IEEE Trans. on Systems, Man, and Cybernetics Part B: Cybernetics, Vol.26, No.3, pp.421-436, 1996.

Model-Based Learning for Mobile Robot Navigation from the Dynamical Systems Perspective

Jun Tani
Sony Computer Science Laboratory Inc.
Takanawa Muse Building, 3-14-13 Higashi-gotanda,
Shinagawa-ku,Tokyo, 141 JAPAN
tani@csl.sony.co.jp, http://www.csl.sony.co.jp/person/tani.html
(Published in IEEE Trans. System, Man and Cybernetics (Part B),
Special Issue on Learning Autonomous Robots,
Vol.26, No.3, 1996, 421–436)

November 22, 2004

Abstract

This paper discusses how a behavior-based robot can construct a "symbolic process" that accounts for its deliberative thinking processes using models of the environment. The paper focuses on two essential problems; one is the symbol grounding problem and the other is how the internal symbolic processes can be *situated* with respect to the behavioral contexts. We investigate these problems by applying a dynamical system's approach to the robot navigation learning problem. Our formulation, based on a forward modeling scheme using recurrent neural learning, shows that the robot is capable of learning grammatical structure hidden in the geometry of the workspace from the local sensory inputs through its navigational experiences. Furthermore, the robot is capable of generating diverse action plans to reach an arbitrary goal using the acquired forward model which incorporates chaotic dynamics. The essential claim is that the internal symbolic process, being embedded in the attractor, is grounded since it is self-organized solely through interaction with the physical world. It is also shown that structural stability arises in the interaction between the neural dynamics and the environmental dynamics, which accounts for the *situatedness* of the internal symbolic process. The experimental results using a mobile robot, equipped with a local sensor consisting of a laser range finder, verify our claims.

1 INTRODUCTION

In recent research on autonomous robots, the majority of interest has shifted from the AI-based approach to so-called behavior-based robotics [5, 31]. A consensus, that the emphasis on deliberative computation and explicit representation of AI does not provide satisfactory solutions to the scale-up of toy-problems to real-world complex problems, has encouraged research in behavior-based robotics. The behavioral functions of these robots are simple, but their reactive-type action-selection mechanism makes them suitable for real-world environments. Furthermore, the reaction against explicit coding schemes in AI resulted in initiating a new approach of so-called adaptive behavior [33]. Adaptive behavior focuses on how an "animal" or "animat" can attain an intrinsic function that coordinates perception and action solely based on its own behavioral experience. It has been demonstrated that an autonomous robot can acquire simple behavioral functions such as collision-avoidance, wall-following, or road-following by various adaptation methodologies including neural learning [28, 37], genetic programming [26], reinforcement learning [8, 22] and others. These approaches are equivalent in a general sense in that the aim of the adaptation is to self-organize a direct state-action map which allows situated behaviors of the agent.

However, there is a suspicion that the absence of representations (modeling of the world) in these approaches might restrict the robot's progress in emulating the equivalent cognitive abilities of animals or humans. An intelligent robot should have a certain "symbolic process", with which it is capable of simulating its own potential action plans flexibly using its internal model, before choosing a course of action. Such high-order cognitive activities stand on the combinatorial power of symbol systems [15], which enable the robot to conduct certain grammatical manipulations of knowledge. We consider that the Deliberative Thinking Paradigm of AI itself is not misleading at all. However, the paradigm faces two essential problems. One is the "symbol grounding problem" as Harnad [15] has discussed, namely "*How can the semantic interpretation of a formal symbol system be made intrinsic to the system, rather than just parasitic on the meanings in our heads ?*" This problem asks us how to build the internal representations, and how to use them without generating fatal gaps from the physical data obtained from the environment. The other problem is how the symbolic processes can be *situated* in the current behavioral context—i.e. how the robot can recognize its situation, which has been determined from the history of its interaction with the environment. The aim of this paper is to provide an answer to the above problems by presenting our novel approach of *model-based learning* in the domain of mobile robot navigation.

The problem of mobile robot navigation has been studied using explicit global representation. More specifically, a robot builds an environmental map, represented in global coordinates, by gathering geometrical information as it travels [2, 10, 13]. Though a variety of methodologies have been proposed in this context, potential problems still remain especially in robot localization. The robot's position is mathematically identifiable by comparing the current local sensory input (typically range image and dead reckoning) with the global map. This process is not always robust enough for realistic "noisy" environments. There is a gap between what the global map represents and what the robot senses in the real environment.

There have been efforts [27, 48] to construct a direct sensor-situation (position) map by utilizing the idea of Kohonen's Self-Organizing-Map [25]. Although this approach,

which is based on a statistical clustering technique, might be able to generate an intrinsic representation of the sensory-situation association, it seems to have potential limitations in its localization capability. The position cannot always be identified solely from the current sensory input since the sensory input could be the same in different locations. In order to avoid this problem, Krose [27] utilizes global orientation information from compass readings, and Zimmer [48] utilizes dead reckoning information. We, however, speculate that the current situation or position may be identifiable without introducing global information, but instead from context-dependence: by utilizing the past sensory sequence acquired during its travel. The problem of how to construct context-dependent representation seems to be important in this instance.

Kuipers [29], Mataric [32] and others have developed an alternative approach using landmarks, which is aimed at achieving behavior-based local representation. A mobile robot acquires a graphical representation of landmark types as it moves in an environment. This representation is equivalent to a finite state machine (FSM), and comprises a topological modeling of the environment. Thus, it represents the grammatical structure of the environment with obstacles. In navigation, the robot can identify its topological position by anticipating the landmark types in the FSM representation. Although this behavior-based local representation scheme is likely to improve robustness in navigation, its stability is not clear if an erroneous landmark-matching happens to take place: a FSM would halt if fed an illegal symbol. This navigation strategy is susceptible to such a catastrophe if the landmark type is misread. Although robustness can be enhanced through improving the landmark detection scheme by combining, for example, global positioning (as conducted in [32]) or other sensor-fusion techniques, it would remain limited as long as the model is represented symbolically.

In this paper, we focus on the dynamical systems approach [3, 19] as an alternative, with the expectation that its language can be utilized to build an effective representational and computational framework for behavior-based robots. Although one may speculate that highly analog representations by dynamical systems lack the combinatorial power of symbolic systems, a recent study of symbolic dynamics [7] showed otherwise. Crutchfield [7] clarified the relations between formal language [16] and nonlinear dynamical systems, in which he showed that chaotic dynamics correspond to a regular or higher language in the language hierarchy [16]. Furthermore, experimental studies on a recurrent neural network (RNN) [36, 44] demonstrated that the network can learn primitive grammatical descriptions from examples by generating deterministic chaos. Therefore, it is quite plausible that symbolic processes, which account for the cognitive tasks of planning or *mental simulation*, can be constructed as being embedded in chaotic dynamical systems.

Upon describing the internal symbolic processes of the robot using the language of dynamical systems, we become able to analyse its interactions with the physical environment. We focus on the coupling between the internal dynamics and the environmental dynamics made through the sensory-motor feedback, then we investigate how coherence is made between these two dynamical systems. We speculate that a key to solving the symbol grounding as well as the *situatedness* problems lies in the qualitative understanding of the dynamical mechanism of this coherence. Our analysis of this mechanism will explain the dynamical structure that is essential to building a behavior-based robot that is characterized by its *model-based* intelligent activities.

The remainder of this paper is organized as follows. Section II defines the navigation problem which we study in this paper. In order to clarify the navigational conditions,

the basic navigation architecture is introduced in Section III. Section IV presents our formulation of *model-based learning* using the forward modeling scheme [19, 21, 46], which is implemented using a recurrent neural network (RNN) [18, 35, 36] architecture. We describe the application of chaotic dynamics to the planning process and discuss its qualitative meaning from the view point of deterministic dynamics. Section V presents a series of experiments using the mobile robot in order to test our approach. Section VI shows an analysis of the dynamical structure that accounts for the mechanisms of *situatedness*. Section VII discusses and summarizes the themes of this paper.

2 THE NAVIGATION PROBLEM

Before presenting the detailed formulation, we describe the navigation problem on which this paper focuses. We consider that there are two types of approaches to navigation-learning which are fundamentally different in how the navigational knowledge is represented and how it is utilized. The first type is *skill-based learning*. In this approach the robot learns skills for achieving a fixed goal such as a homing or cyclic routing task. The robot has to go home or move into a predetermined cyclic loop, starting from an arbitrary position in the adopted workspace. Our previous work [42, 43] showed that the robot can achieve these tasks by acquiring an adequate state-action map (a map of sensory-based internal states to motor commands). The second type is *model-based learning*, which is the main subject of this paper. The advantage of *model-based learning* is that the process of planning using the internal model enables the robot to adapt flexibly to different goal tasks. Our *model-based learning* approach, applied to a real mobile robot, assumes the following conditions and specifications.

- The robot cannot access its global position, but it should navigate based on its local sensory (range image) input.
- There are no explicit landmarks accessible to the robot in the adopted workspace.
- No *apriori* knowledge of the workspace geometry is given. The robot should obtain it from its travel experience.
- The robot should be capable of planning the shortest route to an arbitrary position.
- The robot should be robust against temporary disturbances including noise and temporary geometrical changes in the workspace.

3 NAVIGATION ARCHITECTURE

The *YAMABICO* mobile robot [17] was used as an experimental platform. Figure 1 shows a picture of *YAMABICO*. The robot can obtain range images by a range finder consisting of laser projectors and three CCD cameras. The ranges for 24 directions, covering a 160 degree arc in front of the robot, are measured every 150 milliseconds by triangulation. The specifiable range is 0.2 m to 5.0 m. The main navigation level computation is conducted on a host computer via wireless communication. The robot maneuvers by differentiating the rotation velocity of the left and right wheels, and it normally moves with a speed of 0.3 m/s.

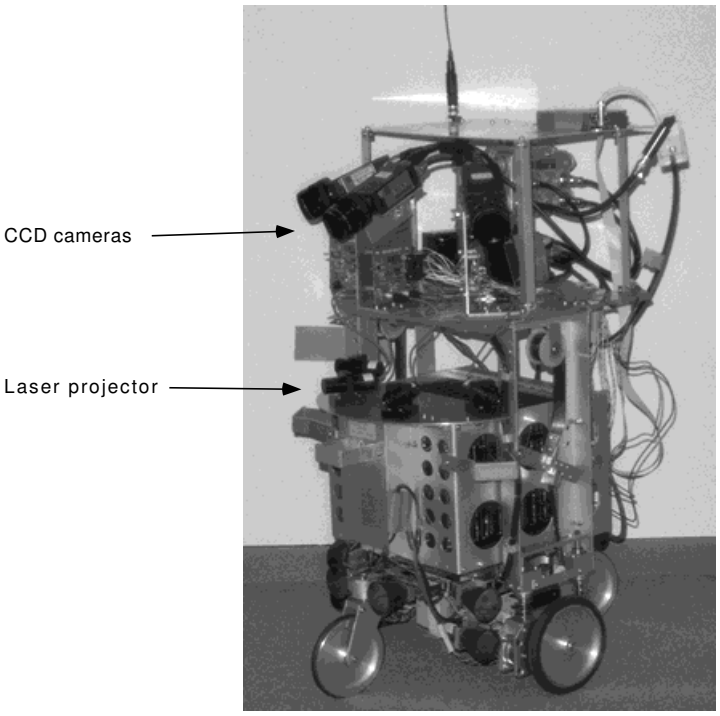

Figure 1: The *YAMABICO* mobile robot equipped with a laser range sensor.

In our formulation, maneuvering commands are generated as the output of a composite system consisting of two levels. The control level generates a collision-free, smooth trajectory using a variant of the potential field method [24], while the navigation level directs the control level in a macroscopic sense, responding to the sequential branching that appears in the sensory flows. The control level is fixed; the navigation level, on the other hand, can be adapted through learning.

Firstly, let us describe the control level. The robot can sense the forward range readings of the surrounding environment, given in robot-centered polar coordinates by r_i $(1 \leq i \leq N)$, as shown in Fig. 2. The angular range profile R_i is obtained by smoothing the original range readings through applying an appropriate Gaussian filter. The maneuvering focus of the robot is the maximum (the angular direction of the largest range) in this range profile. The robot proceeds towards the maximum of the profile (an open space in the environment). This control scheme is implemented as follows:

$$V_{dif} = k_p \cdot \theta_f \tag{1}$$

where V_{dif} is the differential rotational velocity between the left and right wheels, θ_f is the angular displacement of the focus point from the center, and k_p is a constant gain.

The navigation level focuses on the topological changes in the range profile as the robot moves. Fig. 3 (a) shows a robot trajectory measured in an experimental workspace; Fig. 3 (b) shows the corresponding temporal sensory flow. After starting from the "start", the robot moves through the workspace by tracking the maximum. The corresponding range profile contains a single maximum. (In the shaded sequence, the middle part, corresponding to a larger range, is darker. The sides having a smaller range are brighter.) As the robot moves through the workspace, the profile gradually changes until another

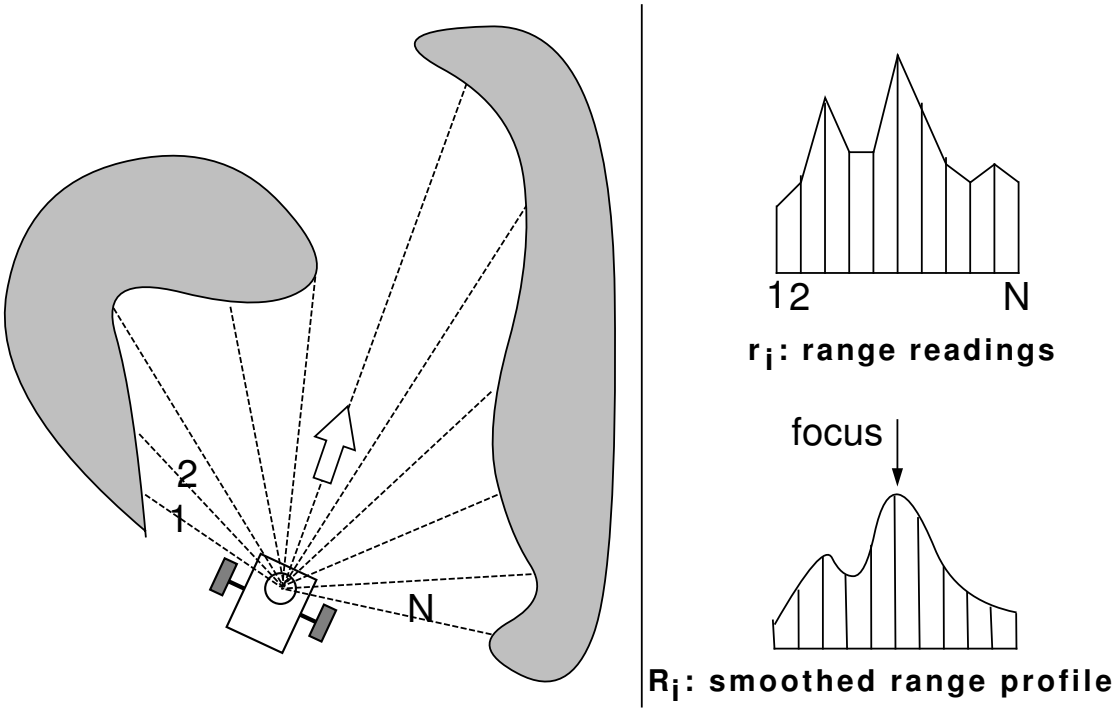

Figure 2: The Range profile is obtained from the frontal side of the robot. The robot moves by tracking the maximum in the range profile.

local maximum appears when the robot reaches location (1) and it perceives a new open area to the left. At this moment of branching the navigation level decides whether to transfer the focus to the new local maximum or to remain with the current one. (In this implementation, the occurrence of branching is not confirmed until a time lag $\triangle T$ so that the sensing of branching may not be perturbed by noise.) In this example, the robot decides to transfer the focus to the new maximum and proceeds to the left. In the same manner, the decision process is repeated at point (2) where the focus transfers to a new local maximum on the left-hand side, and at point (3) where it stays with the current branch by traveling forwards, and at (4) where it transfers to a new branch to the left. These binary branching decisions generate the trajectory shown. In some instances, the robot maneuvers into a concave dead end from which the robot cannot escape with the above maneuvering scheme. To avoid this, the navigation scheme is enhanced as follows: when the robot comes extremely close to a dead end, the robot is instructed to turn through 180 degrees. Thereafter it proceeds as usual. (The concave dead end is easily detected by monitoring the range values in the forward direction with respect to a certain threshold.) Hereafter, we denote this dead end point as the "terminal point".

Although the proposed binary branching scheme simplifies the problem of navigation substantially, a technical difficulty arises when multiple branching situations take places– i.e. more than two new branches are sensed simultaneously. In such singular situations, the robot takes the right most (or the left most) new branch as the new one, thereby losing the opportunity to select other new branches. Therefore, with the current navigation scheme, there are cases in which the robot cannot take all the possible paths allowed by the geometry of the workspace.

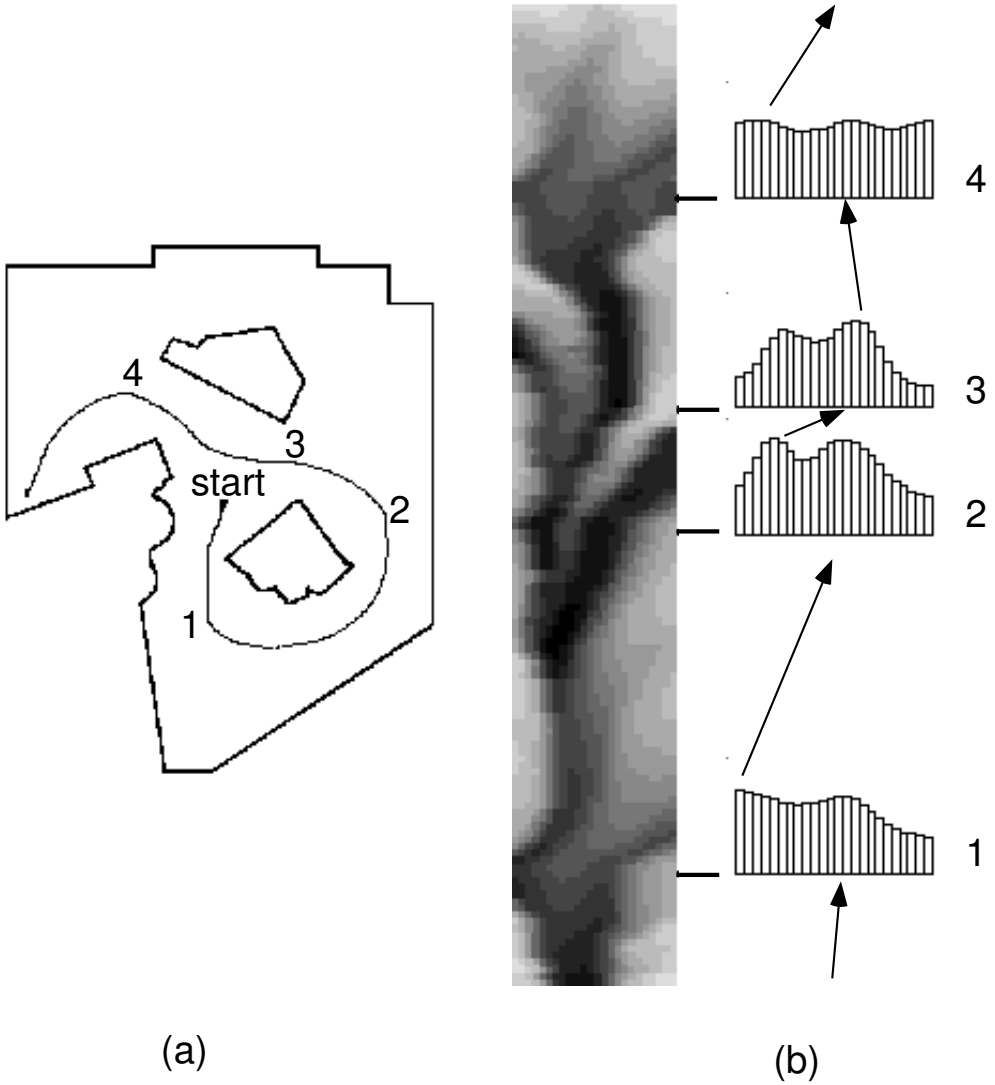

Figure 3: An example of travel and its associated sensory flow.

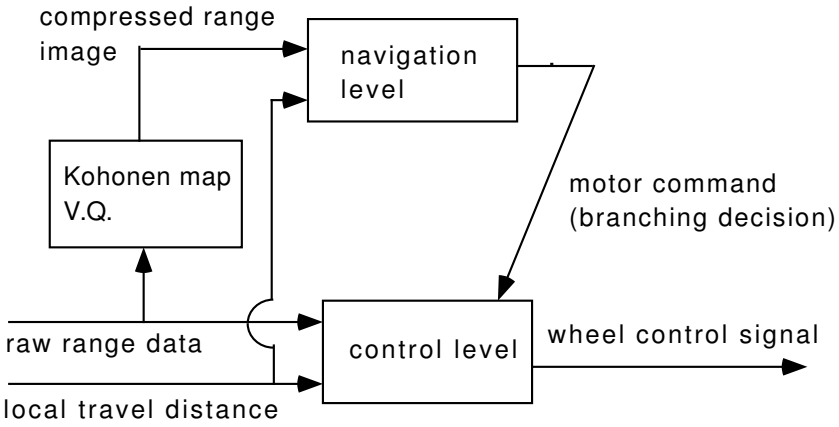

Figure 4: The navigation architecture employed here, comprising the control and navigation levels.

The navigation level utilizes two types of sensory inputs at branch or terminal points. Those are the current range image and the local travel distance from the previous to the current point as measured by the wheels' rotational encoders. The filtered range profile consists of $N = 24$ range values. Since the pertinent information in the range profile at a given moment is assumed to be only a small fraction of the total, we employ a vector quantization technique, known as the Kohonen network [25], so that the information in the profile may be compressed into specific lower-dimensional data. The Kohonen network employed here consists of an l-dimensional lattice with m nodes along each dimension (l=3 and m=6 for the experiments with *YAMABICO*). The range image consisting of 24 values is input to the lattice, then the most highly activated unit in the lattice, the "winner" unit, is found. The address of the winner unit in the lattice denotes the output vector of the network. The virtue of this scheme is that the original topological structure of the input space is well-preserved in the output space, which assures the generality of the transformation because the output vector is a smoothly-varying function of the input profile. Although the real range image exhibits its stochastic distribution at each branch point, the Kohonen network is capable of clustering such noisy range image information with a topology-preserving map that is self-organized in the off-line learning phase.

The navigation architecture presented in this section is summarized in Fig. 4. In this architecture, the navigation problem is simplified to one of determining the branching sequence. Hereafter, we focus on how the navigation level achieves this. We use the terms "motor command" and "motor program" to indicate a branching decision and a sequence of branching decisions, respectively.

4 MODEL-BASED LEARNING

This section describes how the robot learns the internal model of the environment, and how such a model can be utilized to generate navigation plans—i.e. what motor programs are needed in order to reach a given goal. Here, we attempt to apply the scheme of forward modeling [19, 21, 46] to the problem. Before presenting our formulation, we show the outline of the robot's operation in *model-based learning*.

First, the robot goes through the learning phase. The robot wanders around an adopted workspace by collision-free maneuvering, making each branching decision at random. Meanwhile the robot collects the sensory-motor sequence—i.e. the sequence of branching decisions and the resulting sequence of sensory input perceived at the branch points. Thereafter, the robot attempts to acquire a "topological" model of the workspace in terms of a forward model through off-line neural learning. After learning has taken place, we examine how accurately the robot has learned the model of the workspace by measuring the robot's capability in lookahead prediction. If the learning of the model is found to be insufficient, the above learning process is repeated through sampling more data.

After the learning phase is completed, the navigation phase can be initiated. In the navigation phase, the robot conducts plan-based navigation. A branch or terminal point is selected as a goal position, which is specified to the robot by using the sensory input that would be obtained at that position. The robot has to generate a motor program (the sequence of branching decisions) to reach the goal by the shortest route.

After completing the off-line learning, initially the robot cannot recognize its situation/position and therefore it cannot initiate any planning activities. The problem to consider is how one can situate the robot in the environment. In addressing this problem, we use two distinct operational modes, namely the open-loop mode and the closed-loop mode, and introduce a switching mechanism between them. First, the robot travels around the workspace receiving sensory input at each branch. In the meanwhile the robot begins to recognize the current situation/position from what it has sensed during its travel. We call this mode the "open-loop mode" since the internal computation of the robot is coupled to the environment via the sensory-motor loop. When the robot becomes *situated*, it is ready for planning activities. By shutting off the sensory-motor loop with respect to the environment, the robot simulates internally its sensory-motor sequence, then generates a motor program to reach a given goal point. This is the closed-loop mode. Once a motor program is generated, it is executed with the robot once again switched to the open-loop mode.

4.1 Forward modeling

The idea of forward modeling has been used to explain planning and trajectory control for voluntary human arm movements as well as for industrial manipulators [19, 46]. In the motor skill learning for an arm, the transformation of proximal coordinate systems (e.g., joint torques of the arm) to distal coordinate systems (e.g., endpoint coordinates for the arms) is trained on a feed-forward network which serves as a forward dynamical model for the arm. Once this forward model is acquired, the necessary temporal joint torque, for a given specification in distal coordinates such as the arm endpoint, can be computed by the network through the forward model. This is called "computation of inverse dynamics". This framework can be applied to our problem of navigation learning.

By learning the forward model of the workspace, the robot becomes able to conduct lookahead prediction of the sensory input sequence for an arbitrary motor program—i.e. it can simulate the resultant travel from a given navigation plan. Also, the acquired forward model can generate a motor program to reach a goal, specified by its distal sensory image, through computation using inverse dynamics. This sub-section explains how the forward model can be used as a predictor of the sensory input sequence, and how such

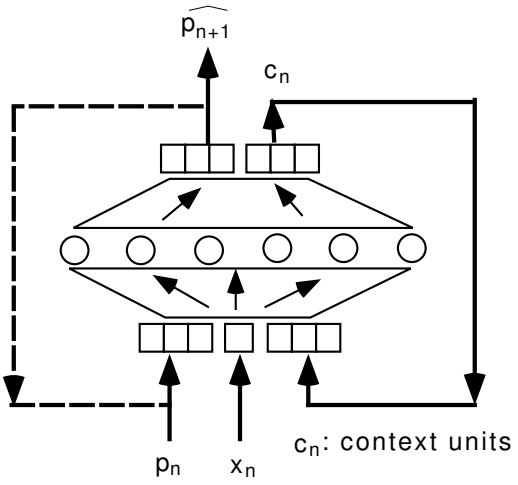

Figure 5: Forward model using the RNN architecture. The dotted line indicates the closed sensory loop which is used for lookahead prediction in multiple steps.

a forward model can be learned by means of a recurrent neural network (RNN). The scheme of generating a motor program using inverse dynamics will be explained in the next sub-section.

The RNN architecture

The objective forward model is embodied using a standard discrete time RNN architecture [11, 18] as shown in Figure 5. This RNN architecture receives the current sensory input p_n, the current motor command x_n, then outputs the prediction of the next sensory input $\hat{p_{n+1}}$. Here, p_n and x_n are a vector and a scalar respectively. A standard sigmoid type function [38] is employed to compute the activation of each neural unit. The sensory input p consists of the compressed range profile p^r obtained from the output of the Kohonen net and the local travel distance p^l. The motor command x_n takes a binary value of 0 (corresponding to staying at the current branch) or 1 (corresponding to transit to a new branch).

We employ the idea of the context loop [11, 18] which enables the network to obtain a certain temporal internal representation. (In Figure 5, there is a feedback loop from the context units in the output layer to those in the input layer.) The current context input c_n (a vector) is a copy of the context output at the previous time step: by this means the context units remember the previous internal state. The navigation problem is an example of a so-called "hidden state problem" [30]: a given sensory input does not always represent a unique situation/position of the robot. Therefore, the current situation/position is identifiable, not by the current sensory input, but by the memory of the sensory-motor sequence stored during travel. Adequate temporal internal representation of the travel history, by taking advantage of the context loop, can achieve just such a memory structure.

Here, the mapping function of the RNN can be written as;

$$\begin{aligned} c_{n+1} &= f_c(p_n, x_n, c_n, W_c) \\ \hat{p_{n+1}} &= f_p(p_n, x_n, c_n, W_p) \end{aligned} \tag{2}$$

where f_c and f_p are the nonlinear maps from the current step to the next step; W_c and W_p denote their parameter sets of connectivity weights. These connectivity weights are determined through the training of the RNN, the methodology of which will be described later.

Using the forward model

As we have described earlier, the forward model represented by this RNN architecture is switched to the open-loop mode before it is used in the closed-loop mode in the navigation phase. In the open-loop mode, the robot conducts the one-step lookahead prediction (it predicts next sensory input as the result of the current motor command) while it travels in the workspace using an arbitrary motor program. The one-step prediction is obtained by inputting the current sensory input and the current motor command to the network. The RNN, at the beginning of travel, cannot predict the next sensory input correctly since the initial context value is set randomly. However, the context value can become *situated* as the RNN continues to receive the sensory-motor sequence. Then the RNN will begin to predict correctly. (In section VI we will explain the underlying mechanism of *situatedness* in detail.) This *situatedness* also accounts for the auto-recovery mechanism of the robot from miscellaneous temporary disturbances during its travel. Although the robot might loose its context by sudden noise or temporary geometrical changes in the workspace, it can recover the context as long as it continues to travel using the sensory input sequence.

After the prediction in the open-loop mode recovers, the RNN can be switched into the closed-loop mode by stopping the robot at a branch point. A lookahead prediction of an arbitrary length for a specified motor program can be made by copying the previous prediction of the sensory input to the current sensory input. (The dotted line in Figure 5 indicates how the closed-loop for sensory input is made.) Let us denote the motor program by $x* : (x_0 x_1 x_2 \cdot \cdot\cdot)$. Then the lookahead prediction of the sensory input sequence $\hat{p*} : (\hat{p_1}\hat{p_2}\hat{p_3} \cdot \cdot\cdot)$ can be obtained by recursively applying $x*$ to the RNN mapping function, using the initial values of the context units c_0 and the sensory inputs p_0 which have been obtained in the open-loop mode.

Learning the forward model

We will now describe briefly how to determine the connectivity weights by using the sensory-motor sequence sampled during the actual wandering travel of the robot. The training of the RNN searches for the optimal W_c and W_p such that the RNN switched to the closed-loop mode can make a correct lookahead prediction $\hat{p*}$ for the sampled sequence $p*$ using the associated motor program $x*$. (This search process should also determine the value of initial context c_0 that produces a correct lookahead prediction for the sampled sequence.) Therefore, the network is trained to minimize the cost function J given by:

$$J = 1/2 \sum_n (p_n - \hat{p_n})^T (p_n - \hat{p_n}) \tag{3}$$

The optimal connectivity weights and initial context minimizing the cost J in (3) are computed using the back-propagation through time (BPTT) algorithm [38]. The RNN is transformed into a cascade network without loops. The forward computation of this network with the temporal connectivity weights and the temporal initial context value c_0 generates the temporal lookahead prediction $\hat{p*}$ which corresponds to the motor program

$x*$. Then, the error between the sampled sequence $p*$ (as a teacher) and the temporal lookahead prediction $\hat{p}*$ is calculated, which is back-propagated [38] in order to update the temporal values of W_c, W_p and c_0. This computation is repeated until the error (the cost function J) is minimized. This learning achieves locally optimal mapping functions of f_c and f_p in (2) by organizing an adequate temporal internal representation in the context loop. In the actual training, the sequence of the sampled data is broken into smaller sub-sequences (15 data units for each sub-sequence in the experiment described later), each of which is used to train the network simultaneously. This technique is used since the error due to temporal lookahead prediction over numerous steps can accumulate to a substantially large value in the middle of training which hampers the smooth convergence of the learning process.

Numerous studies have been conducted of the problem of learning the sequential behavior of agents [6, 30, 47] including our prior work on *skill-based learning* [43, 42]. These studies have shown that certain temporal internal representation are indispensable to the solution. *Model-based learning*, presented here, differentiates itself in that its learning comprises not just learning sequences but also extracting grammatical structure hidden in the sequences. Elman [11] investigated experimentally the capability of RNNs for learning a simple grammar from certain letter sequences, and examined the internal representation obtained as a function of time. His study showed that the RNN, after successful learning, becomes capable of following letter sequences since the activation of the context units represents the hidden state of the target automaton. Learning the forward model of the environment is analogous to the learning of a grammar by a finite state machine (FSM). The robot attempts to extract grammatical regularities hidden in the branching structure of the environment from the sensory-motor sequences sampled so that they can be used to generate the lookahead prediction of the sensory sequence for an arbitrary motor program. This objective is achieved when an adequate memory structure is successfully self-organized in the RNN.

4.2 Plan generation

The objective of planning is to find a motor program $x*$ that generates a path to the desired branch or terminal points under the condition of minimum travel distance. We investigate how an optimal motor program can be derived from the obtained forward model in an autonomous manner. We consider the following cost function for the motor program.

$$\begin{aligned} E &= \epsilon E_g + \gamma E_c + \mu E_m \\ E_g &= 1/2(p_d - \hat{p_\tau})^T(p_d - \hat{p_\tau}) \\ E_c &= 1/2 \sum_{n=1}^{\tau} (\hat{p_n^l})^2 \\ E_m &= -\sum_{n=0}^{\tau-1} \int_0^{x_n} [\phi((x-0.5)/T) - x]dx \end{aligned} \tag{4}$$

The total cost E is defined by the summation of three different cost items, E_g, E_c and E_m, with their respective weights, ϵ, γ, and μ. E_g denotes the norm between the lookahead prediction of the sensory input at the τth step $\hat{p_\tau}$ and the desired sensory input p_d. Here, τ represents the number of branching steps from the current to the goal point. This

cost item indicates how close the current motor program's prediction of the distal sensory input is to that of the goal. E_c denotes the cost incurred for the minimum travel distance, which is the mean-square sum of the local travel distance $\hat{p_n^l}$ over τ steps. The term E_m is employed in order to restrict the value of each motor command to a binary value of 0.0 or 1.0. (Note that only binary values are legal for the motor commands.) ϕ is a standard sigmoid function, and T is a parameter defining its steepness.

Now, the optimal motor program, which minimizes the cost function, is computed iteratively. A difficult point is that the number of steps of the motor program τ is also a variable to be determined, since we cannot tell *apriori* how many branching steps ahead the goal point is. In our formulation τ is determined through the iterations. We have defined the maximum number of future branching points to be considered in the planning by τ_{max} (we took $\tau_{max} = 15$ steps in the experiment described later). As a result, the RNN is transformed into a cascaded feed-forward network consisting of τ_{max} steps. The forward computation is conducted on the cascaded network, in which the lookahead prediction of τ_{max} steps for the temporal motor program $(x_0 x_1 x_2 \cdots x_{\tau_{max}})$ is obtained. On determining the temporal value of τ, the cost E is computed with changing τ from 1 to τ_{max} based on (4). The τ which shows the minimum cost is taken as the temporal value of τ. This τ represents the valid length of the temporal motor program. Next, an update of the motor command at each step is obtained. The gradient of the cost function with respect to each motor command $x_n\,(0 \leq n \leq \tau - 1)$ is calculated; this indicates the direction of update for the motor commands. $\delta x_n\,(0 \leq n \leq \tau - 1)$ is given by:

$$\begin{aligned} \delta x_n &= -\frac{\delta E}{\delta x_n} \\ &= -\epsilon \frac{\delta E_g}{\delta x_n} - \gamma \frac{\delta E_c}{\delta x_n} + \mu(\phi((x_n - 0.5)/T) - x_n) \end{aligned} \tag{5}$$

In the second line of this equation, the gradient of the cost function is represented as the sum of the gradient of each cost item. In obtaining $\frac{\delta E_g}{\delta x_n}$, the error between the desired sensory input and the lookahead prediction of the sensory input at the τth step is calculated, then this error is back-propagated [38] through the cascaded network to the motor command unit x_n so that the contribution of x_n to the error can be estimated. This estimate yields the objective gradient value. The value of $\frac{\delta E_c}{\delta x_n}$ is also calculated by using the back-propagation scheme. The prediction of the local travel distance at each step is obtained as an output from the cascaded forward network. Then, back-propagation is applied from the output unit to each motor command unit so that the contribution of each motor command's value can be obtained in order to minimize the local travel distance. The third term is the gradient of E_m, which is obtained analytically from (4). Since the sequence of motor commands after the τth step does not contribute to the cost, $\delta x_n\,(\tau \leq n \leq \tau_{max})$ is set as 0. The exact update for each motor command in the temporal motor program $\Delta x_n\,(0 \leq n \leq \tau_{max})$ can be obtained by applying the steepest descent method to (5) using:

$$\Delta x_n(t+1) = \eta \delta x_n + \alpha \Delta x_n(t) \tag{6}$$

where η is the search rate and α is the momentum term. The details for the method of back-propagation through the forward model are given in ref. [19]. One cycle of forward and backward computation is completed after updating the temporal motor program. The temporal motor program as well as its valid length τ change gradually through

the iteration of this cycle, thereby minimizing the cost. When the cost is minimized, the sequence of motor commands obtained as $x_n\,(0 \leq n \leq \tau - 1)$ is the desired motor program.

4.3 Chaotic search

One might think that the optimal motor program could be obtained easily through iterative calculations by the steepest descent method described in the prior subsection. However, this is not true. Many researchers [23, 24] have studied robot path planning for the minimum travel in complex obstacle domains, and they have shown that such planning cannot avoid a combinatorial explosion. This indicates that the landscape of the defined cost function would be quite "rugged" in our formulation, and that the computation of x_n by the method of steepest descent, for which the search dynamics are those of a typical fixed point because of its positive damping term, can easily be trapped by a local minimum. Such planning processes would halt after generating a sub-optimal plan. (The experiments described later will show this explicitly.) In order to realize an autonomous search process which generates various alternatives, it is necessary to introduce nonequilibrium dynamics which are capable of avoiding the local minimum problem.

In our previous work, we have studied the characteristics of a dynamical system called the "chaotic steepest descent" model (CSD) that has a nonlinear resistance which varies periodically [41]. We review this model briefly. Let us consider a dynamical system defined on a rugged energy landscape by:

$$\begin{aligned} m\ddot{x} + R(\dot{x}, \omega t) &= -\kappa \nabla E(x) \\ R(\dot{x}, \omega t) &= [d_0 sin(\omega t) + d_1]\dot{x} + d_2 \dot{x}^2 sgn(\dot{x}) \end{aligned} \tag{7}$$

where m is an inertia constant, R is the nonlinear resistance function, $\nabla E(x)$ is the gradient of the energy function, κ is the gradient constant, ω is the periodicity of the resistance perturbation and d_0, d_1 and d_2 are the nonlinear resistance coefficients. The resistance may have a positive or negative damping, depending on the $\dot{x}$ value. As the resistance characteristics change, by slowly increasing the negative damping part, the resulting state tends to travel from one energy basin to another. On the other hand, when the positive damping is increased, the state tends to converge. Repetition of these unstable and stable phases generates chaotic state transitions among basins. We employ the CSD model to update x_n:

$$m\ddot{x}_n + R(\dot{x}_n, \omega t) = -\epsilon \frac{\delta E_g}{\delta x_n} - \gamma \frac{\delta E_c}{\delta x_n} + \mu(\phi((x_n - 0.5)/T) - x_n) \tag{8}$$

The dynamics run for a pre-determined period, during which various motor programs are generated in a stable phase, and the motor program with the minimum cost is selected as an optimal solution.

The question may arise as to why the planning search method uses chaotic dynamics rather than other alternatives. An easy alternative might be an exhaustive random search in the binary space of the motor programs. Although the random search does work perfectly for an application of the current experimental size, it would not be scalable to more complex tasks. Another alternative is to apply stochastic dynamics to the search process. External additive noise would prevent a search based on the steepest descent method from

entering the local minimum traps. Although there is no present mathematical proof to confirm that the efficiency of a chaotic search is better than that of a stochastic search in combinatorial search problems, numerous studies, especially in biological systems, have suggested its plausibility. Sakarada and Freeman [40], Aihara [1], Tsuda, Koerner and Shimizu [45] have suggested that for effective memory searches the biological brain takes advantage of internal noise, induced by the deterministic chaos which emerges in natural neural circuits; Nara and Davis [34] stressed that control of the parameter set can "harness" a chaotic search into an effective subspace far smaller than the problem domain. They observed that this harnessing of chaos exhibits much more diverse behavior than that of stochastic systems involving temperature control. The introduction of chaos here is based on the hypothesis that cognitive tasks of planning in biological brains use the forward model described by deterministic dynamics. Recent research [39] has found a biological example of computing the inverse dynamics of eye movement in the cerebellum. This suggests that biological brains actually compute certain simple motor plans based on deterministic dynamics using internal models. We believe that planning in the cognitive level utilizes deterministic chaos to solve problems because it must deal with combinatorial computation.

5 EXPERIMENTS

We conducted experiments on the scheme presented above using the mobile robot *YAMABICO*. In the learning phase, the robot repeats cycles of the learning trial with increasing number of samples in the training data until statistical tests of lookahead prediction satisfy certain criteria. Then experiments of plan-based navigation are conducted.

5.1 Learning and lookahead prediction

The adopted RNN architecture is three-layered, and has 10, 12 and 9 units for the input, hidden, and output layers respectively including four context units in the input and output layers. During each learning trial the robot wanders around an adopted workspace for a certain period, making each branching decision at random, in order to collect an additional amount of data (sensory-motor sequences). Thereafter, the data set which has been accumulated so far is used for training the RNN. For each trial, the connectivity weights of the network are set randomly and trained off-line using the data. The training of the RNN is conducted for 20,000 iterations, which are repeated if the mean square learning error per output unit cannot be decreased below 0.01. After the learning error is minimized, the test of a given lookahead prediction is conducted for 10 different travels. Each travel starts from an arbitrary position in the workspace using random branching. The robot travels with the RNN switched in the open-loop mode until the RNN becomes able to predict the next sensory input (i.e. it becomes *situated*). The robot is stopped when the prediction error for all sensory input units becomes less than 0.15 twice in succession. At this moment, we assume that the robot is *situated*. Then, lookahead prediction is conducted, with the RNN switched in the closed-loop mode, for an arbitrary motor program which comprises seven branching steps. Thereafter, the robot is directed by the motor program in order to compare the actual sensory input with the lookahead prediction. After 10 travels, the mean square prediction error per sensory input unit (MSPE) is calculated. If

Table 1: Summary of three trials of learning, which show the number of samples in the training data set, the iterations required for the training of the RNN, the average branching steps necessary to become *situated*, and the mean square prediction error per sensory input unit (MSPE) during navigation over 10 travels for each trial.

trial	number of samples used for training	learning iterations	avg. steps	MSPE
1st.	49	20,000	19.4	0.131
2nd.	102	20,000	9.6	0.072
3rd.	193	40,000	5.2	0.009

the test of lookahead prediction is not satisfactory, we let the robot travel again in order to sample the data furthermore, which is used to re-train the network.

The experimental results of the learning are summarized in Table 1. In the first trial, the robot sampled 49 steps of the sensory-motor sequence, then the training process of the RNN with the sampled sequence converged after the first 20,000 iterations. This computation took about 20 minutes using a Sony News workstation with R4000 CPU (100MHz). In the travel after this training, many steps were required (the average steps in ten travels were 19.4) until the RNN in the open-loop mode supplied good predictions (i.e., became situated). In the ensuing lookahead predictions in the closed-loop mode, the RNN could usually not predict more than three steps ahead. It seemed that the RNN learned only particular instances of the sampled sequences but not in a more general way. The MSPE calculated was 0.131. In the second trial, the robot sampled further sensory-motor pairs, by which the number of samples in the training data set was raised to 102. The training process of the RNN with the data set converged after 20,000 iterations (it took about 45 minutes). After the training, the necessary steps to become *situated* were shortened (in the average 9.6 steps), and the lookahead prediction often was good for several steps. However, once the prediction failed in the middle of a sequence, it continued to fail for subsequent steps. The MSPE was reduced to 0.072. In the third trial, the RNN was trained with 193 sensory-motor pairs after 89 pairs were sampled. The training could not converge within the first 20,000 iterations, but it converged after another 20,000 iterations (it took about 170 minutes as total). After this learning trial, it was observed that the robot became *situated* within a few steps (the average steps were 5.2), and also that lookahead predictions became accurate except in cases affected by noise. The MSPE was reduced to 0.009. Since the RNN could correctly predict sequences it had never exactly learned, it can be said that the RNN succeeded in extracting the necessary rules in the form of generalized ones. Figure 5.1 shows the distribution of the prediction error for all sensory input units in the third trial. It is shown that the fraction of "good" predictions with an error of less than 0.1 is more than 70 percent. This result implies that the robot successfully learned the forward model of the workspace.

An example of the comparison between a lookahead prediction and its sensory sequence during travel is shown in Figure 5.1. In (a) the arrow denotes the branching point where the robot conducted a lookahead prediction using a motor program given by 1100111. The robot, after conducting the predictions, traveled following the motor program, generating an "eight-figure" trajectory, as shown. In (b) the left-hand side shows the sensory input sequence, while the right-hand side shows the lookahead sequence, the motor program

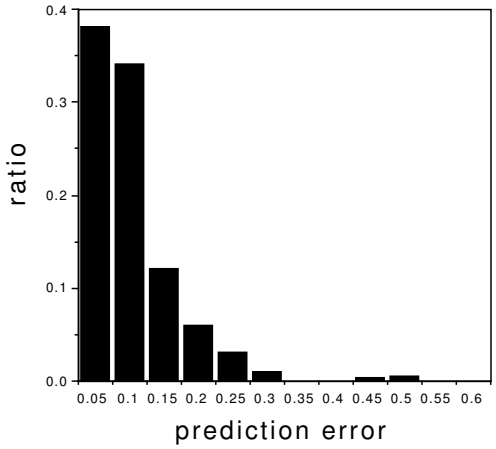

Figure 6: Distribution of the prediction error for all sensory input units in the final trial of the learning phase.

and the context sequence. The values are indicated by the bar heights. This sequence consists of eight branching steps (from the 0th to the 7th step) including the initial one in the "start" point. It can be seen that the lookahead for the sensory input agrees very well with the actual values. It is also observed that the context as well as the prediction of sensory input at the 0th and the 7th steps are almost same. This indicates that the robot predicted its return to the initial position at the 7th step in its "mental" simulation. The robot actually returned back to the "start" point at the 7th step in its test travel.

We have stated that the *situatedness* accounts for the mechanism of the auto-recovery from temporary perturbation. The next experiment demonstrates such an example. The robot traveled in the workspace while predicting the next sensory input with the RNN switched to the open-loop mode. During this travel, an additional obstacle was introduced. Figure 8 (a) shows the trajectory of the robot's travel; Figure 8 (b) shows the comparison of the actual sensory input and the corresponding one-step lookahead prediction. The branching sequence number is indexed beside the trajectory; these numbers correspond to the prediction sequence in Figure 8 (b). The prediction starts to be incorrect once the robot passes the second branching point, as it encounters the unexpected obstacle. The robot, however, continued to travel and in the meanwhile we removed the obstacle from the workspace (when the robot passes the fourth branch). After the sixth branching point, the prediction returns to the correct value. This indicates that the lost context is recovered while the RNN receives the regular sensory input sequence. It is noted that the values of the context units in this branch are almost the same as those of the first branch. This shows that the robot recognized its return to the same branching point because it became *situated* in the behavioral context again.

5.2 Planning

In this section, we demonstrate that our scheme provides a mechanism for the autonomous generation of motor programs. Consider the following experiment. In Figure 5.2, the robot was stopped at the branch A after it became *situated*. Then the robot performed its planning of the route to the given destination, B. B is one of the dead-end positions, and

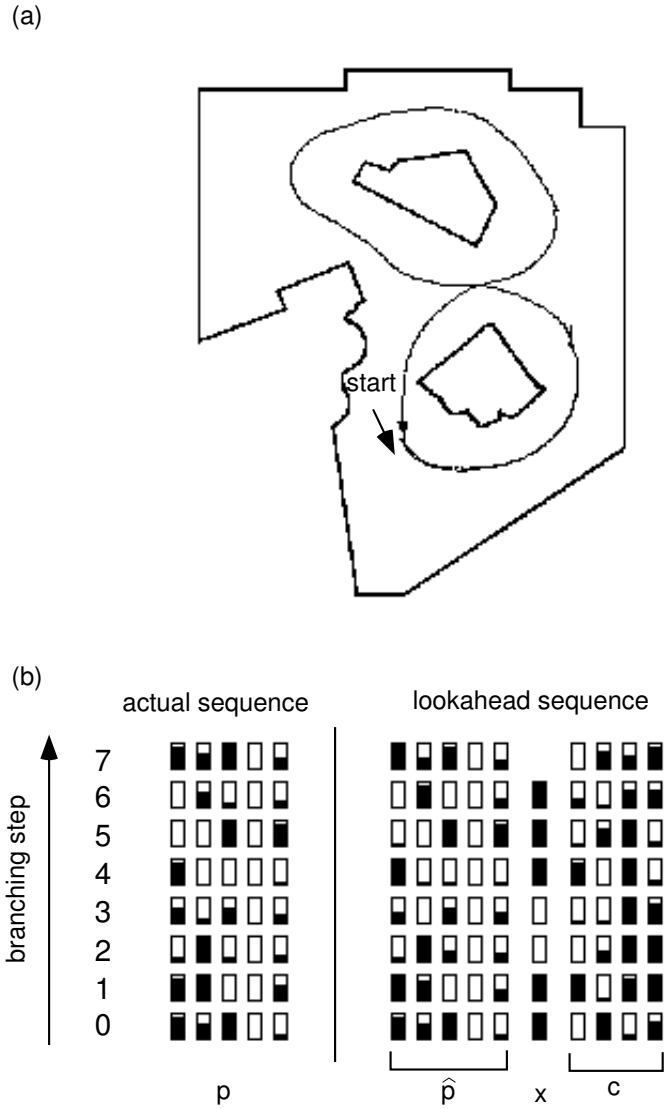

Figure 7: (a) The robot conducted lookahead prediction for a motor program given by 1100111 at the branching point indicated by the arrow, after which it traveled according to the motor program, generating the trajectory shown. In (b) the left side shows the actual sensory sequence, and the right side shows the lookahead prediction, the motor program and the context sequence. The sensory and the context sequences are shown for eight steps, including their initial values, p_0 and c_0, at the bottom. The motor program is shown for seven steps ($x_0 \rightarrow x_6$).

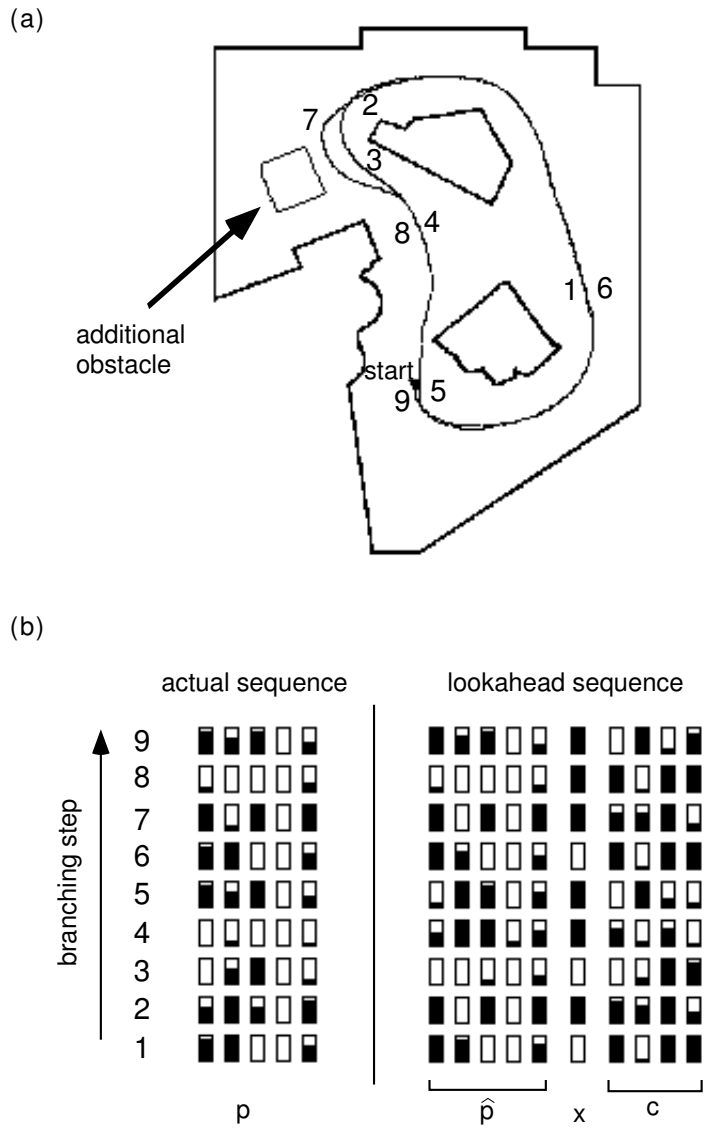

Figure 8: Auto-recovery from the addition of an obstacle. In (a) the trajectory of the robot's travel is shown. The additional obstacle is indicated by an arrow. The numbers indicate the branching sequence number. (b) shows the sensory sequence on the left and the one-step lookahead, the motor program and the context sequence on the right.

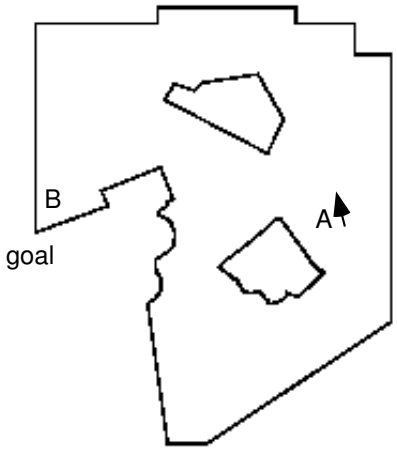

Figure 9: The robot planning for specified goal B from the current location A

its sensory input has already been given to the robot. The robot conducted its planning by following the dynamics described by (8) with the following parameters: $\epsilon = 1.0$; $\gamma = 0.05$; $\mu = 20.0$; $T = 0.05$; $m = 1.0$; $d_0 = 0.1$; $d_1 = -0.11$; $d_2 = 3.8$; $\omega = 2\pi/400$; and $\kappa = 0.001$.

Figure 5.2 shows the resulting time evolution for a planning process involving 2,500 iterations. The temporal motor program (for every 10 iterative steps) is shown in the lower part, and the cost of each plan is plotted in the upper part. A temporal motor program is indicated by a column consisting of black and white squares, where a white square denotes a persistence in the current branch (0) and a black square denotes transit to a new one (1). Symbol size indicates actual activation value of x_n. In this figure, we only show the valid length (τ steps) of motor commands in the temporal motor program. Note that the valid length of the temporal motor program changes through the iterations.

From Figure 5.2, it can be seen that multiple motor programs with relatively low cost are generated at stable phases through successive state transitions. These are 101, 01010, 110 and 00 as indicated by arrows in Figure 5.2.

We tested these motor programs by letting the robot activate them. Figure 11 (a)-(d) shows the resultant travel for each of programs. While program (b) proved to be redundant, generating a fruitless loop, and program (c) pursued the wrong goal, the other programs produced acceptable results. Note that the good programs produced slightly lower costs. We examined the (c) case and determined that a false goal was reached because the sensory pattern resembled that of the desired goal.

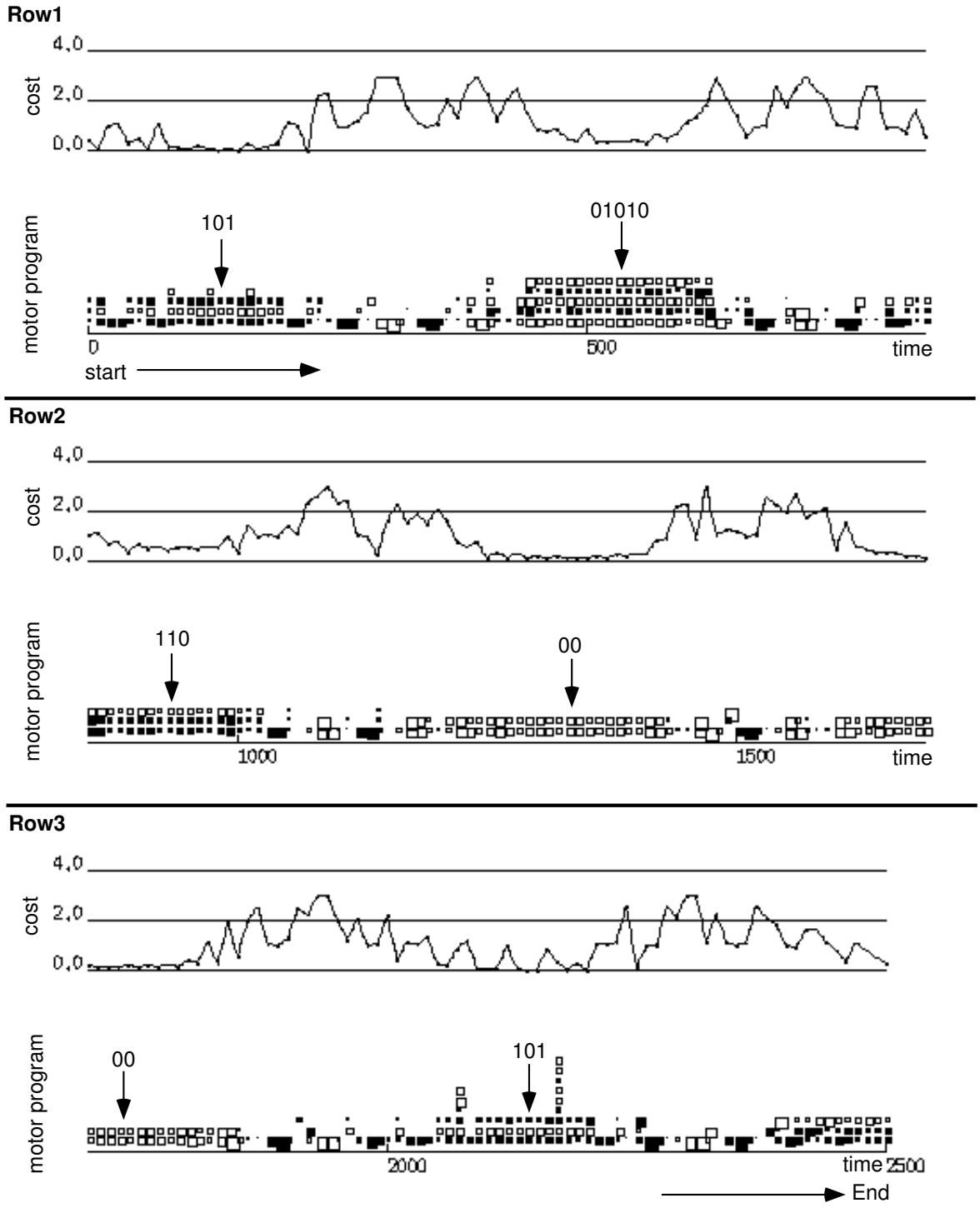

Figure 10: The chaotic search process of the motor program is shown in the three rows. In each row, the upper part indicates the cost time history, and the lower part the temporal motor program. The temporal motor program is indicated by a column consisting of black and white squares, representing motor commands of 1 and 0 respectively. The motor programs obtained in the stable phase (101, 01010, 110, 00) are indicated by arrows. Time flows from row 1 to row 3.

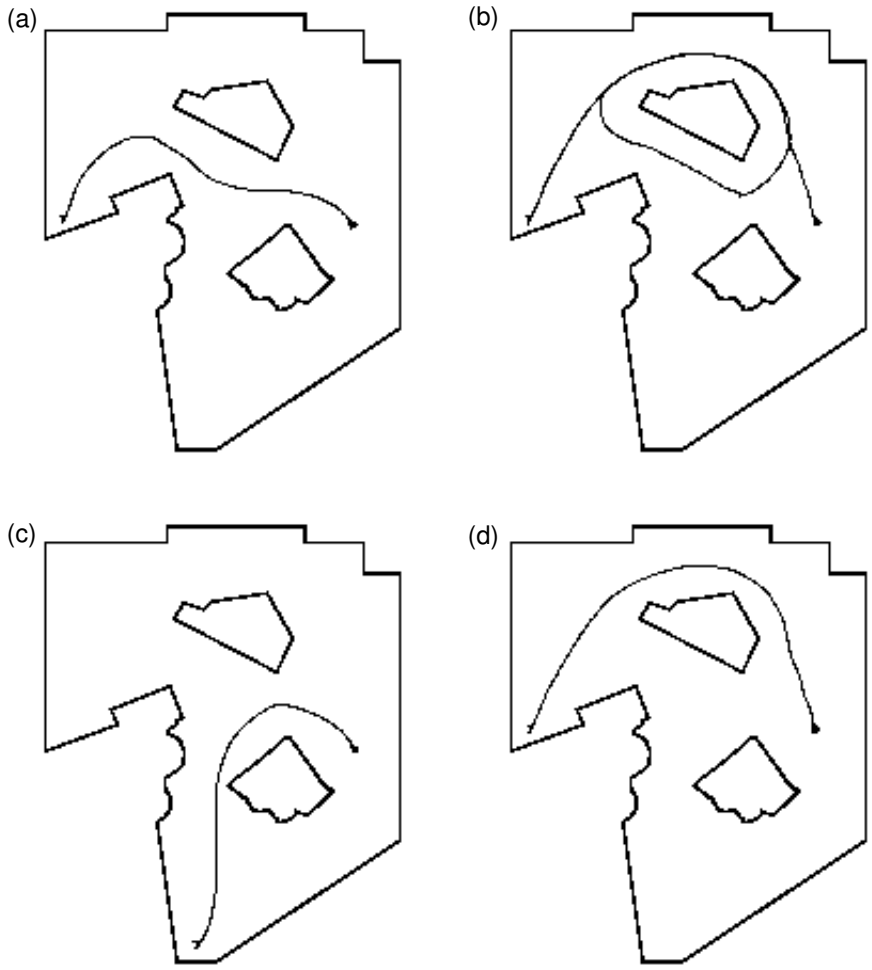

Figure 11: Travels based on various motor programs. (a) and (d) are almost optimal trajectories, (b) is redundant, and (c) pursues the wrong goal. (a), (b), (c) and (d) correspond to the motor programs 101, 01010, 110 and 00, respectively.

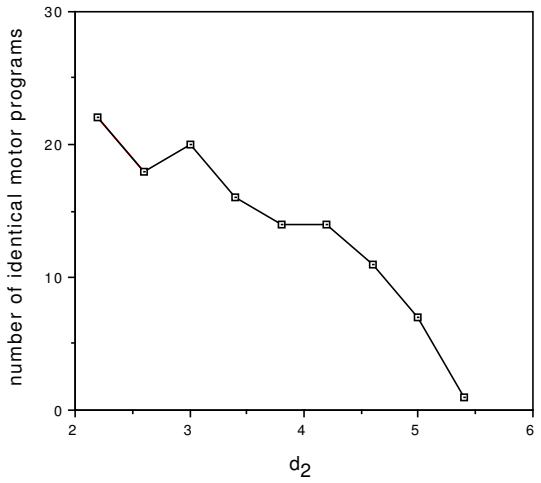

Figure 12: The number of identical motor programs generated during the search as a function of d_2.

5.3 Parameter sensitivity of chaotic searching

It is interesting to observe how the parameter settings affect the chaotic search. The parameter d_2 represents a coefficient of positive damping in the CSD dynamics, therefore it is assumed that its value will affect the dynamical characteristics substantially. In the following, we focus on this parameter, and investigate how the characteristics of the chaotic search vary depending on its value. We conducted this experiment using the same planning task described above.

The parameter d_2 was varied between 2.2 and 5.4 with intervals of 0.4, for which the search process was computed for 40,000 steps. The motor program was sampled at the most stable phase of each cycle i.e. when $\omega t = \pi/2$ in (8), which results in 100 samples of the motor program for each search process. First, we examined how the diversity of the generated programs varies with d_2. Figure 5.3 plots the number of identical motor programs generated for each parameter value. It is shown that the number of identical motor programs decreases as the value of d_2 increases. When d_2 was set to 5.4, only one motor program was generated — no state transitions took place. Furthermore, we investigated the cost distribution and the frequency of the state transition in order to examine the detailed structure of the search process. In Figure 5.3, the left-hand side shows the cost frequency of the programs generated, and the right-hand side shows the frequency of repeated occurrences of the same program, for d_2 values of 2.2, 3.8, and 5.0. The figure shows that, for small d_2 values, the cost tends to be spread over a wide range, and the motor program generated is different on almost every cycle. The search proceeds almost at random in the wide range of the problem's space. On the other hand, for larger d_2 values, the cost distribution becomes approximately optimal, and the probability of repeating the same motor program increases. The search tends to proceed more precisely towards optimal and sub-optimal solutions. However for these cases the search becomes more likely to be trapped in one of the sub-optimal solutions for long periods (the state transition takes place only intermittently.) The risk of local minimum traps becomes more pronounced as d_2 is set to larger values.

An important question, therefore, is how to determine the optimal value of d_2. We

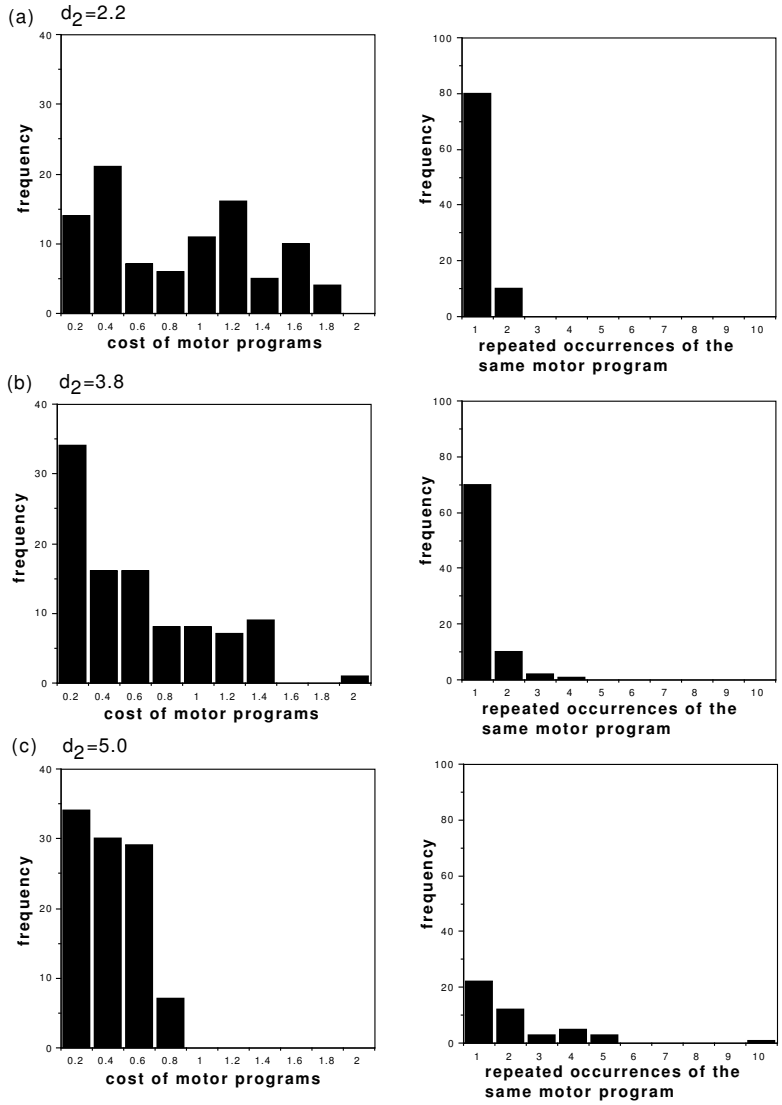

Figure 13: Frequency of the cost of generated motor programs (left chart), and the frequency of repeated occurrences of the same motor program (right chart), for d_2 values of (a) 2.2, (b) 3.8, and (c) 5.0.

believe that it is determined by the trade-off between the time required for planning and the cost of the motor program thus obtained. If the optimal cost plan is that regardless of the period of time required, d_2 might be set to a small value. The resultant random search would find the minimum cost plan in the long run, without being captured by the local minimum traps. On the other hand, if the time spent is crucial, d_2 should be set to a larger value. The resultant search would find a sub-optimal plan quickly, but the search might not be able to reach the global cost minimum in such a limited time. We believe that the optimal parameter will be determined by considering the real-time requirements of the planning process. Although the current paper will not include further discussion of real-time planning issues, the experimental results convince us that the idea of "harnessing chaos" is potentially applicable in this context.

6 THE DYNAMICAL MECHANISM OF *SITUATEDNESS*

Our experimental results have shown that the robot can become *situated* from an arbitrary initial state through interaction with the environment. This section explores the mechanism underlying this *situatedness* by investigating the essential dynamical structure that arises from the coupling of the internal neural system and the environmental system.

First, we define the term "attractor" for both the environmental and the neural dynamics. Let us focus on the environmental dynamics F which define how the robot actually travels in the workspace with respect to the motor command. Suppose the robot travels in the workspace for an infinite time period, receiving a motor program $x*$ which is generated randomly. Let $s* : (s_0 s_1 \cdots s_\infty)$ and $p* : (p_0 p_1 \cdots p_\infty)$ be the sequences of branch positions and the sensory input, respectively, during the resultant travel of the robot. The branch position s_n represents the state of the environmental dynamics F at the time n. Since $s*$ would be limited to a subspace of the entire workspace after an initial transient period, an invariant set $\underline{s}*$ is formed in $s*$. (Hereafter, $\underline{X}*$ and $\underline{X_n}$ represent the invariant set and one of its elements, respectively, in an infinite sequence $X*$.) We define this invariant set $\underline{s}*$ as the attractor of F. It is important to note that this attractor is the global attractor, since the robot's trajectory in the workspace converges to the same attractor regardless of its starting position. Also, we define an invariant set $\underline{p}*$ for the sequence of sensory input which $\underline{s}*$ corresponds to.

For the neural dynamics f, let us consider a lookahead prediction of the RNN (in the closed-loop mode) with respect to a motor program $x*$ of an infinite length which is generated randomly. This generates an infinite sequence of the transitions of the context $c*$. When this infinite sequence forms an invariant set, this invariant set $\underline{c}*$ is defined to be the attractor of f. The prediction of sensory sequence which corresponds to $\underline{c}*$ is indicated as $\underline{\hat{p}}*$. We note that the generation of the global attractor might not be assured for f, depending on the learning process. A trajectory with a different c_0 might reach a different attractor. Since the objective of learning is to make the neural dynamics f emulate the environmental dynamics F by means of the sequence of sensory input, f in the limit of a learning process satisfies:

$$\exists \underline{c}_0, \exists \underline{s}_0 \Rightarrow \underline{\hat{p}}* = \underline{p}* \tag{9}$$

for an arbitrary motor program $x*$. The notion here is that there is at least one attractor

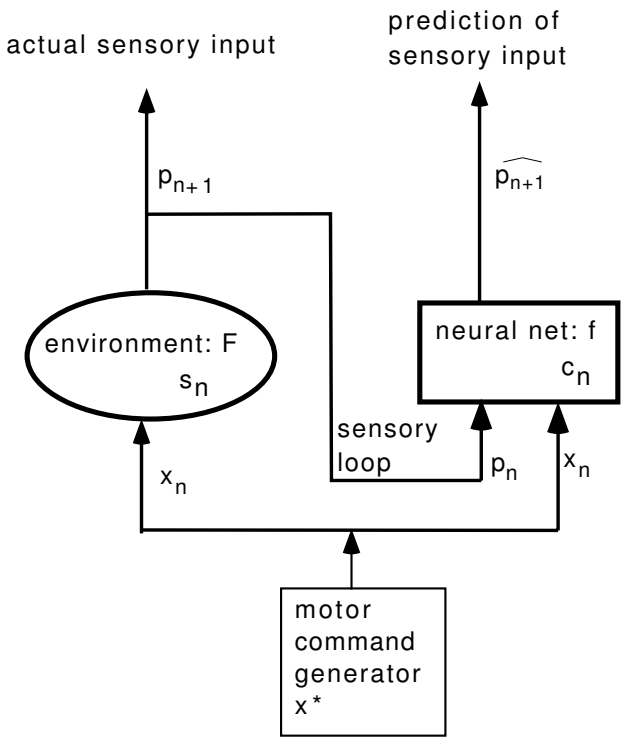

Figure 14: The internal dynamics are made coherent by the environmental dynamics through entrainment using sensory coupling.

for f for which the lookahead prediction of the sensory input can be made correctly, which will satisfy (9).

Now let us consider the coupling between these two dynamics as shown schematically in Figure 6. In the open-loop mode, the RNN predicts the next sensory input as $p_{n+1}^{\hat{}}$ using the current sensory input p_n while the robot travels following the motor program $x*$. In the figure, the sensory return loop exiting from the environmental dynamics and in input to the neural dynamics is shown. Suppose that both the environmental dynamics F and the neural dynamics f start to run from their transient states (s_0, c_0). Initially, the prediction of the sensory input does not match the measured one. In the meanwhile, the environmental dynamics F converge onto their global attractor, thereby producing the regular sequence of sensory input $\underline{p}*$. Upon receiving this regular sequence of sensory input $\underline{p_n}$, the neural dynamics f begin to output the next prediction $p_{n+1}^{\hat{}}$ correctly, as being equal to $\underline{p_{n+1}}$ while the context values c_n converge onto $\underline{c}*$. This convergence is assured generally for $\forall c_0$ by (9), provided the neural dynamics f are trained to have the correct global attractor.

The above analysis has shown that the creation of the global attractor in the neural dynamics is essential to achieving the *situatedness* and the auto-recovery mechanism of the robot. Here, we examine whether or not the RNN, which was trained in our previous experiment, has generated the global attractor. The RNN is switched to the closed-loop mode, then forward computation is conducted with a randomly generated motor program for two thousand forward steps. The resultant orbit of the context $c*$ is plotted in the two-dimensional space (c^1, c^2) by using the activation states of two context units (these two

units are arbitrarily chosen), and excluding the first 100 points which resulted from the initial transient steps. Fig. 6(a) shows the orbit obtained, while (b) shows an enlargement of part of (a) where a highly one-dimensional structure is observed. We repeated this several times with different initial values for the context units, and found that they all resulted in the same invariant set. (The same qualitative results were obtained for any pair of the context units.) This confirmed that the neural dynamics, which have been used in the experiment, are characterized by a global attractor. Although no theory has been established to explain the creation of a low-dimensional global attractor in recurrent neural learning, this tendency has been observed in other numerical experiments on the learning of simple grammatical structures [36, 44].

The mechanism underlying *situatedness* will now be discussed qualitatively by introducing the physical term "entrainment" [9, 12]. Entrainment is a dynamical phenomenon that coupled nonlinear oscillators become synchronized stably. Recently, Beer [3] studied the self-organization of locomotion controllers in the context of walking motions of insects, where he observed the entrainment of the intrinsic oscillation of the leg controller by the environmental dynamics. Similarly, in our case, the internal neural dynamics become coherent with the environmental dynamics through the sensory loop, where we observe the entrainment of the internal dynamics by the environmental dynamics. In the initial transient state, the neural system and the environmental system are "incoherent", therefore the neural system cannot recognize the present situation/position. In the meanwhile, the two systems start to become coherent by means of entrainment, with the result that the dynamical state of the entire system is confined to the attractor which has reduced dimensionality. At this point, it can be said that the internal neural system has been *situated* in the environment. This dynamical mechanism which generates the *situatedness* is an inherent one as long as the essential dynamical structure of the coupled system is characterized by the global attractor dynamics.

7 DISCUSSION AND CONCLUSION

A primitive conceptualization of the symbol grounding process is conjectured as the result of our experiments. Figure 7 illustrates the concept. As the robot travels around the workspace, clusters of sensory input are collected in the sensory space arising from its branching sequences. Meanwhile the dynamical mapping is self-organized in the internal state space such that it accounts for the transitions among the clusters of the collected sensory inputs. If different symbols are assigned to each cluster of sensory input, the *mental simulation* process carried out by the internal chaotic dynamics might be equivalent to the symbolic process of manipulating a set of symbols: terminal symbols (the sensory input) and nonterminal symbols (the internal state). Here, our primitive symbols are not in the arbitrary shape of usual symbol tokens[1], but in the nonarbitrary shape arising from the physical interaction between the robot and the environment.

One might consider that such symbolic processes could be represented more easily in the form of a FSM. We, however, consider that the internal representation of a FSM is still "parasitic," since symbols are manipulated into an arbitrary shape regardless of their

[1]The discussion inherits Harnad's [15] claim: "Symbol manipulation would be governed not just by the arbitrary shapes of the symbol tokens, but by the nonarbitrary shapes of the icons and category invariants in which they are grounded."

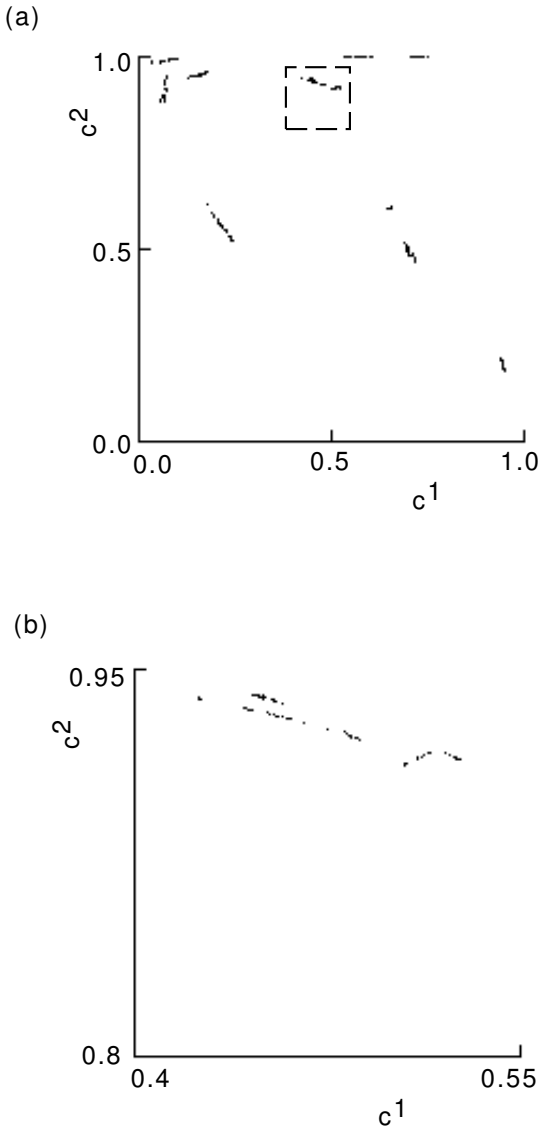

Figure 15: (a) shows the orbit $c*$ projected in (c^1, c^2) space using the activation states of two context units, (b) is an enlargement of the rectangular section in (a), in which a highly one-dimensional structure is seen.

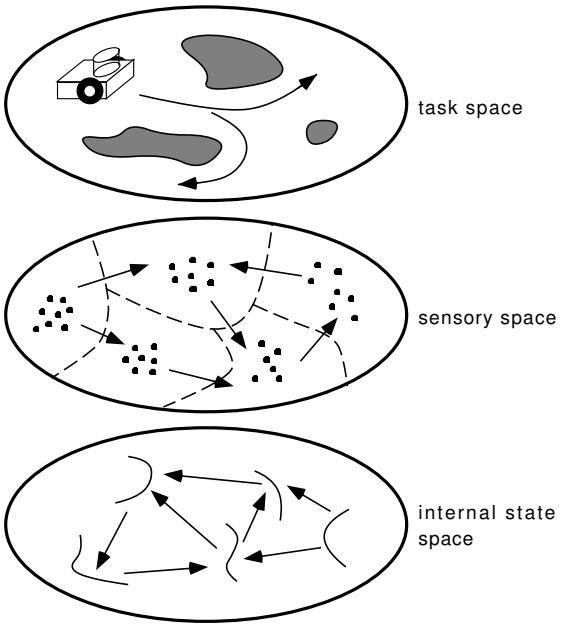

Figure 16: The symbol grounding process.

meaning in the physical world. A crucial gap exists between the actual physical systems defined in the metric space and their representation in the non-metric space, which makes the discussion of the structural stability of the whole system difficult. In contrast to this state of affairs, the representation in our scheme can be said to be intrinsic to the system since it is embedded in the attractor dynamics which share the same metric space with the physical environment. Here, structural stability arises in the interaction between the internal and environmental systems, which accounts for the *situatedness* of the internal process. Although the symbol grounding process, described here, is still primitive, we believe strongly that this philosophy is indispensable to design intelligent autonomous robots operating in the physical world.

It is important that we address the issue of the scalability of our scheme. Although the learning of the forward model, in the adopted simple workspace, successfully generated the global attractor after some trial and error, this sort of global convergence would inevitably become more difficult in more complex environments. Facing this problem, one approach to take is to refine the learning algorithms of the RNNs since the scalability largely depends on the learning capability of the RNNs. Recent research into the RNNs' learning processes have shown progress. Giles [14] reported that increasing the "order" of the connectivity of an RNN enhances remarkably its learning capability. (The "order" refers to the dimensionality of product terms in the weighted sum, which reflects the connectivity of the network.) Bengio *et al.* [4] showed theoretically that learning the long-term dependencies with a standard gradient-descent method applied to back-propagation is difficult. They showed the advantages of other non-gradient methods, such as pseudo-Newton, time-weighted pseudo-Newton, multi-grid random search, and simulated annealing. The idea of expert nets proposed by Jordan and Jacobs [20] is also attractive. The essential idea is to divide a complex learning problem into simpler problems by introducing a "sub-net" architecture. It will be challenging to study whether the same learning principle can be applied to the learning of large size FSMs by RNNs.

Although numerous other research projects concerning RNNs are in progress, the learning capability of RNNs is still an open problem. It is expected that further advances in the theory of RNNs will lead to the discovery of better learning algorithms.

Another possible research direction consists of the investigation of strategies that could cope with insufficient learning without fatal degradation of the system performance in more complex environments. The insufficient learning can arise in three ways. Firstly, the robot might not be able to obtain all possible input data (sensory-motor sequences), which is necessary for building a correct model of the environment, through its behavior. Secondly, even if the robot could obtain all of the possible input data, it might not be able to "digest" all of it to form a correct model in a limited learning time. Thirdly, the environment may change after the robot has learned its model. In these situations, our basic assumption of embedding a correct model in the form of the global attractor is inevitably constrained, and thereby the mechanisms of *situatedness* as well as of optimal planning might not be assured. However it is expected that the robot could recover its context temporarily if it happened to travel around well-known parts of the workspace, and then it might be able to generate certain sub-optimal plans from there.

The current paper has formulated a *model-based* approach based on the assumption of sufficient learning of the model. It is, however, expected that the theory will have to be extended further to the problem of insufficient learning which is likely in open environments associated with more complexity. This field of study is quite attractive to us since an animal or "animat" often lives under such conditions.

ACKNOWLEDGEMENT

The author wishes to thank Marco Dorigo and the anonymous referees who greatly helped to improve the presentation of the ideas in this paper.

References

[1] K. Aihara. Chaotic neural networks. *Physical Letters A*, Vol. 144, No. 6, pp. 333–340, 1990.

[2] M. Asada. Map building for a mobile robot from sensory data. *IEEE Trans. Syst. Man Cybern.*, Vol. 37, No. 6, pp. 1326–1336, 1990.

[3] R.D. Beer. A dynamical systems perspective on agent-environment interaction. *Artificial Intelligence*, Vol. 72, No. 1, pp. 173–215, 1995.

[4] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks*, Vol. 5, No. 2, pp. 157–166, 1994.

[5] R. Brooks. A robust layered control system for a mobile robot. *IEEE J. Robotics and Automat.*, Vol. RA-2, No. 1, pp. 14–23, 1986.

[6] M. Colombetti and M. Dorigo. Training agents to perform sequential behavior. *Adaptive Behavior*, Vol. 2, No. 3, pp. 247–275, 1994.

[7] J.P. Crutchfield. Inferring statistical complexity. *Phys Rev Lett*, Vol. 63, pp. 105–108, 1989.

[8] M. Dorigo and M. Colombetti. Robot shaping: developing autonomous agents through learning. *Artificial Intelligence*, Vol. 71, No. 2, pp. 321–370, 1994.

[9] T. Eisenhammer, A. Hubler, T. Geisel, and E. Luscher. Scaling behavior of the maximum energy exchange between coupled anharmonic oscillators. *Physical Review*, Vol. A-41, pp. 3332–3342, 1990.

[10] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE J. Robotics and Automation*, Vol. RA-3, No. 3, pp. 249–265, 1987.

[11] J.L. Elman. Finding structure in time. *Cognitive Science*, Vol. 14, pp. 179–211, 1990.

[12] T. Endo and S. Mori. Mode analysis of a ring of a large number of mutually coupled van der Pol oscillators. *IEEE Trans. Circuits Syst.*, Vol. CAS-25, No. 1, pp. 7–18, 1978.

[13] F. Freyberger, P. Kampman, and G. Schmidt. Constructing maps for indoor navigation of a mobile robot by using an active 3D range imaging device. In *Proc. of the IEEE Int. Workshop on Intelligent Robots and Systems '90*, pp. 143–148, 1990.

[14] C.L. Giles, G.Z. Sun, H.H. Chen, Y.C. Lee, and D. Chen. Higher order recurrent networks and grammatical inference. In D.S. Touretzky, editor, *Advances in Neural Information Processings 2*, pp. 380–387. San Mateo, CA: Morgan Kaufmann, 1990.

[15] S. Harnad. The symbol grounding problem. *Physica D*, Vol. 42, pp. 335–346, 1990.

[16] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley, 1979.

[17] S. Iida and S. Yuta. Vechicle command system and trajectory control for autonomous mobile robots. In *Proc. of the IEEE/RSJ Int. Workshop on Intelligent Robots and Systems '91*, pp. 212–217, 1991.

[18] M.I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proc. of Eighth Annual Conference of Cognitive Science Society*, pp. 531–546. Hillsdale, NJ: Erlbaum, 1986.

[19] M.I. Jordan. Indeterminate motor skill learning problems. In M. Jeannerod, editor, *Attention and Performances, XIII*. Cambridge, MA: MIT Press, 1988.

[20] M.I. Jordan and R.A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, Vol. 6, No. 2, pp. 181–214, 1994.

[21] M.I. Jordan and D.E. Rumelhart. Forward models: supervised learning with a distal teacher. *Cognitive Science*, Vol. 16, pp. 307–354, 1992.

[22] L.P. Kaelbling. An adaptive mobile robot. In *Proc. of the First European Conf. on Artificial Life*, pp. 41–47, 1992.

[23] S. Kambhampati and L.S. Davis. Multiresolution path planning for mobile robots. *IEEE J. Robotics and Automation*, Vol. RA-2, No. 3, pp. 135–145, 1986.

[24] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The Int. J. of Robotics Research*, Vol. 5, No. 1, pp. 90–98, 1986.

[25] T. Kohonen. Self-organized formation of topographically correct feature maps. *Biological Cybernetics*, Vol. 43, pp. 59–69, 1982.

[26] J.R. Koza. Evolution of subsumption using genetic programming. In *Proc. of the First European Conf. on Artificial Life*, pp. 110–119, 1992.

[27] B.J. Krose and M. Eecen. A self-organizing representation of sensor space for mobile robot navigation. In *Proc. of the Int. Conf. on Intelligent Robotics and Systems '94*, pp. 257–263, 1994.

[28] B.J. Krose and J.W. van Dam. Adaptive state space quantisation for reinforcement learning of collision-free navigation. In *Proc. of the IEEE Int. Workshop on Intelligent Robots and Systems '92*, pp. 9–14, 1992.

[29] B. Kuipers. A qualitative approach to robot exploration and map learning. In *AAAI Workshop Spatial Reasoning and Multi-Sensor Fusion (Chicago)*, pp. 774–779, 1987.

[30] L.-J. Lin and T.M. Mitchell. Reinforcement learning with hidden states. In *Proc. of the Second Int. Conf. on Simulation of Adaptive Behavior*, pp. 271–280, 1992.

[31] P. Maes, editor. *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*. Cambridge, MA: MIT Press, 1991.

[32] M. Mataric. Integration of representation into goal-driven behavior-based robot. *IEEE Trans. Robotics and Automation*, Vol. 8, No. 3, pp. 304–312, 1992.

[33] J.A. Meyer and S.W. Wilson, editors. *From Animals to Animats: Proc. of the First International Conference on Simulation of Adaptive Behavior*. Cambridge, MA: MIT press, 1991.

[34] S. Nara and P. Davis. Memory search using complex dynamics in a recurrent neural network model. *Neural Networks*, Vol. 6, No. 7, pp. 963–974, 1993.

[35] F. Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, Vol. 59, pp. 2229–2232, 1987.

[36] J.B. Pollack. The induction of dynamical recognizers. *Machine Learning*, Vol. 7, pp. 227–252, 1991.

[37] D.A. Pomerleau. *Neural Network Perception for Mobile Robot Perception*. Boston: Kluwer, 1993.

[38] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J.L. Mclelland, editors, *Parallel Distributed Processing*. Cambridge, MA: MIT Press, 1986.

[39] M. Shidara, K. Kawano, H. Gomi, and M. Kawato. Inverse-dynamics encoding of eye movement by purkinje cells in the cerebellum. *Nature*, Vol. 365, pp. 50–52, 1993.

[40] C.A. Skarada and W.J. Freeman. Does the brain make chaos in order to make sense of the world? *Behavioral and Brain Sciences*, Vol. 10, pp. 161–165, 1987.

[41] J. Tani. Proposal of chaotic steepest descent method for neural networks and analysis of their dynamics. *Electronics and Communication in Japan, Part 3*, Vol. 75, No. 4, pp. 62–70, 1992.

[42] J. Tani and N. Fukumura. Embedding task-based behavior into internal sensory-based attractor dynamics in navigation of a mobile robot. In *Proc. of the IEEE Int. Conf. of Intelligent Robots and Systems '94*, pp. 886–893, 1994.

[43] J. Tani and N. Fukumura. Learning goal-directed sensory-based navigation of a mobile robot. *Neural Networks*, Vol. 7, No. 3, pp. 553–563, 1994.

[44] J. Tani and N. Fukumura. Embedding a grammatical description in deterministic chaos: an experiment in recurrent neural learning. *Biological Cybernetics*, to appear.

[45] I. Tsuda, E. Koerner, and H. Shimizu. Memory dynamics in asynchronous neural networks. *Prog. Theor. Phys.*, Vol. 78, pp. 51–71, 1987.

[46] Y. Uno, M. Kawato, and R. Suzuki. Formation and control of optimal trajectory in human multijoint arm movement. *Biological Cybernetics*, Vol. 61, pp. 73–85, 1989.

[47] B. M. Yamauchi and R. D. Beer. Sequential behavior and learning in evolved dynamical neural networks. *Adaptive Behavior*, Vol. 2, No. 3, pp. 219–246, 1994.

[48] U. R. Zimmer. Robust world-modelling and navigation in a real world. *Neuro Computing*, to appear.