under these conditions, evolved robots are also able to re-form behavioral coordination (even if they have not been trained to adapt to the induced perturbation).

In this paper, we use the basic homeostatic neural network proposed by Di Paolo (2000). There are different kinds of plastic rules to change the weight connections in the original model but a single rule is used for the simplicity. The homeostatic neural network is applied to a multi-legged robot.

**Robot and environment**

The robot consists of a main body and 5 legs which have two joints. The schematic view of the robot is shown in Fig. 1. One joint connects the main body with the leg and has a single degree of freedom (DOF). Another joint moves like a knee and also has a single DOF. The total number of DOF becomes 10. Every joint moves in the range [-\(\pi/2\), \(\pi/2\)]. The robot has the acceleration sensor and can detect the velocity of their movement. The robot is placed on a flat surface of the ground and there is no distraction like walls or objects. Our simulation is executed on PhysX, physics engine (NVIDIA).

![Figure 1: Schematic view of the legged robot. Top left: Top view of the robot which has 5 legs. Bottom: Side view of the leg parts. Each leg has 2 joints which can move in the range [-\(\pi/2\), \(\pi/2\)]. Top right: The robot implemented on physical simulator.](image)

The time evolution of the neural states is expressed as:

\[
\tau_i \dot{y}_i = -y_i + \sum_{j=1}^{N} w_{ij} z_j(y_j) + I_i, \tag{1}
\]

\[
z_j(y_j) = \frac{1}{1 + \exp[-(y_j + b_j)]}, \tag{2}
\]

where \(y_i\) represents the cell potential of neuron \(i\), \(N\) is the number of neurons, \(z_j\) is the neural activation, \(b_i\) is the bias, \(w_{ij}\) is the connection weight from neuron \(j\) to neuron \(i\), \(\tau_i\) is its time constant, and \(I_i\) represents the sensory input, non-zero only for an input neuron. There is a single input neuron in our neural network. The sensory input is calculated by multiplying a gain parameter (which is set to 3 in this paper) and the average velocity of the main body for last three seconds. There are 10 output neurons to control joints of the robot (2 joints x 5 legs = 10). Each output neuron is assigned to a single joint actuator. The neural activations of the output neurons are linearly mapped to the angle positions of the corresponding joint by the following equation.

\[
\theta_i = \pi \left( z_i(y_i) - \frac{1}{2} \right), \tag{3}
\]

where $\theta_i$ is the target position of the joint that neuron $i$ controls. Because $x_i$ takes $[0, 1]$, the range of the joint movements is $[-\pi/2, \pi/2]$.

The connection weights are initialized by the genetically obtained values at the beginning of a trial while a plastic mechanism allows for the lifetime modification of the connections. A homeostatic region is described as the finite, zero-value set of a plasticity function of the post-synaptic neural cell potential and bias. The homeostatic region is arbitrarily determined to follow the idea of the ultrastable system. The neural states in the most sensitive part of the sigmoid function is defined as the homeostatic region and too high or low neural states are defined as the out of homeostasis. The important thing is that the plasticity function must have two states (working/non-working) to realize homeostasis. The important thing is that the plasticity function that defines the homeostatic region (Fig. 2).

**Figure 2: Plasticity as a function of cell potential (bottom). The top graph shows the corresponding neural activations.**

Real-coded genetic algorithm is used to obtain the constant parameters for the homeostatic neural controller. All network parameters, i.e., initial weights $w_{ij}$, learning rates $\delta_{ij}$, time constant $\tau_i$, and biases $b_i$, are evolved. They are represented by a real-valued vector $(0,1)$ which is decoded linearly to the range corresponding to the parameters ($w_{ij}: [-8,8], \delta_{ij}: [-0.9,0.9], \tau_i: [0.25,2.5], b_i: [-6,6]$). Crossover and vector mutation operators, which add a small random vector to the real-valued genotype, are used (Beer, 1996).

The population size in the simulation is 240. The best 12 (5%) agents of the population are kept without changes as the elitism. 60 (25%) agents of the population are generated by randomly mating agents from the previous generation according to rank. The remaining 168 (70%) agents are mutated copies of agents from the previous generation, also selected according to their fitness rank.

The desired behavior for the robot is moving around. The task is to train the desired behavior while keeping the neural activations in the homeostatic regions. The fitness function evaluates how much the desired behavior is achieved ($f_d$) and how much the neural activations stay in the homeostatic regions ($f_h$) at the same time. Those two evaluations are multiplied. The actual fitness function is as follows.

$$ F = f_d \times f_h, $$

$$ f_d = \begin{cases} \frac{p_i - p_d}{12} & \text{if } d_i \leq 12, \\ 1 & \text{otherwise} \end{cases} $$

$$ f_h = \frac{1}{(T-10) \times N} \sum_{t=10}^{T} \sum_{i=1}^{N} h(z_i), $$

$$ h(z) = \begin{cases} 1 & \text{if } 0.119 \leq z \leq 0.881, \\ 0 & \text{otherwise} \end{cases} $$

where $p_i$ and $p_d$ represent the positions at the beginning and the end of a trial, $N$ is the number of neurons, $T$ is time for the trial, and $h(z)$ is a function that check if the neural activations are within the homeostatic regions. At the beginning of a trial, all the genetic parameters are set to the homeostatic neural network for the robot. The robot can move around based on the outputs of homeostatic neural controller for $T = 40$ in this paper) seconds as a single trial. $f_d$ simply measures the distance from the initial to the final position. $f_h$ calculates the time-average of the proportion of neurons that have behaved homeostatically. Because the behavior is not stable for the first 10 seconds, the last 30 seconds out of 40 seconds are used for the evaluation of the homeostasis. The fitness function is obtained by multiplying $f_d$ and $f_h$. By this, we can obtain the robot that can move around while keeping the neural activations in the homeostatic regions.

**Evolutionary Algorithm**

Real-coded genetic algorithm is used to obtain the constant parameters for the homeostatic neural controller. All network parameters, i.e., initial weights $w_{ij}$, learning rates $\delta_{ij}$, time constant $\tau_i$, and biases $b_i$, are evolved. They are represented by a real-valued vector $(0,1)$ which is decoded linearly to the range corresponding to the parameters ($w_{ij}: [-8,8], \delta_{ij}: [-0.9,0.9], \tau_i: [0.25,2.5], b_i: [-6,6]$). Crossover and vector mutation operators, which add a small random vector to the real-valued genotype, are used (Beer, 1996).

The population size in the simulation is 240. The best 12 (5%) agents of the population are kept without changes as the elitism. 60 (25%) agents of the population are generated by randomly mating agents from the previous generation according to rank. The remaining 168 (70%) agents are mutated copies of agents from the previous generation, also selected according to their fitness rank.

The desired behavior for the robot is moving around. The task is to train the desired behavior while keeping the neural activations in the homeostatic regions. The fitness function evaluates how much the desired behavior is achieved ($f_d$) and how much the neural activations stay in the homeostatic regions ($f_h$) at the same time. Those two evaluations are multiplied. The actual fitness function is as follows.

$$ F = f_d \times f_h, $$

$$ f_d = \begin{cases} \frac{p_i - p_d}{12} & \text{if } d_i \leq 12, \\ 1 & \text{otherwise} \end{cases} $$

$$ f_h = \frac{1}{(T-10) \times N} \sum_{t=10}^{T} \sum_{i=1}^{N} h(z_i), $$

$$ h(z) = \begin{cases} 1 & \text{if } 0.119 \leq z \leq 0.881, \\ 0 & \text{otherwise} \end{cases} $$

where $p_i$ and $p_d$ represent the positions at the beginning and the end of a trial, $N$ is the number of neurons, $T$ is time for the trial, and $h(z)$ is a function that check if the neural activations are within the homeostatic regions. At the beginning of a trial, all the genetic parameters are set to the homeostatic neural network for the robot. The robot can move around based on the outputs of homeostatic neural controller for $T = 40$ in this paper) seconds as a single trial. $f_d$ simply measures the distance from the initial to the final position. $f_h$ calculates the time-average of the proportion of neurons that have behaved homeostatically. Because the behavior is not stable for the first 10 seconds, the last 30 seconds out of 40 seconds are used for the evaluation of the homeostasis. The fitness function is obtained by multiplying $f_d$ and $f_h$. By this, we can obtain the robot that can move around while keeping the neural activations in the homeostatic regions.

**Evolutionary Algorithm**

Real-coded genetic algorithm is used to obtain the constant parameters for the homeostatic neural controller. All network parameters, i.e., initial weights $w_{ij}$, learning rates $\delta_{ij}$, time constant $\tau_i$, and biases $b_i$, are evolved. They are represented by a real-valued vector $(0,1)$ which is decoded linearly to the range corresponding to the parameters ($w_{ij}: [-8,8], \delta_{ij}: [-0.9,0.9], \tau_i: [0.25,2.5], b_i: [-6,6]$). Crossover and vector mutation operators, which add a small random vector to the real-valued genotype, are used (Beer, 1996).

The population size in the simulation is 240. The best 12 (5%) agents of the population are kept without changes as the elitism. 60 (25%) agents of the population are generated by randomly mating agents from the previous generation according to rank. The remaining 168 (70%) agents are mutated copies of agents from the previous generation, also selected according to their fitness rank.

The desired behavior for the robot is moving around. The task is to train the desired behavior while keeping the neural activations in the homeostatic regions. The fitness function evaluates how much the desired behavior is achieved ($f_d$) and how much the neural activations stay in the homeostatic regions ($f_h$) at the same time. Those two evaluations are multiplied. The actual fitness function is as follows.

$$ F = f_d \times f_h, $$

$$ f_d = \begin{cases} \frac{p_i - p_d}{12} & \text{if } d_i \leq 12, \\ 1 & \text{otherwise} \end{cases} $$

$$ f_h = \frac{1}{(T-10) \times N} \sum_{t=10}^{T} \sum_{i=1}^{N} h(z_i), $$

$$ h(z) = \begin{cases} 1 & \text{if } 0.119 \leq z \leq 0.881, \\ 0 & \text{otherwise} \end{cases} $$

where $p_i$ and $p_d$ represent the positions at the beginning and the end of a trial, $N$ is the number of neurons, $T$ is time for the trial, and $h(z)$ is a function that check if the neural activations are within the homeostatic regions. At the beginning of a trial, all the genetic parameters are set to the homeostatic neural network for the robot. The robot can move around based on the outputs of homeostatic neural controller for $T = 40$ in this paper) seconds as a single trial. $f_d$ simply measures the distance from the initial to the final position. $f_h$ calculates the time-average of the proportion of neurons that have behaved homeostatically. Because the behavior is not stable for the first 10 seconds, the last 30 seconds out of 40 seconds are used for the evaluation of the homeostasis. The fitness function is obtained by multiplying $f_d$ and $f_h$. By this, we can obtain the robot that can move around while keeping the neural activations in the homeostatic regions.

Figure 3: Fitness values of mobility ($f_d$) and homeostasis ($f_h$) during the evolution.

Simulation Result

The robot implemented with the homeostatic neural controller could achieve the task, i.e. moving around while keeping the neural activations in homeostatic regions. Figure 3 shows the fitness values of mobility and homeostasis during the evolution. The further investigation for adaptivity is performed with the best evolved robot obtained at the 120 generation.

Behaviors for the longer time span

During the evolution, each robot is tested for 40 seconds and the generated neural dynamics and behaviors are evaluated. The fitness values of the mobility and homeostasis reach the maximum value, which means that the robot can move around while keeping homeostasis for 40 seconds. However, it is not clear if the robot can behave properly for more than 40 seconds.

The robot is tested for 300 seconds to see the stability of the behaviors. Figure 4 shows the average velocity of the main body of the robot for 300 seconds. The robot can keep moving around even after 40 seconds while keeping the homeostasis. This result shows that the evolved robot has enough stability to maintain the structure and to generate the desired behavior.

Robustness against relatively small disruptions

In order to investigate adaptivity of the obtained homeostatic neural controller, a disruption which the robot has never experienced during the evolution is introduced. In the experiment, one of the knees gets stuck and the angle of the knee is fixated. The knee is the second joint from the body. Because the broken knee is just fixated at a certain angle and it is still available as a leg by moving the basal joint, the disruption is not large in terms of moving around.

The left graph in Fig. 5 shows the average moving velocity. For the first 40 seconds, the whole body parts move properly based on the neural outputs, however a single knee is broken at time 40. The neural output that controls the knee is just ignored after that. Before time 40, the robot can move around with about 0.7 m/s. After the disruption happens, the velocity gradually decreases. However, the disruption does not cause the serious damage to the robot. The velocity converges around 0.5 m/s and is stabilized. The right graph in Fig. 5 shows the neural activations of a selected neuron in the homeostatic neural controller. Before the disruption, the neural activation oscillates in the homeostatic region and keeps the neural structure (weights) constant except for the beginning of the simulation. After the disruption, the neural activation changes very slowly without moving out of the hostatic region. It means that the neural structure (weights) does not change because the plasticity mechanism does not work within the homeostatic region. The neural activations change in response to the new sensori-motor coordination formed under the knee-broken situation. The adaptivity toward the disruption is realized through robustness, which is the generalization ability as a usual neural network.

Figure 5: Left: Velocity of the robot. Right: Neural activation of a selected node. The green and pink lines shows the maximum and minimum values of homeostatic regions, respectively. A single knee joint is broken at time 40.

Figure 6: Average velocities of the robot (left) and neural activations (right). The two joints of the same leg get stuck at 40. The green and pink lines shows the maximum and minimum values of homeostatic regions, respectively.

Figure 7: Continuation of average velocities of the robot (left) and neural activations of a selected neuron (right) of Fig. 6. These are the results of the same experiment shown in Fig. 6 but they are shown in longer time span.

Figure 4: Average velocity of the robot that behaves for 300 seconds.

Figure 7: Continuation of average velocities of the robot (left) and neural activations of a selected neuron (right) of Fig. 6. These are the results of the same experiment shown in Fig. 6 but they are shown in longer time span.

Homeostatic adaptation against large disruptions

A relatively large disruption is introduced to investigate adaptivity of the homeostatic neural controller against the more serious damaged situation which the robot also has never experienced during the evolution. The single knee is broken in the previous section, however the knee and the basal joints of the same leg are broken at the same time here. It means that the first and second joint of the leg are fixated at the same time. Figure 6 shows the average velocity and the neural activations from 0 to 300 seconds. The disruptions happen at time 40 same as the previous section. Soon after the disruptions, the robot gets stuck and the moving velocity becomes very slow. The collapse of the desired behavior suddenly causes the change of the input to the homeostatic neural controller and the neural activation cannot be kept in the homeostatic regions due to the sudden changes of input. Then, the plasticity mechanism starts working and changes the network structure (weights). The plasticity keeps working until the network finds the new homeostatically stable states. Figure 7 shows the continuation of average velocity and the neural activations of Fig. 6 to see the adaptation by the plasticity mechanism in the much longer time scale. It shows that the robot behavior is recovered very slowly compared with the time scale in the previous section. The moving velocity returns to about 0.4 m/s. It is not full recovery because the velocity is around 0.7 m/s before the disruptions.

However, the neural activations also start returning to the inside of the homeostatic regions at the same time. Therefore, the neural structure can be kept after the activation returns. Consequently, the robot keeps moving around after the homeostatic adaptation.

During the homeostatic adaptation processes, internal representations of sensori-motor coordination must change. Figure 8 shows the neural states of homeostatic neural network in the different phases. The red lines in all graphs indicate the original neural states when they are formed without any disruptions and blue ones show the changes of neural states after the disruptions happen. Before the disruptions, the neural states changes periodically and form the limit cycle to generate stable moving-around behaviors. After the disruptions happen, the limit cycle is collapsed (Fig. 8(a)). The neural states apparently find a similar limit cycle close to the original one (Fig. 8(b)). However, the structure is gradually changing and cannot be fixed because the neuron values do not stay in the homeostatic regions (Fig. 8(b), (c), (d) and (e)). After time 5000, the neural states can keep them within the homeostatic regions and repeat the same neural changes (Fig. 8(f)). At that time, the behaviors are also sufficiently good and the behavior can be repeated with the new sensori-motor coordination.

The motion patterns are also shown in Fig. 9. The dynamics is not as simple as the neural activations because the body interacts with the environment. After the disruption occurs, the motion behavior breaks down ((a) and (b)), which leads to the slowdown of the movements. The behavior dynamics form a different attractor when the movement of the robot become close to zero ((d) and (e)). After recovering the movements, the behavior is stabilized and kept (f).

To show how much the homeostatic adaptation contributes to the behavior recovery, the plasticity mechanism is removed when the disruptions happen and we see how much the behavior can be recovered from the serious damaged situation without any neural structural changes. This is adaptation without the plasticity mechanism, that is, only robustness works. The velocity after the disruptions is about 0.06 m/s. Compared with the velocity when the plasticity mechanism works, it is much lower. The velocity is not improved without the plasticity mechanism. It means that the behavior recovery for the large disruptions is realized by the homeostatic adaptation.

Figure 8: Neural activations of selected three neurons. The data is plotted during different period shown at the bottom of each graph. Time is corresponding to time in Fig. 7. The red lines in all graphs indicate the neural states from 20 to 35 where the disruption is not introduced yet.

Not always possible to adapt

The previous sections show the processes of adaptation to the disruptions when it is successfully adapted. The homeostatic neural controller cannot always adapt to the disruptions. As our real life is, if the changes are too large or critical, the robot cannot recover the desired behaviors. In fact, there are some cases that the robot just stop moving when different pairs of joints get stuck in our simulation. The results of the all cases of different pairs of joints are shown in Fig. 10. The graph indicates the minimum and final average velocities after a pair of joints get stuck. The final average velocities are calculated over 1000 seconds after time passed long enough since disruption happens. The results are classified into three groups. The first group is that the robot becomes unable to move once and stops in the end (Fig. 10(A)). The second is that the robot becomes unable to move once however recovers the movements (Fig. 10(B)). The last group is that the final and minimum velocities are almost same and the robot keeps moving before and after the disruption (Fig. 10(C)). The group (A), (B), and (C) correspond to no adaptation, homeostatic adaptation, and robustness. The robustness and homeostatic adaptation accommodate 60% and 33% of disruptions, respectively. In the remaining 7% cases, the robot cannot adapt.

Discussion

Adaptation happens in the following manner. To realize the desired behavior (i.e. moving around) is associated with the internal homeostasis by the evolutionary algorithm. Because the input of the homeostatic neural network is the velocity of the main body, the input becomes high when the desired behavior is achieved. It means that the input values are always high when the internal homeostasis is realized. In other words, to maintain the homeostasis requires the high input values. When the disruptions occur and the behavior is damaged, the velocity becomes slow and the input value decreases. If the robustness cannot accommodate the changes, it causes the breakdown of the internal homeostasis. The plasticity starts working because of the breakdown of the homeostasis until it finds a new structure that can get enough input values to maintain homeostasis, which leads to the desired behavior.

Our results show that there are possibilities that the multi-legged robot implemented with the homeostatic neural controller can recover the desired behavior when novel disruptions happen. In the homeostatic regions, the neural dynamics also show robustness to the disruptions. If it is not accommodated by the robustness, the homeostatic neural controller start changing network structures. The multi-legged robot have the robustness and homeostatic adaptation at the same time.

However, the recovery by the homeostatic adaptation were sometimes not good enough and sometimes no adaptation occurred. We discuss about improvements. Firstly, regarding to the plasticity mechanism, the method that we used here is based on Di Paolo’s homeostatic adaptation model and Hebbian type of learning is used. Although the parameters of the methods were obtained by the genetic algorithm, the
selection pressure for these parameters must be weak because those parameters work only when the internal homeostasis breaks down. It may be useful to give a specific serious damage to the robot during evolution in order to train these parameters and to adapt to the different disruptions that have not been experienced. This is also related to the learning speed. In fact, the learning speed is very slow in the current setting. To increase the learning rates would improve speed however it is connected with the whole adaptation dynamics as explained in this paper. It should be self-organized during evolution.

Secondly, the relation between the input values and the desired behavior is rather direct. The direct relation makes adaptivity poor. The advantage of the homeostatic neural controller is that the whole mechanism including the evaluation of the desired behavior is distributed in the network. Nothing is centralized. The learning mechanisms (e.g. backpropagation or genetic algorithm) can be applied to adjust the weight connections with the input signal instead of using homeostatic mechanisms. However, if the learning and evaluation mechanism has a damage, the system will not work. In the case of the homeostatic mechanism, those mechanisms are not centralized. From this point of view, the input value of the current setting is not enough. When the sensor is broken to measure the velocity, the system will not work properly. This should be improved for the multi-legged robot.

**Conclusion**

In the conventional studies, the homeostatic adaptation was only applied to a wheeled or simple robot but we implemented the homeostatic adaptation model on the multi-legged robot. In the same manner as the conventional method, the homeostatic neural network was evolved to perform a desired behavior while keeping the neural states in the homeostatic regions. Because we used more complex structured robot, different levels of disruptions could be considered. When the small disruption was introduced, the robot with the homeostatic neural network kept moving around without any structured changes of the network. However, the homeostatic adaptations happened when the relatively big disruptions were given and the robot could recover the desired behavior. Our result shows that the homeostatic neural network can work properly even in the more complex robot and the problems of applying homeostatic adaptation was discussed.

**References**


