HONOURS YEAR PROJECT REPORT

Mobile Robots That Learn to Navigate

Low Kian Hsiang

School of Computing

National University of Singapore
2000/01

Project No.	H40020
Supervisor	A/P Leow Wee Kheng, A/P Marcelo H. Ang Jr.
Deliverables	Report: 1 Volume

Abstract

A sensorimotor controller has been implemented to enable a mobile robot to learn its motion control autonomously and perform simple target-reaching movements. This controller is able to perform fine motion by reducing its self-positioning error and also, reach a designated target location with minimum delay. The control architecture is in the form of a neural network known as the Self-Organizing Map. Besides implementing the motor control and the online learning algorithms, the essentiality of a pre-learning phase is also evaluated. Then, we explore the possibility of incorporating a novel concept known as Local Linear Smoothing into our batch training algorithm; this notion allows the elimination of the boundary bias phenomenon. Lastly, we suggest a simple approach to learning in an obstacle-ridden environment.

Subject Descriptors

I.2.6	Learning
I.2.9	Robotics
I.5.1	Models

Keywords

Self-Organizing Map, Batch Training, Local Linear Smoothing, Online Unsupervised Learning, Sensorimotor Coordination, Obstacle Avoidance, Command Fusion Mechanism.

Implementation Software and Hardware

Linux Mandrake 7.2, Webots 2.0, Khepera Robot.

Acknowledgements

First and foremost, I would like to thank my supervisor, A/P Leow Wee Kheng, for his motivation and support to my project. His knowledge and enthusiasm have moulded my genre towards research, in particular neural network.

I would also like to show my sincere gratitude towards A/P Marcelo H. Ang Jr. and the Laboratory of Control and Mechatronics in the Mechanical Engineering Department for the loan of the Khepera robot.

I would like to thank Dr. Olivier Michel at Cyberbotics for his kind assistance in the usage of Webots.

A big thanks to my family and Ms. Cocoa Yeo Hwee Kiang for their immense support and encouragement.

Last but not the least, I would want to show my appreciation to Mr. Ho Ngai Lam for his knowledge in spatial data structures.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

This chapter identifies the problem domain and the specific objectives to be achieved. The structure of this thesis report is also presented.

1.1 Project Motivation

In recent years, an increasing amount of mobile robotics research has focused on the fundamental problem of autonomous goal-directed navigation in unstructured environments; this is essential for mobile robots operating in hostile conditions, like space, sea and contaminated habitats, as well as in the emerging field of service robotics, that include waste management [39], cleaning [55], luggage transfer, mail delivery, reconnaissance and handicap assistance. Nevertheless, a trivial solution cannot be derived due to the following impediments.

- Unstructured and dynamically changing navigation domain.
- Mobility constraints of the mobile robot such as non-linear motor control and degrees of freedom.
- Presence of systematic and non-systematic noise in the environment and the robot sensors and actuators [32].

It would therefore be a challenging task to evolve a robust and flexible navigation technique, which circumvents these constraints. To do so, it is our belief that tremendous insights can be drawn from biological agents, which have already achieved this task; the ability to successfully move through a novel environment is imperative to the survival of humans and animals. If the navigation capabilities of a mobile robot hinge on this analogy, it must be endowed with the ability of autonomous or unsupervised learning. This should be the primary inspiration for any of its navigational tasks.

Goal-directed navigation is a problem that is too complex to be tackled en bloc. Logically, we should employ a divide-and-conquer strategy to solve this problem; we can hierarchically and modularly decompose it into high-level, complex modules such as self-localization, map building, path planning and feature extraction and low-level, primitive motor control modules. This project focuses on the implementation of a low-level motor control module that enables a mobile robot to learn its motion control autonomously and perform simple target-reaching movements such as beacon aiming [40] or homing [6]. This will establish the most fundamental, yet crucial, foundation for subsequent higher-level modules to be mounted.

1.2 Research Scope and Objectives

Based on the reasoning of our motivation, the mobile robot can acquire its motor control skill through unsupervised learning. The implementation of the motor controller should achieve the following criteria.

- ***Learning Method*** :- The motor controller must be able to learn the association between the sensory input and the motor output without an external teacher. A fast rate of convergence in learning is also desired.
- ***Motor Control Performance*** :- The robot must be able to perform fine motion to reduce its self-positioning error and it must also be able to reach a designated

target location with minimum delay. These two requirements are in fact our performance metrics for the evaluation of our motor controller.

- ***Noise Tolerance*** :- The motor controller must be robust against systematic and non-systematic noise and still achieve the two above-mentioned objectives. To do this, it must be adaptable to environmental noise and changes and resistant to noise in the robot sensors and actuators.

The objectives stated above are intended for general mobile robotics applications and by no means exhaustive. Specific applications may require additional constraints.

1.3 Thesis Overview

Chapter 1 Introduction identifies the problem domain and the specific objectives to be achieved.

Chapter 2 Background and Related Work describes the research direction and some critical design considerations to evolve the unsupervised learning and motor control mechanisms. Following that, our proposed method is differentiated from other related work.

Chapter 3 Characteristics of Mobile Robot gives an overview of the robotic tools, the development environments and the resource constraints in the implementation of the sensorimotor controller. The approaches to circumvent these constraints are also discussed.

Chapter 4 Network Architecture of Sensorimotor Controller presents an overview of the network architecture as well as the details of our sensorimotor control and online unsupervised learning mechanisms. The need of the random activity generator to bootstrap the learning process is also evaluated. Lastly, we introduce a novel concept of local linear smoothing into our pre-learning phase and our batch training algorithm.

Chapter 5 Simulation Results and Analysis compares the performance of our online unsupervised learning algorithm, the inclusion of the pre-learning phase and our batch training algorithm using local linear smoothing. The real Khepera robot is also tested, based on one time step.

Chapter 6 Learning in an Obstacle-Ridden Environment suggests a mechanism to fuse our sensorimotor controller and the Braitenberg's vehicle type 3-C obstacle avoidance behavior.

Chapter 7 Discussion, Future Work and Conclusion does an analytical comparison with two pieces of related work. Future research directions have been proposed and the conclusion sums up my contributions to this project.

Chapter 2

BACKGROUND AND RELATED WORK

The background section describes the research direction and some critical design considerations to evolve the unsupervised learning and motor control mechanisms. Following that, we differentiate our proposed method with other related work.

2.1 Background

2.1.1 Motor Control in Mobile Robots

Recent work in behavioral robotics has shown much success in low-level navigational tasks [41, 42, 43, 44, 45, 46, 47]. Our low-level motor control task can similarly be translated into a simple, low-level navigational behavior, known as **homing**[1], which is predominant in most animals [1]. However, it is a mandatory skill towards the acquisition of more complex navigational behaviors [27, 28]; the general problem of learning to navigate between a given number of arbitrary locations or landmarks can be decomposed into simpler navigational components between one-one, many-one and one-many locations as shown in Fig. 1.

		Destination Locations	
		One	Many
Source Locations	One	Simple Interlocation Navigation	Example: Foraging
	Many	Example: Homing	General-Purpose Navigation

Fig. 1 A hierarchical decomposition of the general navigation problem.

[1]It can be defined as the ability of an autonomous agent to navigate to a particular "home" location from arbitrary locations within a specific environment.

To acquire this low-level homing behavior, the mobile robot must gain knowledge of its motion control, otherwise known as kinematics, in a given environment. This relates to **sensorimotor coordination**[2]. How then can the mobile robot be acquainted with its own kinematics?

Our research rooted off by examining the control architectures of **robot arms** [5, 8, 9, 14, 15, 36, 37] to identify the underlying concepts behind automatic end effector positioning; these concepts may be applicable to our cause. In doing so, we have to observe that the direct translation of learning and sensorimotor control techniques from robot arms to mobile robots is not possible due to several critical constraints. Let me illustrate a few examples to substantiate my claim.

Firstly, the arbitrary target location of a mobile robot is specified in a 2D environment, instead of a 3D environment for the robot arm. Secondly, the workspace of the mobile robot is not limited by the robot's structure, unlike that of a robot manipulator; the mobile robot must make movements to targets at arbitrary distances and angles. In my opinion, it is not feasible to train the mobile robot to sample all possible movements in an arbitrarily large workspace (**global map space**) due to scalability of network size and training time. Thus, we have adopted an incremental movement approach (**local perceptual space**) such that each resulting movement is held over a short space-time interval. When the target is beyond the maximum distance and/or angle in the local perceptual space, our motor controller generates wheel speeds that will move the robot to the largest distance and/or angle in this space. As a result, the robot will incrementally turn and move towards the target at the maximum rate. Once the target falls within this space, the motor wheel speeds will be reduced until the robot comes to a stop over the target.

[2]This term refers to the association of signals coming from various sensory modalities, such as audition [59], vision [7], sonar [29], infrared [30] and even smell [31], to motor commands in view of a given task, such as homing [7] or obstacle avoidance [30].

Thirdly, the mobile robot does not possess redundant degrees of freedom; we do not need to deal with issues of redundancy such as smoothness of robot movements, rate of reaction and fatigue [10]. Lastly, the mobile robot cannot generate arbitrary movements; it cannot be displaced laterally with a single movement.

We have also realized that the classical approach to motor control is to assume an ***"a priori"*** model of correlation between the sensors and motor commands; it is a mathematical model built on the foundation of physical laws of kinematics. The forward kinematics (motor commands to sensors) is utilized in a popular localization or self-positioning technique known as dead reckoning[3]. Based on proprioceptive sensor data such as displacement encoders, the momentary position of the mobile robot, relative to a known start position, can be computed using simple geometric equations (a forward kinematics model) shown in [32]. This method cannot be employed for long distances as it suffers from various drawbacks; the kinematics model always has some inaccuracies, encoders have limited precision and external sources affecting the robot's motion, such as wheel slippage, are not observable by the sensors. Thus, the self-positioning error accumulates over time. The inverse kinematics (sensors to motor commands) can be represented by mathematical equations [5] and it also suffers the same drawbacks. The *"a priori"* model denotes the ideal kinematics behavior, observed only in simulated environments, which is extremely noise sensitive. In reality, the real robot deviates significantly from this model. How can we produce a replica closer to the true model of the real robot? The solution lies in the next section.

[3]Dead reckoning refers to the process of updating an estimate of one's current position based on self-knowledge of time, speed and direction of self-motion.

2.1.2 Learning Paradigms

An alternative approach to the *"a priori"* model has to be considered; the ***"a posteriori"*** model is built on observed data of the correlation between exteroceptive sensor perceptions, such as sonar or vision, and motor actions collected during the robot's navigation. Therefore, the model does not require knowledge of the robot's kinematics. These pairs of real-time perception-action data are used to estimate the parameters of the model, thus accounting for the noisy components. The final product of this model can even move a "semi-handicapped" robot correctly to its destination. This whole approximation problem is in fact a learning process. It is our intention to investigate and implement this process.

Should this learning process be supervised or unsupervised? **Supervised learning** methods are straightforward and relatively easy to implement but there are serious shortcomings. Firstly, the external teacher's knowledge used in the supervised training is usually varied, in the case of multiple external supervisors, and biased in terms of distribution, sequence and selection of the training samples in the learning set. Secondly, it is not adaptable to novel environments. The whole learning process has to be repeated with the involvement of the external teacher. On the contrary, **unsupervised learning** techniques have overcome such problems due to its robustness and flexibility. No external supervisor is required, thus eliminating the bias in the training samples. As mentioned before in Section 1.1, unsupervised learning allows the mobile robot to adapt easily to a novel environment. Thus, unsupervised learning will be used to train our motor controller.

Many unsupervised learning algorithms have been dedicated to the application of sensorimotor coordination on robot manipulators and mobile robots. They include evolutionary optimization or genetic algorithms [33], rule-based algorithms [34], fuzzy logic [35], artificial neural networks [5, 6, 7, 8, 9, 14, 15, 30, 36, 37, 48, 49, 50, 60] and reinforcement learning [38]. The choice of the learning algorithm must take into consideration all the objectives stated in Section 1.2. In particular, artificial neural network fits our purpose extremely well; it replicates the cerebellum of a mammalian brain, which performs sensorimotor coordination. We shall now discuss how and why this is so.

Our approach stems from the process of **circular reaction** in Fig. 2, presented by M. Kuperstein et al. [3]. The current use of circular reaction is an extension of one of J. Piaget's development stages [4]. This unsupervised learning-by-doing cycle operates in two phases. The initial training phase involves the learning of the sensorimotor relations via correlations between input and output signals while the performance phase uses the learned correlations to evoke the correct motor signals to move to the target location. The sequences within these two phases are briefly mentioned in Fig. 2.

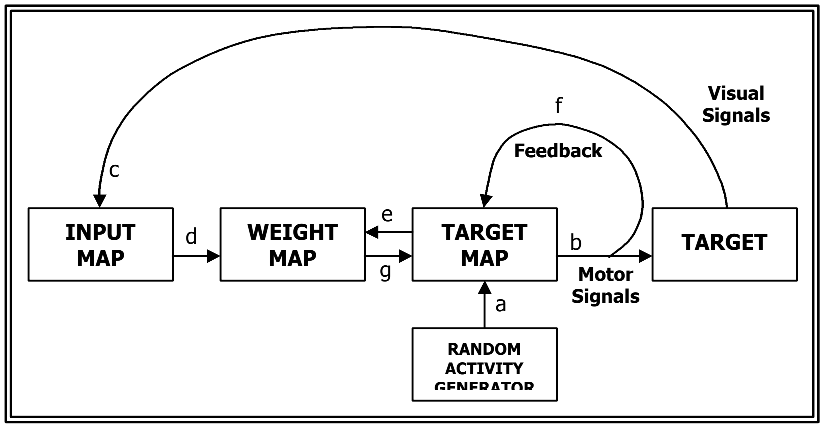


Fig. 2 The circular reaction. Self-produced motor signals are correlated with sensory signals. The sequence for training is a, b, c, d, (e+f), g. Correlated learning is done in step g. After this correlation is achieved, sensory input signals can evoke associated motor output signals to accurately move to the target location. The sequence for performance is c, d, e, b.

From the same diagram, we can observe two issues of interest. Firstly, how do we integrate the input map, weight map and target map modules so that the training process that interlinks these modules can be successful and efficient in producing an accurate weight map? In other words, how should we implement the sensorimotor control and learning mechanisms? Secondly, how effective is the random activity generator in bootstrapping the learning process? Can we eliminate this random activity generator, restructure the usual procedures in the two phases, such that both phases run concurrently in a "split brain" fashion, and still achieve a reasonable quality in learning? We shall now investigate the first problem and postpone the discussion of the second problem to Chapter 4.

2.1.3 Self-Organizing Map

To solve the first problem, various neural network architectures for mobile robots have been proposed [5, 6, 7, 60]. They have a common goal, which is to autonomously learn the inverse kinematics of a mobile robot. An accurate weight map, previously shown in Fig. 2, must be obtained to verify the correctness of their solutions. Their approaches rely on a crucial concept known as **"linearization"**. It suggests that the non-linear sensorimotor transformation can be decomposed into local linear mappings recorded by the weight map. Thus, "linearization" is a viable adoption for our purpose. Our problem can now be narrowed down to the representation and interaction of the input map, weight map and target map modules in Fig. 2 to facilitate the learning process. This calls for the need to establish a map representation, which can assimilate learning and sensorimotor control.

Our biologically motivated framework is in the form of a self-organizing neural network, which performs "linearization". It takes inspiration from a robot arm experiment on end effector positioning realized by K.J. Schulten et al. [8, 9, 36, 37], which utilizes an extension of T. Kohonen's **Self-Organizing Map** Algorithm [2]. T. Kohonen's Self-

Organizing Map (SOM) is an unsupervised learning neural network method that produces a similarity graph of input data. It consists of a finite set of models, each approximating a local disjoint region in the open, unlimited set of input data. These models are associated with nodes or neurons that are arranged as a regular, usually two-dimensional grid. They are adapted by a learning process that automatically orders them on the two-dimensional grid along with their mutual similarity. The algorithm operates recursively in two simple steps. Upon each presentation of input data vector, it performs a search for the neuron with the closest[4] model vector. This "winning" neuron and its neighboring neurons are adapted by learning rules. Details of this algorithm will be presented in Chapter 4.

Two immediate advantages of SOM can be observed. Firstly, it performs dimensionality reduction of the input data. Secondly, SOM can first be computed using any representative subset of old input data and new input items can be mapped straight into the most similar models without re-computation of the whole mapping. SOM has been widely utilized in several fields such as machine vision and image analysis, robotics, data processing, linguistic and AI problems, neurophysiological research and etc. Examples of these applications can be found in [53] and [54], which are bibliographies used for our search of SOM-related materials. [51, 52] are the other SOM resources used for the implementation of our motor controller.

Apart from the SOM weight map, the representation of sensory input map and the motor target map must also be determined. Should the sensory input map be the global map space of the environment or the local perceptual space of the mobile robot? The global map space is typically employed by the traditional planner-based or **deliberative strategies**. They rely on this centralized world model to verify sensory information and

[4]The distance measure may be, for instance, Manhattan or Euclidean distance.

generate actions in the world [48, 49, 50]. The information in the world model is used by the planner to produce the most appropriate sequence of actions for the robot. Thus, it is not a direct result of sensory data but rather an outcome of a series of sense-model-plan-act stages. Uncertainties in sensing and action and environmental changes can require frequent re-planning, which amounts to high computational cost. Planner-based approaches have been criticized for scaling poorly with the complexity of real-world problems, and making real-time reaction to sudden world changes impossible.

Conversely, the local perceptual space is utilized by **reactive approaches**, which can achieve real-time performance in autonomous travel [5, 7, 60]. They maintain no internal models. Typically, they apply a simple functional mapping between sensory stimuli and appropriate motor responses, usually in the form of a lookup. They have the same theme of constant-time run-time direct encodings of the appropriate action for each input state. These mappings rely on a direct coupling between sensing and action and fast feedback from the environment. Thus, these strategies for low-level sensorimotor control tasks have proven effective in reacting to dynamic changes in the environment.

How should the motor target map be represented? Should it be discretized into specific, **state-based** events such as ‘move forward’, ‘move backwards’, ‘turn left’, ‘turn right’, ‘rotate on the spot’ and so on [30]? Recall that one of our objectives in Section 1.2 requires fine motion in the mobile robot to reduce its self-positioning error. Clearly, the previous representation does not permit this. Therefore, a **continuous motor output space** is adopted.

2.2 Related Work

2.2.1 Zalama, Gaudiano and Coronado (NETMORC)

A Neural NETwork MObile Robot Controller (NETMORC) is introduced by E. Zalama et al. [5, 60] to autonomously learn the motion control of the mobile robot. The network architecture essentially comprises of two arrays of 1-D neurons encoding distance and angle of the robot displacement and a 2-D grid of neurons encoding the motor wheel velocities. Each neuron in the two arrays has connection weights to every neuron in the 2-D grid. During the learning phase, random motor wheel velocities are generated to move the robot and the corresponding displacements are observed. Each velocity-displacement pair is pattern-matched against the velocity information encoded in the 2-D grid neurons and the distance and angle information encoded in the two respective sets of 1-D array neurons. The connection weights of the "fired" neurons in the 1-D arrays are trained accordingly using Grossberg's outstar learning rule. In the performance phase, the trained network is used to produce corresponding motor wheel velocities from the input of distance and angle to target. If the target lies beyond the local perceptual space of the robot established in the learning phase, it would need a few incremental moves to reach the target. At the end of each move, the sensory system updates the robot with its new position with respect to the target so that the next move can be determined. This goes on until the robot stops over the target location. This robot controller has been tested in simulations and implemented on the Robuter robot.

2.2.2 Versino and Gambardella (Invertible Kohonen Map)

C. Versino and L.M. Gambardella [7] demonstrated the utilization of a two-dimensional

Self-Organizing Map (SOM) to autonomously learn the motion control of the mobile robot. Each neuron in the SOM encodes a motor weight and a displacement weight. During the learning phase, a set of training samples is collected by randomly generating motor wheel speeds to move the robot and observing the corresponding displacements. The motor component in each training sample is used to determine the "winning" neuron by its closeness to the neuron's motor weight. The weights of this node and of its neighbors are adapted to the training sample through the learning rules of SOM. We can observe that the SOM is trained in the forward mode on a transformation from the space of motor commands to the space of sensory perceptions. In the performance phase, the SOM is used in backward mode. To move to a designated target location, the distance and the angle of the target with respect to the robot serve as the sensory input to the SOM. The network returns a motor output to move the robot towards the target. If the target lies beyond its local perceptual space, the same approach, as discussed previously, is adopted. The network performance has been tested both in a computer simulation and on the Khepera robot.

2.2.3 Our Proposed Method ('Extended' Self-Organizing Map)

These two pieces of related work have been selected for comparison because their design considerations are similar to ours: unsupervised learning, reactive motion control, local perceptual sensory space, and continuous motor output space. However, our work can be significantly distinguished from theirs by the following key features.

- ***Concurrency of Learning and Performance Phases*** :- The two pieces of related work segregate the learning phase and the performance phase such that the former phase precedes the latter. We argue that the two phases can be integrated to operate simultaneously. The simulation results in Chapter 5 prove its feasibility.

- ***Segmentation of Input Space*** :- "Linearization" results in the discretization of the sensory input space into small disjoint regions, each governed by a neuron. How should the space be segmented? In NETMORC, the input space is manually segmented into a fixed, regular topology. This implies that the network has no self-organizing power. Thus, a low self-positioning error can only be achieved by adding enough neurons. Our proposed method takes on an irregular, self-organizing topology, which is automatically shaped by the robot's motion in its environment. This has an effect of dense partitioning in the sensory area used for fine positioning and sparse partitioning in the sensory area that is rarely traversed. This self-organizing ability thus improves the performance and scalability of the network.
- ***Immediate Output of Network*** :- NETMORC and the Invertible Kohonen Map maps the sensory input space directly to the motor output space. Our method assumes a different perspective; the direct output of our SOM characterizes the transformation component that is responsible for the mapping from the sensory input space to the motor output space. We argue that this can overcome certain limitations of the existing approaches, which shall be discussed in Chapter 7.

- ***Online vs. Batch Training*** :- Our method is the first to introduce the notion of online learning and batch training to the motion control of a mobile robot. Our work implements and compares the two forms of learning.

A more detailed analytical comparison of our proposed method with these two pieces of related work will be made in Chapter 7.

Chapter 3

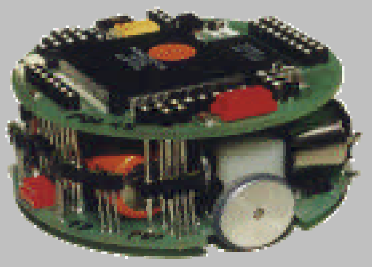
CHARACTERISTICS OF MOBILE ROBOT

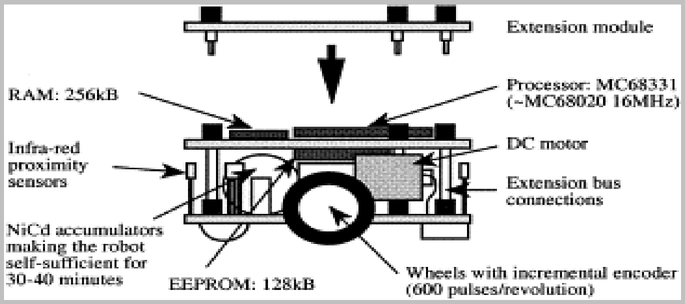
This chapter gives an overview of the robotic tools, the development environments and the resource constraints in the implementation of the sensorimotor controller. The approaches to circumvent these constraints are also discussed.

3.1 Khepera : A Miniature Mobile Robot

3.1.1 Technical Specifications

This mini-mobile robot is developed at MicroComputing and Interface Lab (LAMI) – Swiss Federal Institute of Technology, Lausanne (EPFL) by Edoardo Franzi, Paolo Ienne and Francesco Mondada [26]. It is a commercial product distributed by K-team S.A. (http://www.k-team.com). Fig. 3 describes its basic configuration.




Extension module
RAM: 256kB
Processor: MC68331 (~MC68020 16MHz)
Infra-red proximity sensors
DC motor
Extension bus connections
NiCd accumulators making the robot self-sufficient for 30-40 minutes
EEPROM: 128kB
Wheels with incremental encoder (600 pulses/revolution)

COMPONENT	DESCRIPTION
Processor	Motorola 68331 microcontroller.
RAM	256 Kbytes.
ROM	128 or 256 Kbytes.
Motion	2 DC motors with incremental encoder of about 12 pulses/mm resolution.
Sensors	8 IR proximity and light sensors with range of 4-5 cm and 15-16 cm respectively.
Power	Rechargeable NiCd batteries, allowing 30 min. of maximal activity, or external power supply.
Extension Bus	Additional turrets for vision, manipulator and inter-robot communications can be mounted (Fig. 4 and Fig. 5 on the next page).
Size	Diameter: 55 mm. Height: 30 mm.
Weight	About 70 g.

Fig. 3 Technical Specifications of Khepera robot.

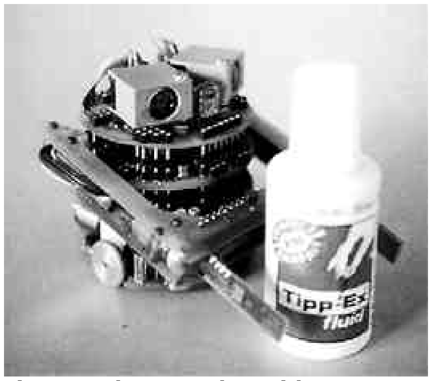


Fig. 4 Khepera robot with stereoscopic vision and gripper turrets.

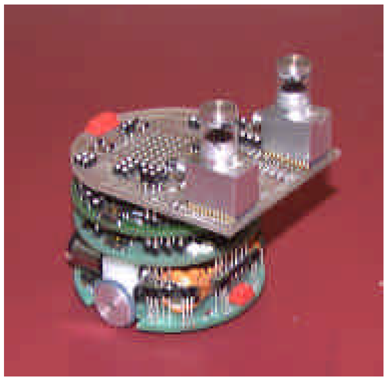


Fig. 5 Khepera robot with a twin panoramic vision turret.

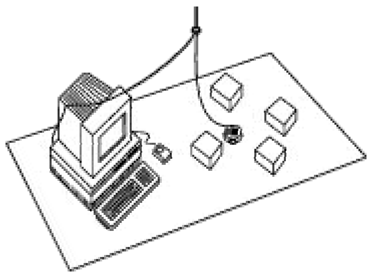
3.1.2 Features and Advantages

- ***Easy to Use and Control*** :- Robotic tool for research, experiments and prototyping. It allows quick entry into collaborative, evolutionary and neurobiological robotics research and also, real world testing of algorithms developed in simulation for trajectory planning, obstacle avoidance, wall following, target searching, collective behavior and hypotheses on behavior processing, among others. The Khepera robot can be manipulated easily.
- ***Easy to Install*** :- It functions like a plug and play device. The serial connection between the PC and the robot can be made with an aerial cable without problems.
- ***Low Cost*** :- The environment is also easy and cheap to build.
- ***Robust*** :- There is no need of an electronic engineer to be consulted every week.
- ***Compact*** :- The user can perform experiments on a small area (i.e. a table top) with everything at hand (PC, robot, environment). With similar functionality to larger robots used in research and education, the miniature robot is relatively much more robust than a big one. Imagine the Khepera robot, 55 mm in diameter, running against a wall at a speed of 50 mm/s. Now compare it with a big robot, 1 m in diameter, going against a

wall at a speed of 1m/s. Will the wall or the robot survive? Thus, its compactness makes it practical to use, as it does not pose any danger.

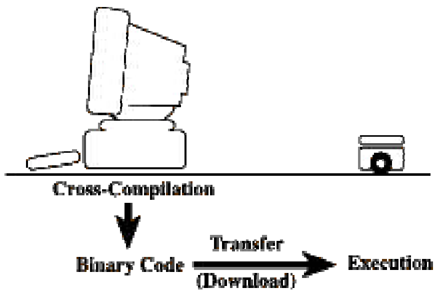
- ***Easy Transportability*** :- To ship or to take the robot to a conference won't be a problem anymore.
- ***Modularity*** :- At software level, an efficient library of on-board applications for controlling the robot, monitoring experiments, and downloading new software has already been implemented. Refer to Section 3.1.3 for more details. At hardware level, a large number of extension modules (Fig. 4 and Fig. 5) makes it adaptable to a wide range of experimentation.
- ***High Computational Power***
- ***Easy to Program*** :- A number of standard or graphical software environments has already been developed. Refer to Section 3.1.3 for more details.

3.1.3 Development Methodology and Environments



There are three possible configurations of the development environments.

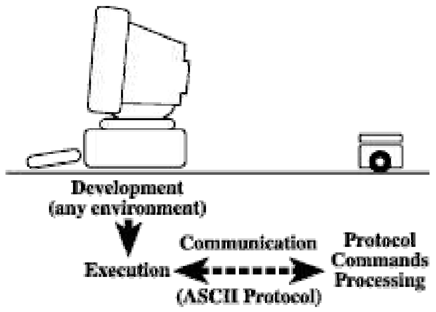
1. *Cross Compiler Method*



The oldest and most well known development methodology consists of building the binary code for the robot on the PC, transferring it to the robot and running it on the robot. A library of procedures is available to control the robot hardware; wheel speed and position control, sensor reading, multitasking management and many other features are included in the BIOS of the robot and can be used from a C program.

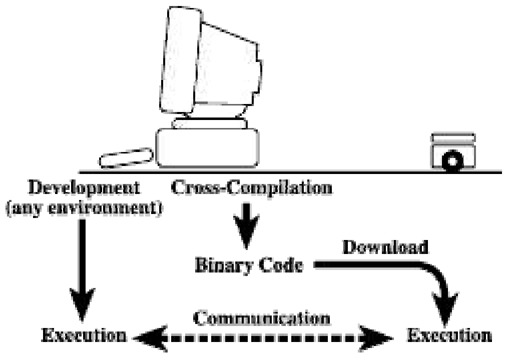
This configuration has the advantage of allowing the robot to run independently from the computer, but the disadvantage is that the user has no access to the running code. This is very problematic for debugging the program and for understanding the tested control program. One way to resolve this is to test and debug the control program on Webots: a mobile robot simulator (Section 3.2) first before running it on the real robot. This simulator automatically builds the binary code from the control program and directly downloads the binary code to the real robot.

2. *Remote Control Method*



Most of the actual users of these robots work by running the control program on a PC in their own programming environment (C, C++, SysQuake, LabVIEW, MATLAB, Webots) by controlling the robot through a standard RS232 serial link. Khepera has a running mode called "SerCom", where all primitive functionalities of the robot, like wheel speed and sensor reading, can be controlled with simple ASCII commands sent on the RS232 serial line.
This configuration splits the control of the robot between the local processor placed on the robot and the host PC. The robot processor manages all strict real-time tasks (motor control) and the PC manages the higher-level, more computationally demanding tasks. Having this part of the control program on the PC allows an easier development (known environment and good debug tools) and a much better understanding of the system due to the possibility of a much better interaction (graphical representation, mouse and keyboard which are not present on the robot).
Refer to [58] and [61] for examples of this configuration.

3. *'Combination of Both' Method*



When users start to understand their algorithms and want another form of distribution of the tasks between robot and host computer such that the robot assumes a higher-level role now, they use a combination of the two previous methodologies. Both the robot and the host PC have their own set of running code, which communicates via the serial link. This configuration is very powerful, but needs a good knowledge of the system.
For example, our unsupervised sensorimotor controller may be hosted entirely on the robot while the PC conducts the high-level navigational tasks such as localization, map building, path planning and feature extraction. The PC instructs the robot via the serial link on the path to head while the robot sometimes feedbacks the sensory inputs to the PC to be stored as signatures for place recognition.

3.2 Webots : A Mobile Robot Simulator

Webots 2.0 is a realistic 3D mobile robots simulator from Cyberbotics (http://www.cyberbotics.com) [25] intended for use in the fields of autonomous agents, neurobiological modeling, intelligent robotics, evolutionary robotics, computer vision, and artificial intelligence research. It is a highly realistic modeling of Khepera, in terms of noise in sensors and actuators, with extension turrets for vision and manipulator. The user can test, debug and understand their control algorithms through the virtual robots using a C/C++ library before downloading onto the real robots via this simulator. A 3D environment editor allows the customization of various robotics scenarios shown in Fig. 6 and Fig. 7.

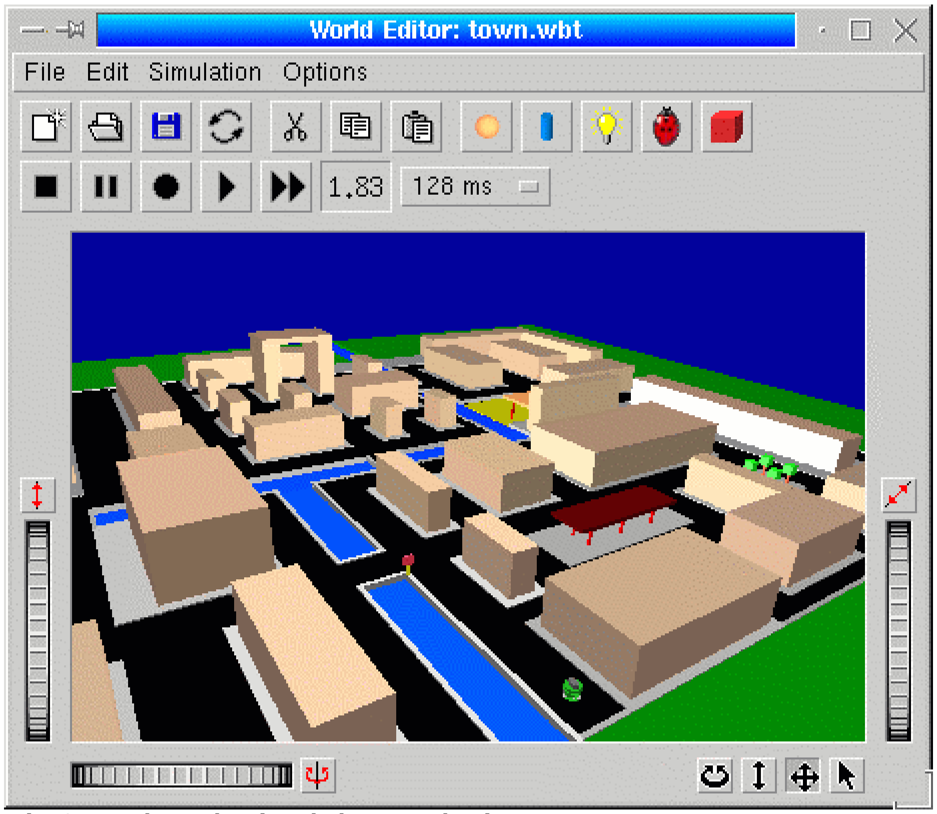


Fig. 6 Webots: the virtual Khepera robot in town.

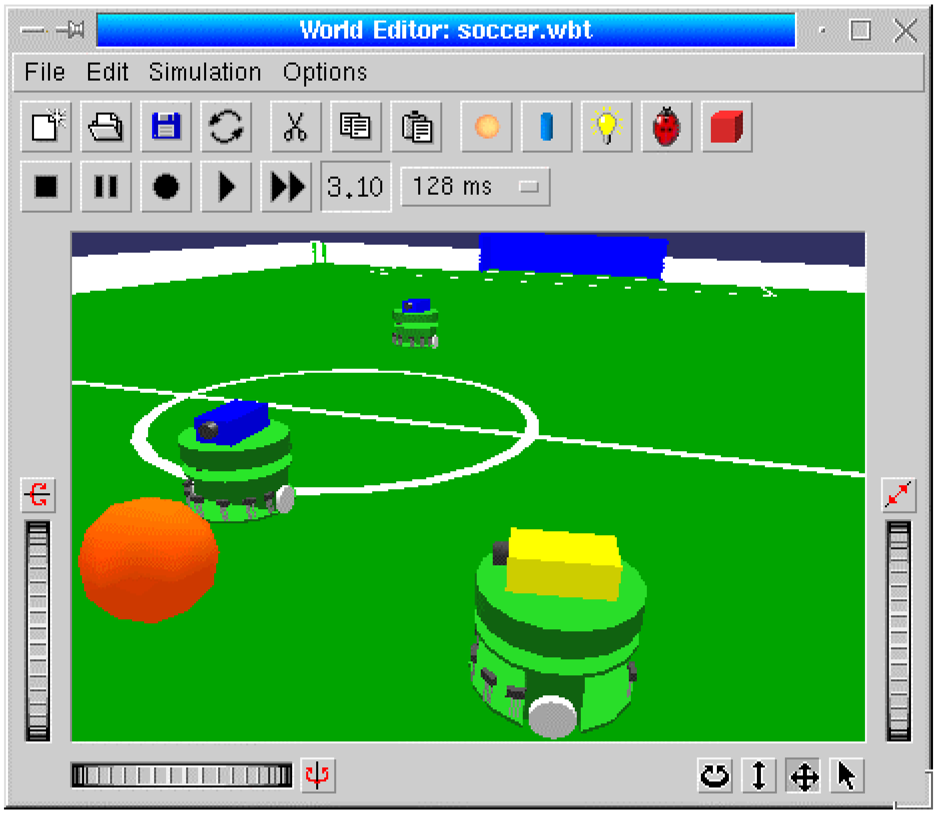


Fig. 7 Webots: the virtual Khepera robots in RoboSoccer.

The flexibility of this simulator allows the setup of all three configurations mentioned in Section 3.1.3.

3.3 Hardware and Software Limitations

Currently, the only available implementation tools are the real Khepera robot in its basic configuration (Fig. 3) and the Webots 2.0 simulator. There is no on-board or external vision system available for use. The eight on-board IR sensors are close range sensors, effective only for real-time reactive obstacle avoidance. Mid or long range sensors (vision, sonar) are very much necessary for the successful learning of homing behavior. Our local perceptual space approach only allows the performance assessment of the controller on the real robot based on one time step. Before this can happen, the controller must be sufficiently trained in Webots 2.0 simulator. Even so, this trained controller does not depict the true kinematics model of the real robot; the learning mechanism is not executed on it. We should therefore expect a tradeoff in performance. Nevertheless, it is still worth testing. Vision turrets are available in Webots 2.0 simulator, which can be mounted on the virtual Khepera robots. But the vision algorithms are missing. Since the vision problem is out of our scope of research, a good representation of the sensor input must be assumed such that it can be generalized to all forms of sensors. A viable sensory input would be the egocentric or robot-centered distance and direction to the target. The motor output is the internal motor speed of the robot's wheels. In the classical control theory, the association between the motor output and the sensory input is denoted by the transformation component known as the control parameters (This was first mentioned in Section 2.2.3). Our sensorimotor controller operates on this theory by accepting a sensory input of the above form and generating the corresponding control parameters as output. An appropriate motor output can then be derived from the two to move the robot towards the target. The next chapter describes the learning and the execution of this process thoroughly.

Chapter 4

NETWORK ARCHITECTURE OF SENSORIMOTOR CONTROLLER

This chapter presents an overview of the network architecture as well as the details of our sensorimotor control and online unsupervised learning mechanisms. The second problem mentioned in Section 2.1.2, that is the need of the random activity generator to bootstrap the learning process, is also evaluated. Lastly, we introduce a novel concept of local linear smoothing into our pre-learning phase and our batch training algorithm to eliminate the boundary bias in our sensory input space and possibly, speed up the rate of convergence in learning.

4.1 Framework Overview

The sensorimotor control or inverse kinematics problem of a mobile robot can be expressed in a simple mathematical form. Given the sensory input of the target location $\mathbf{u} \in \mathcal{U}$ and a corresponding motor output $\mathbf{c} \in \mathcal{C}$, the mapping is defined as

$$\mathbf{F} : \mathcal{U} \rightarrow \mathcal{C} \quad \Rightarrow \quad \mathbf{c} = \mathbf{F}(\mathbf{u}) \tag{1}$$

<u>Motor Control : Linear Case</u>

If this problem is linear, for a one-dimensional motor output c,

$$c = \mathbf{F}(\mathbf{u}) = \sum_{\forall i} m_i u_i = \mathbf{m}^{\mathbf{T}} \mathbf{u} \tag{2}$$

where $\mathbf{m}$ is the control parameters vector. For a multi-dimensional motor output $\mathbf{c}$,

$$\mathbf{c} = \mathbf{M}\,\mathbf{u} \tag{3}$$

where $\mathbf{M} \in \mathcal{M}$ denotes the control parameters matrix.

Motor Control : General or Non-Linear Case

The actual problem is, in fact, non-linear. We apply the concept of "linearization" here. The principle idea is to partition the sensory input space $\mathcal{U}$ into a set of disjoint regions, each governed by a neuron. Within each region, a local linear mapping of a form similar to equation (3) can be defined. Every neuron **r** stores two weights: a weight sensory vector $\mathbf{w_r}$, coding the region center and a control parameters matrix $\mathbf{M_r}$. Thus, the local linear mapping can be defined as

$$\mathbf{c} = \mathbf{M_s}\,\mathbf{u} \tag{4}$$

where **u** is in the region of neuron **s**. This neuron **s** can be determined using a simple nearest neighbor search defined by

$$\|\mathbf{w}_s - \mathbf{u}\| = \min_{\forall \mathbf{r}} \|\mathbf{w}_r - \mathbf{u}\| \tag{5}$$

This is illustrated in Fig. 8.

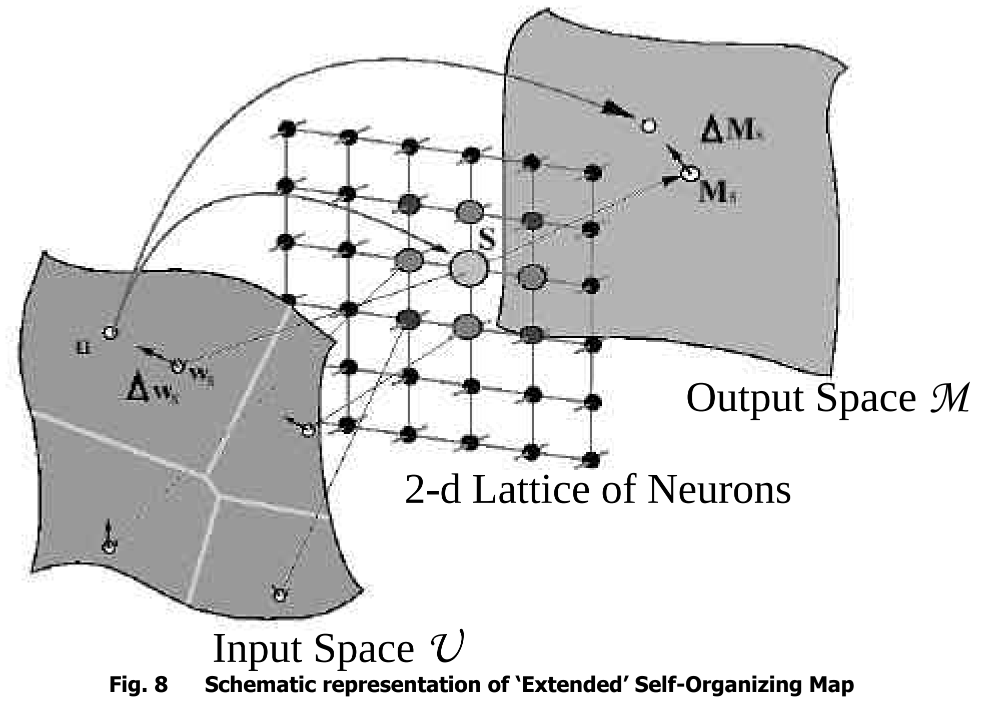


Fig. 8 Schematic representation of 'Extended' Self-Organizing Map Algorithm. The winning neuron is labeled "s" while the neighboring neurons are marked by different gray levels.

We can observe that the neurons are arranged into a 2-d grid lattice. Whenever a new target position vector **u** in the defined workspace or environment is presented, equations (4) and (5) will be used to locate the "winning" neuron **s**. In SOM, we can generalize by using the weights of this neuron and its neighboring neurons to determine the motor output vector **c** to move the mobile robot (This will be explained in the next section). At the same time, these weights are gradually refined through adaptive learning rules described in Section 4.3. The simulation results in Chapter 5 reveal that the influence of this form of collective neighborhood learning is two-fold; it increases the rate of convergence of learning and improves the self-positioning accuracy. This whole learning-by-doing process will be elucidated in the next two sections; Section 4.2 describes the sensorimotor control mechanism while Section 4.3 explains the online unsupervised learning mechanism. If we refer back to Fig. 2, the control and learning mechanisms correspond to the performance and training phases respectively. But in our instance, both mechanisms can run concurrently in a "split brain" fashion, compared to the sequential ordering of the two phases.

4.2 Sensorimotor Control Algorithm

Before explaining the steps in this algorithm, a few assumptions and notations must be explained. The robot is assumed to have a sensory system that can calculate an egocentric (robot-centered) representation of the distance d in meters and angle a in radians to arbitrary target locations. This is shown in Fig. 9. Therefore,

$$\mathbf{u} = [d, a]^T \tag{6}$$

where $d \in \mathcal{R}^+$ and $a \in (-\pi, \pi]$.

$$\mathbf{w_r} = [d_r, a_r]^T \qquad (7)$$

where $d_r \in [0, d_{max}]$ and $a_r \in (-\pi, \pi]$.

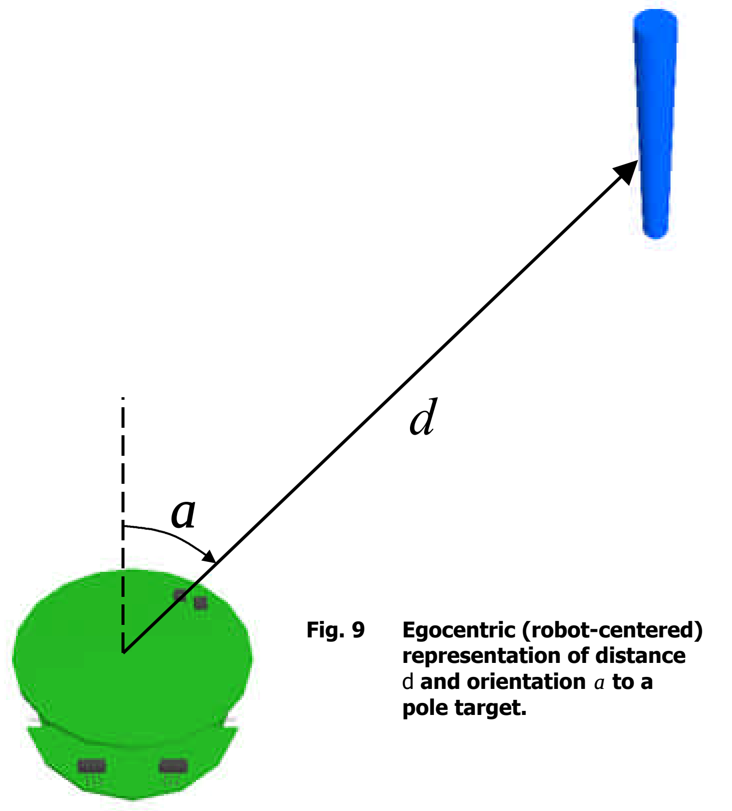


Fig. 9 **Egocentric (robot-centered) representation of distance** d **and orientation** *a* **to a pole target.**

Assuming that the mobile robot is differential drive with two wheels,

$$\mathbf{c} = [c_L, c_R]^T \qquad (8)$$

where L denotes the left wheel and R denotes the right wheel and $c_L, c_R \in [c_{min}, c_{max}]$. Lastly, we assume an instantaneous change in the internal motor wheel speeds.

Sensorimotor Control Algorithm

1. Present a target point in the workspace to the robot.
2. Let the sensory system observe the target position vector **u** of the target point.

3. **Selection of "Winning" Neuron**

Our nearest neighbor search in equation (5) cannot use Euclidean distance to determine the "winning" neuron **s** because d and a are different in type and domain range. Consider that, while a varies in a limited range of values (-π, π], the target distance d can be an arbitrarily large value. This means the target is extremely far but still within sensory range. If both d and a are scaled equally (which is the case for Euclidean distance), d would have had more importance in determination of the "winning" neuron. This can lead to undesirable effects such as always selecting the "winning" neuron, which corresponds to the highest motor speed on both wheels because this "winning" neuron matches the longest incremental distance traveled in one time step of t sec.

Our solution to this is derived from the following observation: it seems natural to give priority to a, so that the robot's first goal is to turn to face the target while on the move. d becomes really relevant when the robot is close to the target point so as to gradually reduce the motor wheel speed. The motor wheel speed will be reduced to zero when the robot reaches the target point. With this in mind, we can now state a set of feasible procedures.

a. Select a subset $\mathcal{K}$ of the $\mathcal{N}$ neurons in the 2-d grid lattice with the smallest a_s - a .

$$\left\| a_s - a \right\| = \min_{\forall r} \left\| a_r - a \right\| \tag{9}$$

b. Find the "winning" neuron **s** in $\mathcal{K}$ with the smallest weighted difference Q_s where

$$Q_s = \min_{i \in k} Q_i \tag{10}$$

$$Q_i = (b(d_i - d)^2 + g(a_r - a)^2)^{1/2} \qquad (11)$$

where $b >> g$.

Since 3a. has already competed using a, we can decrease the importance of a by making g a relatively small value. Nevertheless, its presence in equation (11) eliminates neurons in $\mathcal{K}$ with a_r that do not match a well. It also relaxes the requirement of choosing an optimal size for $\mathcal{K}$.

4. **Weighted Averaging of Motor Output Vector**

a. ***Simple Form***

We can use equation (4) to generate **c**. If c_L, $c_R > c_{max}$ or c_L, $c_R < c_{min}$, replace **c** by

$$\mathbf{c} = \mathbf{M_s}\,\mathbf{w_s} \qquad (12)$$

b. ***General Form***

$$\mathbf{c} = \frac{\sum_{\forall \mathbf{r}} \mathrm{h}(\mathbf{r},\mathbf{s})\,\mathbf{M_r}\,\mathbf{u}}{\sum_{\forall \mathbf{r}} \mathrm{h}(\mathbf{r},\mathbf{s})} \qquad (13)$$

where h(**r**, **s**) defines the Gaussian kernel weight of **r** - **s** .

$$\mathrm{h}(\mathbf{r},\mathbf{s}) = \exp\left(-\frac{\|\mathbf{r}-\mathbf{s}\|^2}{2s^2} \right) \qquad (14)$$

where **r** - **s** defines the lattice distance and s denotes the kernel bandwidth that decreases exponentially with time t. When $t >= t_{max}$, $s(t) = s(t_{max})$.

$$s(t) = s(0)\left(\frac{s(t_{\max})}{s(0)} \right)^{\frac{t}{t_{\max}}} \qquad (15)$$

This general form is known as the weighted/population averaging scheme whereby the neighborhood of the “winning” neuron is also considered for determining **c**. By employing this scheme, considerable generalization to novel situations is achieved due to the inherent interpolation evident in such a step. The above averaging method is a form of interpolation using radial basis functions [11] and is inspired by recent neurophysiological evidence [12] that the superior colliculus, a multilayered neuronal structure in the brain stem that is known to play a crucial role in the generation of saccadic eye movements, in fact employs a weighted averaging scheme to compute saccadic motor vectors. This scheme also allows the application of local linear smoothing, which will be the topic of discussion in Section 4.5.

If c_L, $c_R > c_{max}$ or c_L, $c_R < c_{min}$, replace **c** by

$$\mathbf{c} = \frac{\sum_{\forall \mathbf{r}} \mathrm{h}(\mathbf{r},\mathbf{s})\, \mathbf{M}_{\mathbf{r}}\, \mathbf{w}_{\mathbf{s}}}{\sum_{\forall \mathbf{r}} \mathrm{h}(\mathbf{r},\mathbf{s})} \tag{16}$$

Equations (12) and (16) are mandatory because in equations (4) and (13), if $d \rightarrow \infty$, c_L and $c_R \rightarrow \infty$ or $-\infty$. c_L and c_R will be delimited by c_{min} and c_{max}, making the robot move in the forward direction at c_{max} on both wheels or the backward direction at c_{min} on both wheels. This effect is undesirable, as a will be totally disregarded; the value of a is negligible, compared to d, in the computation of c_L and c_R in equations (4) and (13). This situation is remedied by substituting equations (4) and (13) with (12) and (16) respectively.

c. (Optional) If the robot’s internal motor speeds are in $\mathcal{Z}$, pass **c** through the sigmoid functions below.

$$c_L \leftarrow \begin{cases} c_{min} & \text{if } c_L <= c_{min}, \\ c_{max} & \text{if } c_L >= c_{max}, \\ \text{ceil}(c_L) & \text{if } c_L - \text{floor}(c_L) >= 0.5, \\ \text{floor}(c_L) & \text{otherwise.} \end{cases} \quad (17)$$

$$c_R \leftarrow \begin{cases} c_{min} & \text{if } c_R <= c_{min}, \\ c_{max} & \text{if } c_R >= c_{max}, \\ \text{ceil}(c_R) & \text{if } c_R - \text{floor}(c_R) >= 0.5, \\ \text{floor}(c_R) & \text{otherwise.} \end{cases} \quad (18)$$

d. Execute **c** in 1 time step size of t sec.

e. Let the sensory system observe the corresponding robot displacement vector **v**. A training sample (**v**, **c**) is thus collected for the online learning algorithm in Section 4.3 or the batch training algorithm in Section 4.6. This step 4e. allows sensorimotor control to be performed concurrently with learning. It can be omitted when the learning mechanism stops.

5. If the robot has not reached or stopped over the target location, continue with step 2.

4.3 Online Unsupervised Learning Algorithm

1. **Weights Initialization**

a. Initialize the 2-d Y(number of rows) by X(number of columns) lattice to $\mathcal{N}$ neurons where $\mathcal{N}=X\times Y$.

b. Initialize the synaptic weights of the $\mathcal{N}$ neurons in the 2-d lattice.

   i. Initialize the control parameters matrix $\mathbf{M_r}$ of each neuron to small random values. Code for the random numbers generator is obtained from [63].

   ii. Initialize the weight sensory vector $\mathbf{w_r}$ of each neuron in an "ordered" or topology-conserving manner to achieve a faster rate of convergence as pointed out in [2]. Choose a small subspace, totally enclosed by the sensory input space $\mathcal{V}$, which represent distances between 0 and D_{max} and angles between a_{min} and a_{max}. Initialize $\mathbf{w_r}$ of the four corner neurons in the 2-d lattice to $\mathbf{w_{1,1}}=[0, a_{min}]^T$, $\mathbf{w_{1,Y}}=[0, a_{max}]^T$, $\mathbf{w_{X,1}}=[D_{max}, a_{min}]^T$, $\mathbf{w_{X,Y}}=[D_{max}, a_{max}]^T$ as shown in Fig. 10.

$\mathbf{w_{1,1}}$	. . .	$\mathbf{w_{X,1}}$
⋮	⋮	⋮
$\mathbf{w_{1,Y}}$	. . .	$\mathbf{w_{X,Y}}$

Fig. 10 Assignment of $\mathbf{w_{1,1}}$,$\mathbf{w_{X,1}}$,$\mathbf{w_{1,Y}}$,$\mathbf{w_{X,Y}}$

iii. Initialize $\mathbf{w_r}$ of the remaining neurons through interpolation of the $\mathbf{w_r}$ of the four corner neurons using the following equations:

$$\mathbf{w}_{1,j} = \frac{j-1}{Y-1}\mathbf{w}_{1,Y} + \frac{Y-j}{Y-1}\mathbf{w}_{1,1} \qquad \text{for } j = 2,\ldots,Y-1 \tag{19}$$

$$\mathbf{w}_{X,j} = \frac{j-1}{Y-1}\mathbf{w}_{X,Y} + \frac{Y-j}{Y-1}\mathbf{w}_{X,1} \qquad \text{for } j = 2,\ldots,Y-1 \tag{20}$$

$$\mathbf{w}_{i,1} = \frac{i-1}{X-1}\mathbf{w}_{X,1} + \frac{X-i}{X-1}\mathbf{w}_{1,1} \qquad \text{for } i = 2,\ldots,X-1 \tag{21}$$

$$\mathbf{w}_{i,Y} = \frac{i-1}{X-1}\mathbf{w}_{X,Y} + \frac{X-i}{X-1}\mathbf{w}_{1,Y} \qquad \text{for } i = 2,\ldots,X-1 \tag{22}$$

$$\mathbf{w}_{i,j} = \frac{j-1}{Y-1}\mathbf{w}_{i,Y} + \frac{Y-j}{Y-1}\mathbf{w}_{i,1} \qquad \text{for } i = 2,\ldots,X-1\ \&\ \text{for } j = 2,\ldots,Y-1 \tag{23}$$

c. Initialize the time step parameter t = 0, time step size = t sec and maximum time step t_{max} to $\mathcal{Z}^+$.

2. **Adaptation Rules**

a. When a training sample (**v**, **c**) can be obtained from step 4e. of the sensorimotor control algorithm in the previous section, use step 3 of that algorithm to determine the “winning” neuron **s** with robot displacement vector **v**.

b. Determine the improved values of $\mathbf{w_r}$ of the “winning” neuron **s** and its neighboring neurons.

$$\text{D}\mathbf{w_r} = h \text{ h}(\mathbf{r}, \mathbf{s}) (\mathbf{v} - \mathbf{w_r}) \tag{24}$$

where h defines the learning rate that decreases exponentially with t. When t>= t_{max}, $h(t)$= $h(t_{max})$.

$$h(t) = h(0)\left(\frac{h(t_{\max})}{h(0)}\right)^{\frac{t}{t_{\max}}} \tag{25}$$

h(**r**, **s**) defines the Gaussian kernel weight of **r** - **s** and it is of the form in equation (14). If collective neighborhood learning is not desired,

$$\text{h}(\mathbf{r}, \mathbf{s}) = \begin{cases} 1 & \text{if } r = s, \\ 0 & \text{otherwise.} \end{cases} \tag{26}$$

c. Determine the improved values of $\mathbf{M_r}$ of the “winning” neuron **s** and its neighboring neurons.

$$\text{D}\mathbf{M_r} = h' \text{ h}'(\mathbf{r}, \mathbf{s}) \text{ D}\mathbf{M_r}' \tag{27}$$

where h' defines the learning rate of the form in equation (25) that decreases exponentially with t. When t>= t_{max}, $h'(t)$= $h'(t_{max})$. h’(**r**, **s**) defines the Gaussian kernel weight of **r** - **s** and it is of the form in equation (14). If collective neighborhood learning is not desired, use equation (26). D$\mathbf{M_r}$’ is defined such that it undergoes a stochastic gradient descent learning rule for the quadratic cost function

$$E = \tfrac{1}{2} \quad \mathbf{c} - \mathbf{M_r} \mathbf{v} \quad ^2 \tag{28}$$

and is given by an error correction rule of Widrow-Hoff type [13]

$$D\mathbf{M_r}' = r\ (\ \mathbf{c} - \mathbf{M_r}\ \mathbf{v}\ )\ \mathbf{v} \tag{29}$$

where $r \in [0,1]$ is the learning rate.

3. If $t <= t_{max}$, increase the time parameter t:

$$t = t + 1 \tag{30}$$

Continue with step 2.

Notice that the random activity generator used in Fig. 2 is totally left out of our online unsupervised learning algorithm. Realistically, can we omit this form of pre-learning phase? This issue is discussed in the next section.

4.4 Motor Babbling

The pre-learning phase, using the random activity generator, is termed the motor babbling phase; random motor output vectors are generated to perform movements for training the network. In this phase, homing movements cannot be performed and it incurs training time. Nevertheless, several mobile robot experiments [5, 7] and robot manipulator experiments [8, 9, 14, 15] emphasize on the importance of this phase in their simulations. In [14, 15], P. van der Smagt et al. reasoned that if the network starts with random weights, it is likely that it will always generate very similar motor output commands, which are subsequently reinforced since they constitute new learning samples. In other words, if only a few (similar) learning samples are taught to the network, the network is likely to generate a similar motor output independent of the applied sensory input. If this move is subsequently learned by the network, by reinforcement, all next generated displacements will be in the same direction. This implies that the input space may not be explored at all. By motor babbling, random movements are performed to train the network with the corresponding learning samples. Now, it will be able to generate movements in different directions and the input space can be explored.

The above analysis claims that our robot, equipped with the online learning algorithm in Section 4.3, faces the danger of not learning at all! This calls for the need to implement the motor babbling phase to complement and head start our learning algorithm; the motor babbling phase will be executed for a sufficient period of time enough to mitigate any risks of impedance to learning. Subsequently, our online learning algorithm takes over so that homing movements can be performed straight away. This implementation will also allow us to explore its benefits and compare its performance with our standalone online learning algorithm. Then, we can conclude whether this motor babbling phase is redundant or not. We shall now briefly describe the motor babbling process.

Motor Babbling

1. Generate a random motor output vector $\mathbf{c} = [c_L, c_R]^T$ where $c_L, c_R \in [c_{min}, c_{max}]$.
2. Execute **c** in 1 time step size of t sec.
3. Let the sensory system observe the corresponding robot displacement vector **v**. A training sample (**v**, **c**) is thus collected and stored.
4. Repeat steps 1 to 3 to gather a large enough training samples set $\mathcal{T}$.

The training rules in the motor babbling phase can either be online or batch. Online learning only requires the first three steps of motor babbling. Following that, our online unsupervised learning algorithm described in the previous section is used to train the network. On the other hand, batch training requires all the four steps of motor babbling so that learning can be conducted at each batch interval. Then a batch training algorithm is required.

Online learning reduces the utilization of huge memory storage and allows the possibility of real-time performance. Though these benefits turn into disadvantages for batch training, batch training has its own value. Often in online learning, we face the problem of heuristically determining an optimal or even feasible standard deviation for the random initialization of weight values. A wrong choice of standard deviation can deter the learning process. This difficulty is not encountered in batch training, as it does not rely on the initial weight values. However, training time can be drastically reduced if the initial weight sensory vectors for the neurons in the SOM are initialized in an 'ordered' manner rather than a random manner [2]. An effective 'ordered' map initialization scheme, proposed by M.C. Su [22, 23], is hence introduced.

Map Initialization for Batch Training

1. **Initialization of $\mathbf{w}_r$ of the neurons on the four corners** (Fig. 10)

   We first select a pair of training samples whose weighted Euclidean distance (equation 11) of their **v** coordinates is the largest one among the training set. The **v** coordinates of the two selected samples are used to initialize $\mathbf{w}_{X,1}$ and $\mathbf{w}_{1,Y}$. From the remaining training samples, the **v** coordinates of the sample which is "farthest" to the **v** coordinates of the two selected training samples is then used to initialize $\mathbf{w}_{1,1}$. $\mathbf{w}_{X,Y}$ is set to be the **v** coordinates of the training sample which is farthest to the **v** coordinates of the previously selected three samples.

2. **Initialization of the neurons on the four edges**

   Initialize the $\mathbf{w}_r$ of the neurons according to equations (19), (20), (21) and (22).

3. **Initialization of the remaining neurons**

Initialize the $\mathbf{w_r}$ of the remaining neurons from left to right followed by top to bottom using equation (23).

Can't we directly partition the input space into $X \times Y$ grids and use the coordinates of the grid centers to initialize the weights of the network? As we can observe in Fig. 11(a), this direct method tends to under-sample high probability regions and over-sample low probability ones. As a result, more iterations may be required to refine the map. Our proposed method, shown in Fig. 11(b), overcomes this effect.

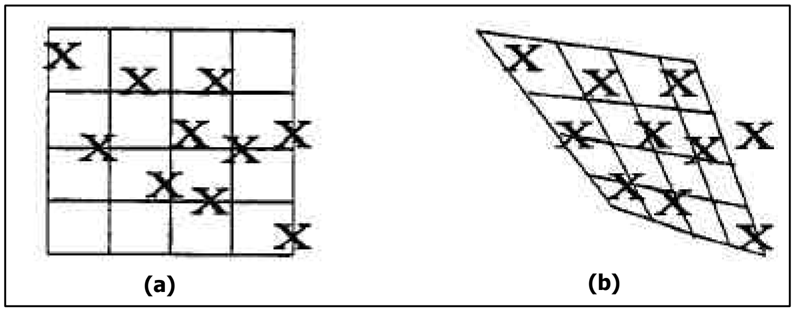


Fig. 11 Differences between direct partitioning method (a) and our initialization scheme (b). Each X represents a training sample.

In the next section, we propose a new concept called local linear smoothing in the implementation of our batch training algorithm.

4.5 Local Linear Smoothing

We can easily implement our batch training algorithm using T. Kohonen's Batch Map Algorithm [16, 17] to adapt the weight sensory vectors $\mathbf{w_r}$ and Batch Gradient Descent Algorithm [18] to adapt the control parameters matrices $\mathbf{M_r}$. Intuitively, the Batch Map algorithm works this way. Recall that each neuron governs a region in the input sensory space $\mathcal{U}$. Whenever a batch of $\mathcal{T}$ training samples is ready, these samples are dispersed into their respective regions. Then, each neuron uses the robot displacement vectors $\mathbf{v}$ of the samples in its region to calculate the mean. Finally, each neuron finds its new weight sensory vector $\mathbf{w_r}$ (local constant fit) by taking a Gaussian weight over the means of every neuron in its neighborhood. This is assuming that each region has equal number of samples. Otherwise, the Gaussian value used on each neuron is further weighted by the sample frequency in that neuron's region. Batch Gradient Descent can be performed in a similar manner, except that the sum of the change in the control parameters matrix, instead of the mean, is calculated over the samples.

These methods have a common flaw; the corner and edge neurons in the SOM will incur a border effect [62]. The cause of this boundary bias is that the Gaussian function cannot differentiate regions outside the local perceptual space with no samples but within its coverage and regions out of its coverage. It simply assumes that the function goes to zero everywhere outside its domain. Consequently, if the border lies within the bandwidth of the Gaussian spread, it will face a boundary bias. Fig. 12 illustrates a univariate example of this problem. Multivariate models like our SOM would naturally face a more severe bias.

What harm does this boundary bias do? Fan J. [20] explains that the rate of convergence at the boundary is slower. This means that if we look at the weight sensory vectors of the neurons, they will initially expand out to fill the local perceptual space and collapse towards the center due to the boundary bias, as shown in the simulation results in the next chapter. This goes on during the training time until the SOM receives enough training with a large sample set at the boundary. Thus, more time is incurred to reach final stability or convergence. The same goes true for the control parameters matrices. To note, our online unsupervised learning algorithm is also under the mercy of this phenomenon as it faces the same problem. We have to search for means to eliminate this boundary bias[5] so that a faster rate of network convergence can be achieved.

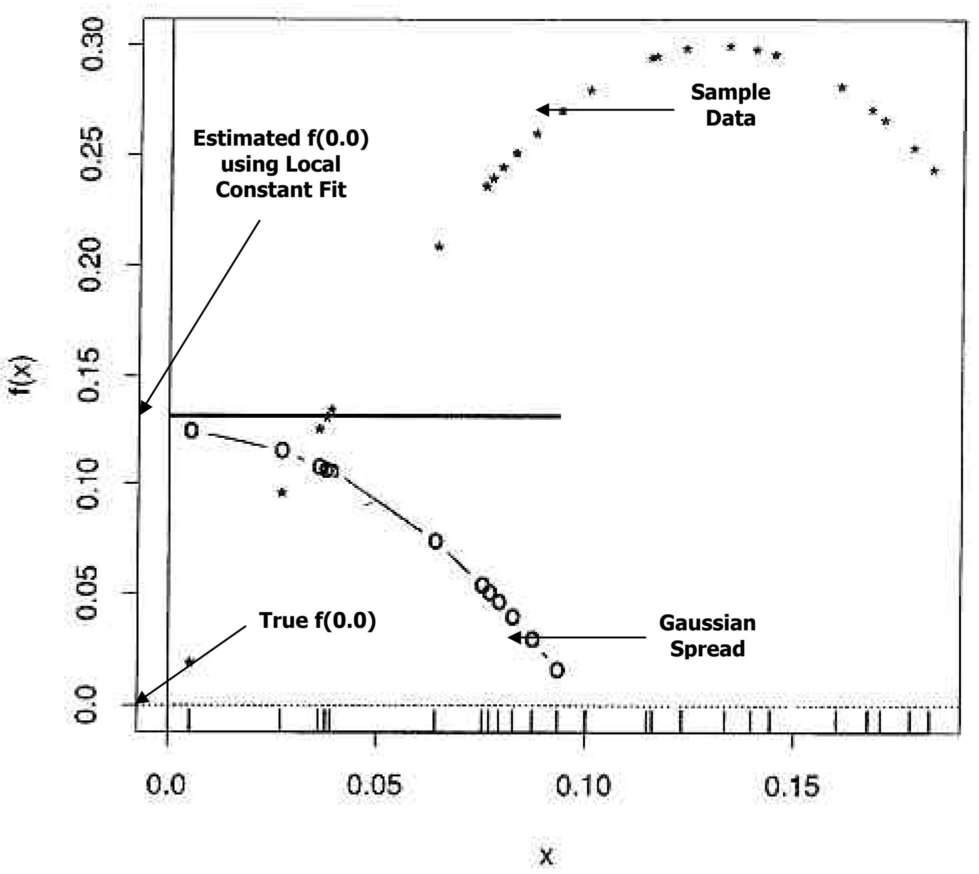


Fig. 12 Nadaraya-Watson estimate, boundary effects. x=0.0 is the boundary of the sample points distribution. Using T. Kohonen's Batch Map algorithm, the estimated sample mean will deviate substantially from the true mean at the boundary, similar to the instance shown in the diagram. This is termed the boundary bias.

T. Kohonen [16] has proposed a heuristic weighting rule for the boundary neurons but it is not intuitive, general or reliable. Moreover, it requires manual fine-tuning of the weighting factor. V. Cherkassky et al. [19] has interpreted our batch training modes as kernel estimation methods; these methods operate in a similar manner except that kernel weighting in a SOM is done in neighborhoods in the grid space rather than in the input space. Our batch training modes are commonly known as Nadaraya-Watson

[5]Note that eliminating the boundary bias does not mean eliminating all bias in the boundary regions. Rather, it means that the boundary regions are no more biased than the interior (non-boundary) regions.

kernel estimator. This estimator generally segments the entire sample data space into small local regions. In each region, an estimated local constant is derived to fit the local data. This estimator method faces a boundary bias, as explained previously.

A more superior kernel estimator, known as Local Linear Smoothing, has been derived to cure this effect [20, 21, 24]. How does it work? Instead of finding a local constant fit, we simply use a local line to fit the local data. I shall now explain how this can be achieved. In a small local neighborhood of the lattice grid point **s** in our 2-D SOM, if grid point **r** is in this same neighborhood, the weight sensory vector at **r** can be linearly defined as

$$\mathbf{w}_{\mathbf{r}} \approx \mathbf{w}_{\mathbf{s}} + \mathbf{w}_{\mathbf{s}}'(\mathbf{r} - \mathbf{s}) \qquad (31)$$

This equation is the line that we try to fit into the local data. The solution to our problem at hand can be found by estimating $\mathbf{w}_{\mathbf{s}}$. To do this, we try to find $\mathbf{w}_{\mathbf{s}}$ and $\mathbf{w}_{\mathbf{s}}'$ to minimize the weighted least squares equation.

$$\sum_{\forall \mathbf{r}} (\mathbf{w}_{\mathbf{r}} - \mathbf{w}_{\mathbf{s}} - \mathbf{w}_{\mathbf{s}}'(\mathbf{r} - \mathbf{s}))^2 \mathrm{h}(\mathbf{r}, \mathbf{s}) \qquad (32)$$

where h(**r**, **s**) defines the Gaussian kernel weight of $\|\mathbf{r} - \mathbf{s}\|$ and it is of the form in equation (14). By solving equation (32), we can obtain an estimate of $\mathbf{w}_{\mathbf{s}}$, which is given by equation (34) in the next section. We can do the same for the control parameters matrices. Fig. 13 shows how this local linear fit eliminate the boundary bias for a univariate model.

We can see from Fig. 13 that equation (32) causes a deformation in the Gaussian kernel near the boundary to the extent of assigning negative weight to those sample data farther away from the boundary, which drastically reduces the bias. The simulation results in Chapter 5 will show the disappearance of the boundary bias phenomenon and a faster rate of convergence in learning. I wish to point out that local linear smoothing only works if the

Gaussian spread covers more than one neuron. Without the existence of a neighborhood, i.e. if only one neuron is affected, boundary bias can never be experienced, even for Nadaraya-Watson estimator. The old Batch Map and Batch Gradient Descent algorithms can be used for this instance. To conclude, local linear smoothing is automatic, has a simple intuitive interpretation and adapts well to our cause. In the next section, we show how this concept can be incorporated into our batch training algorithm.

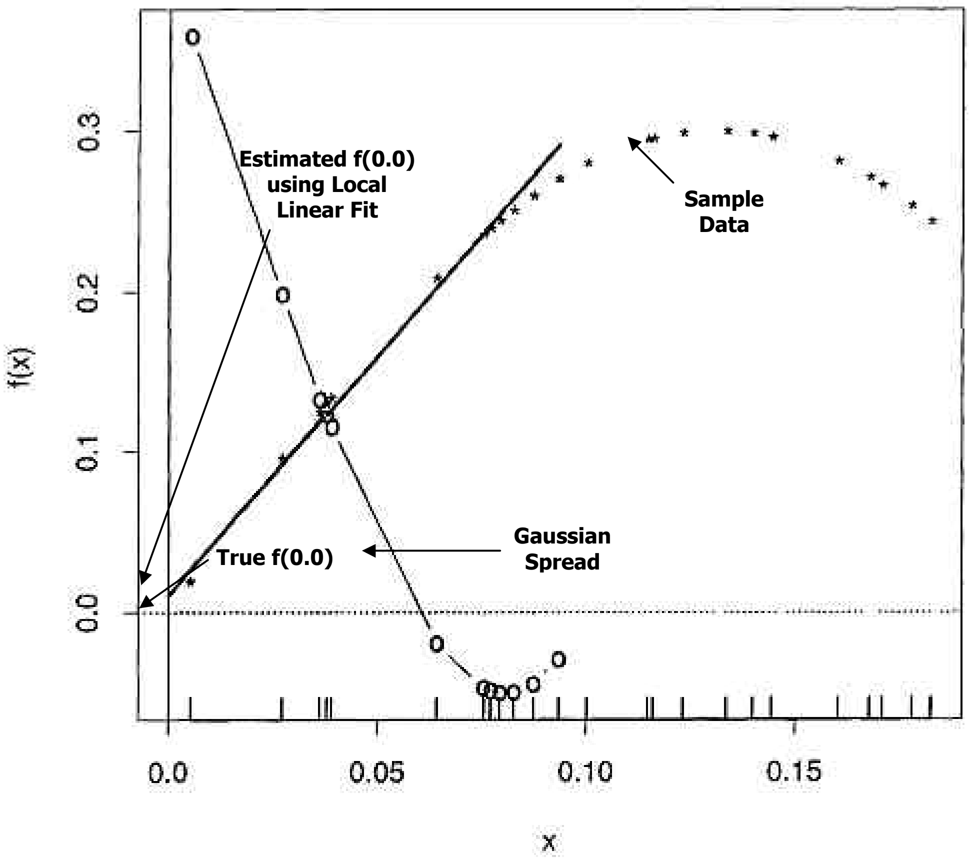


Fig. 13 Local linear smoothing, boundary region. x=0.0 is the boundary of the sample points distribution. We can observe that the bias has been considerably reduced, compared to that of Naradaya-Watson estimator.

4.6 Batch Training Algorithm

Batch Map with Local Linear Smoothing

1. For each neuron **r** in the 2-d lattice, collect a *list* $\mathcal{V}_r$ of all those training samples (**v**, **c**)

from the training set $\mathcal{T}$ whose **v** is "closest" to $\mathbf{w_r}$, using the distance measure in Step 3 of the sensorimotor control algorithm described in Section 4.2.

2. For each neuron **r** in the 2-d lattice, find the number of samples n_r and the mean $\mathbf{m_r}$ over the respective *list*.

$$\mathbf{m_r} = \frac{\sum_{\forall \mathbf{v} \in n_r} \mathbf{v}}{n_r} \tag{33}$$

3. For each neuron **s** in the 2-d lattice, determine its $\mathbf{w_s}$ by

$$\mathbf{w_s} = \frac{\sum_{\forall \mathbf{r}} \mathrm{k}(\mathbf{r}, \mathbf{s})\, \mathbf{m_r}}{\sum_{\forall \mathbf{r}} \mathrm{k}(\mathbf{r}, \mathbf{s})} \tag{34}$$

$$\mathrm{k}(\mathbf{r}, \mathbf{s}) = \mathrm{h}(\mathbf{r}, \mathbf{s})\, n_r\, (\, \mathrm{s}_2(\mathbf{r}, \mathbf{s}) - \quad \mathbf{r} - \mathbf{s} \quad \mathrm{s}_1(\mathbf{r}, \mathbf{s})\,) \tag{35}$$

$$\mathrm{s}_i(\mathbf{r}, \mathbf{s}) = \sum_{\forall \mathbf{r}} \mathrm{h}(\mathbf{r}, \mathbf{s})\, n_r \left\| \mathbf{r} - \mathbf{s} \right\|^{i} \tag{36}$$

where h(**r**, **s**) defines the Gaussian kernel weight of $\mathbf{r} - \mathbf{s}$ and it is of the form in equation (14). k(**r**, **s**) is in fact the deformed Gaussian, which is used to eliminate the boundary bias. If collective neighborhood learning is not desired, local linear smoothing cannot be performed. Since $\mathbf{r} - \mathbf{s} = 0$, the denominator of equation (34) turns zero, making it unsolvable. Then $\mathbf{w_s}$ is simply

$$\mathbf{w_s} = \mathbf{m_r} \tag{37}$$

This algorithm turns into K-means or Linde-Buzo-Gray (LBG) algorithm [64].

4. Iterate all the steps until further change in $\mathbf{w_s}$ is considered minimal.

Batch Gradient Descent with Local Linear Smoothing

1. For each neuron **r** in the 2-d lattice, initialize D$\mathbf{M_r}$' = 0.

2. Do Step 2 of the Batch Map with Local Linear Smoothing.

3. For each neuron **r** in the 2-d lattice, use each training sample (**v**, **c**) in the respective *list* to do

$$\mathrm{D}\mathbf{M}_{\mathbf{r}}' \leftarrow \mathrm{D}\mathbf{M}_{\mathbf{r}}' + r\ (\ \mathbf{c} - \mathbf{M}_{\mathbf{r}}\ \mathbf{v}\ )\ \mathbf{v} \tag{38}$$

   where $r \in [0,1]$ is the learning rate.

4. For each neuron **r** in the 2-d lattice, find the number of training samples n_r collected over the respective *list*.

5. For each neuron **s** in the 2-d lattice, determine its $\mathrm{D}\mathbf{M}_{\mathbf{s}}'$ by

$$\text{Ä}\mathbf{M}_{\mathbf{s}}' = \frac{\sum_{\forall \mathbf{r}} \mathrm{k}'(\mathbf{r},\mathbf{s})\ \text{Ä}\mathbf{M}_{\mathbf{r}}'}{\sum_{\forall \mathbf{r}} \mathrm{k}'(\mathbf{r},\mathbf{s})} \tag{39}$$

   where k'(**r**, **s**) is of the form in equation (35). If collective neighborhood learning is not desired, local linear smoothing cannot be performed (Since **r** - **s** = 0, the denominator of equation (39) turns zero, making it unsolvable). Then equation (39) is simply omitted and this algorithm becomes a normal Batch Gradient Descent [18].

6. For the same neuron **s**, determine the improved value of $\mathbf{M}_{\mathbf{s}}$.

$$\mathrm{D}\mathbf{M}_{\mathbf{s}} = j\ \mathrm{D}\mathbf{M}_{\mathbf{s}}' \tag{40}$$

   where j defines the learning rate of the form in equation (25) that decreases exponentially with *t*.

7. Iterate all the steps until further change in $\mathbf{M}_{\mathbf{s}}$ is considered minimal.

Chapter 5

SIMULATION RESULTS AND ANALYSIS

The simulation results in this chapter compare the performance of our online unsupervised learning algorithm, the inclusion of the pre-learning phase and our batch training algorithm using local linear smoothing. We also test the real Khepera robot based on one time step.

5.1 Simulation Overview

5.1.1 Performance Metrics

Mean Positioning Error (MPE)

This measures the mean accuracy of the robot self-positioning to designated target locations. The robot must come to a stop, that is $\mathbf{c} = [0, 0]^T$, to be considered to have reached the designated target location. It is mathematically defined as

$$MPE = \frac{\sum_{\forall i}^{n} d_i}{n} \tag{41}$$

where d_i defines the self-positioning error at one designated target location and n defines the number of designated target locations to be reached.

Mean Steps per unit Distance (MSD)

Measures the mean rate (number of time steps per meter) at which the robot moves to reach all designated target locations. Recall that each time step is defined to be t seconds.

5.1.2 Experimental Setup

1. Webots 2.0 Simulator, as described in Section 3.2, is used as an implementation tool for our algorithms.
2. This simulator already models 10% white noise in the motor actuators and light/IR sensors.
3. 6 different test cases have been conducted under the same conditions listed in this section. Their results are displayed from sections 5.2 to 5.6 respectively.
4. Each test case is run 5 times. In each run, the number of time steps t_{max} is 100000. The time t for each step is 1024 ms. During these 100000 time steps, the robot will be presented with randomly generated target locations for learning. In fact, target reaching movements can be observed during this learning stage.
5. During each interval of 10000 time steps, the robot will be tested with 50 target locations for its performance. These target locations are initially randomly generated to lie in a 5 by 5 meter square plane. These positions are then fixed for the performance testing in all the 10 intervals. Whenever the robot is tested with these 50 target locations, it will be manually placed at the center of this square plane first.
6. The whole environment is obstacle-free.

5.2 Online Learning : Single Neuron Update

This test case does not incorporate collective neighborhood learning; each neuron adapts its own weights.

Parameters Initialization

Parameters in Sensorimotor Control Algorithm (Section 4.2)
$\mathcal{K} = 15$, $b = 1.0$, $g = 0.1$, $c_{min} = -20$, $c_{max} = 20$.

Parameters in Unsupervised Online Learning Algorithm (Section 4.3)
$\mathcal{N} = 225$, $X = 15$, $Y = 15$, $D_{max} = 0.02$, $a_{min} = -3.0$, $a_{max} = 3.0$, $h(0) = 1.0$, $h(t_{max}) = 0.2$, $h'(0) = 1.0$, $h'(t_{max}) = 0.2$, $r = 0.08$.

Algorithm Performance

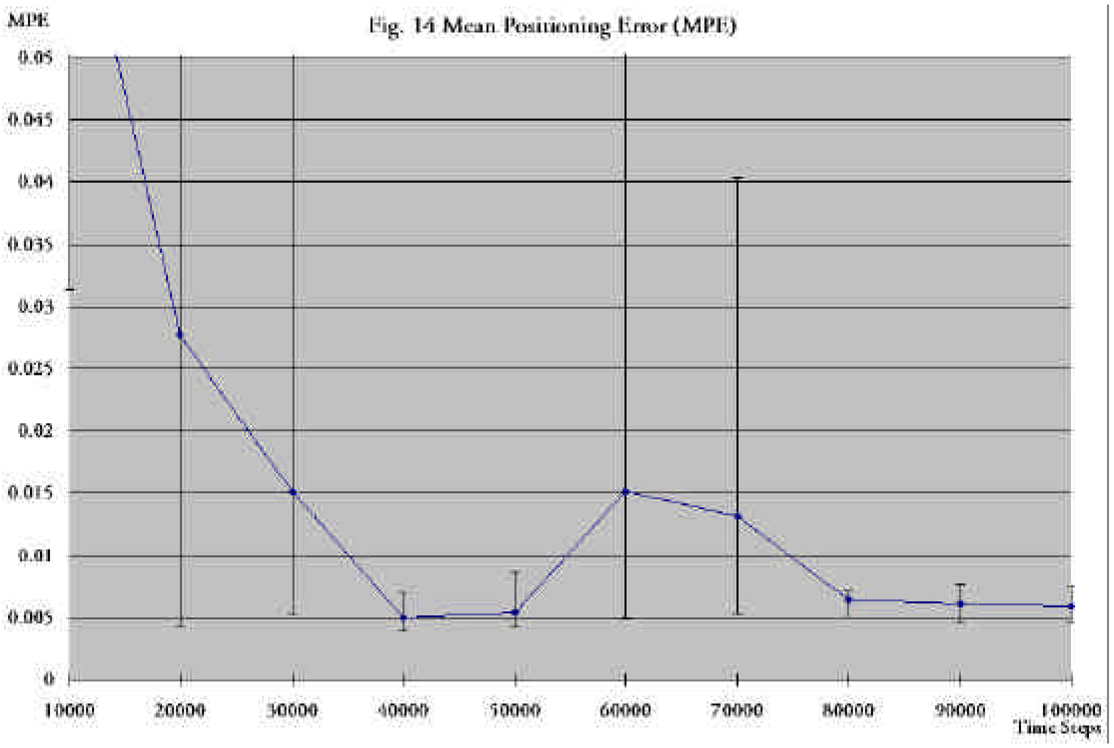
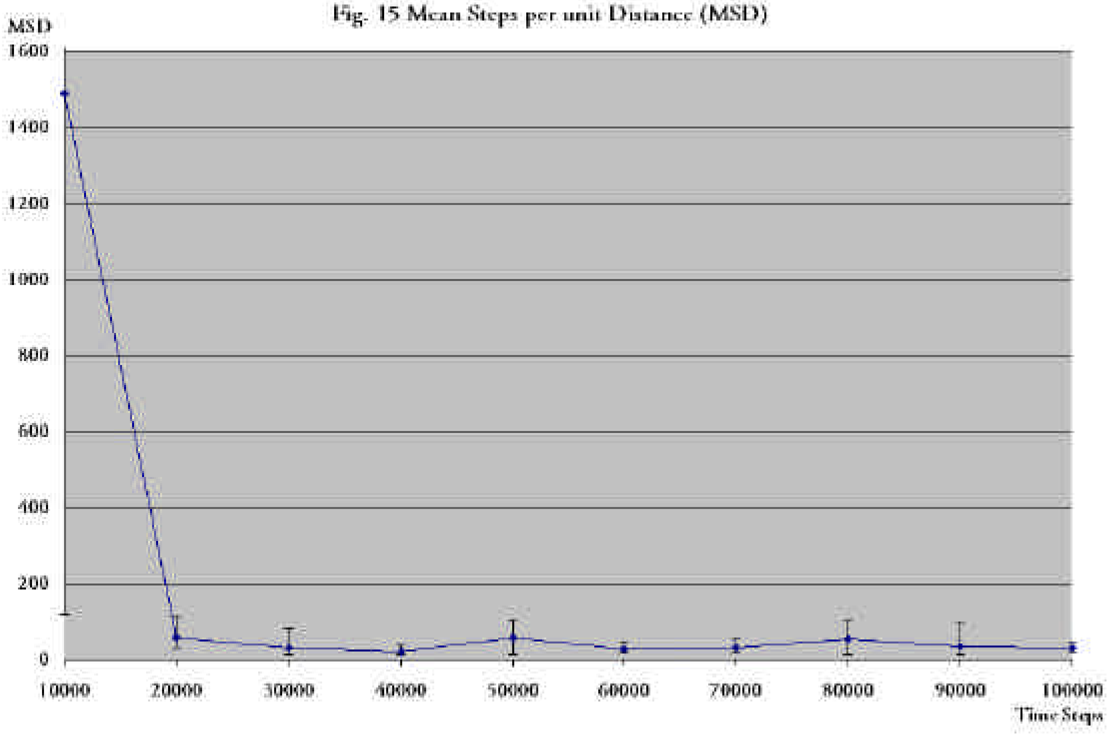


Fig. 15 Mean Steps per unit Distance (MSD)



Observations and Analysis

We can observe from Fig. 14 that the network stabilizes after 80000 time steps at about 0.006 meters for MPE and 35 time steps per meter for MSD. Let us now compare these results with online learning: neighborhood update in the next section.

5.3 Online Learning : Neighborhood Update

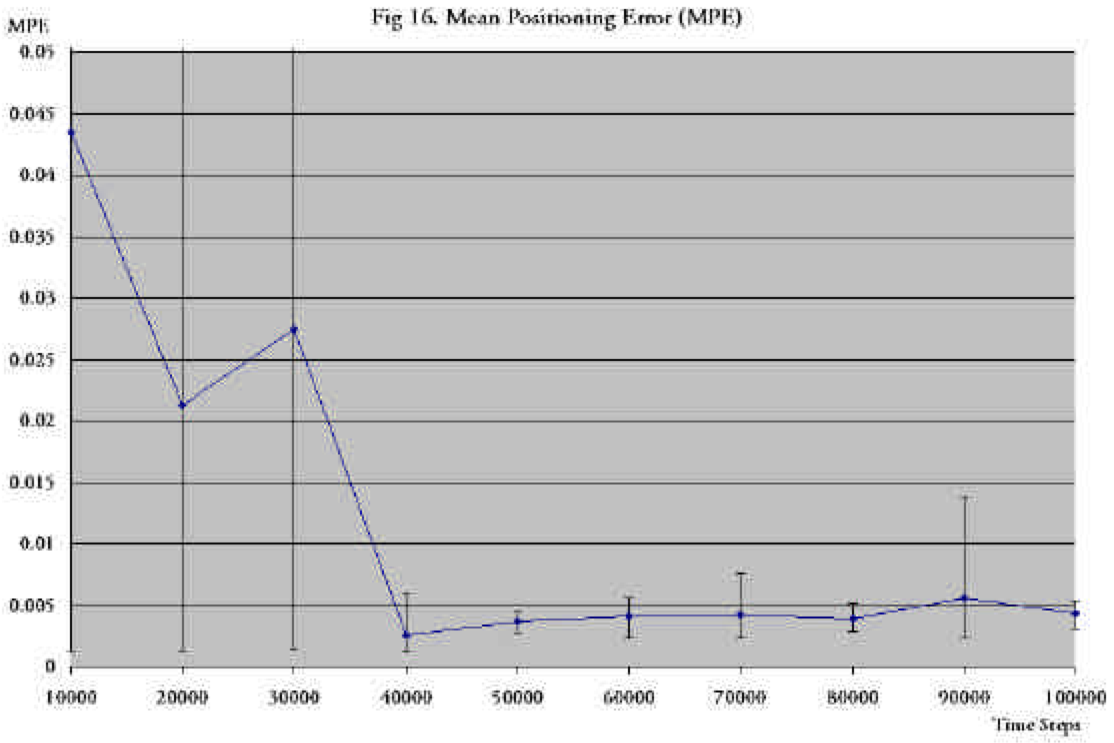
This test case incorporates collective learning.

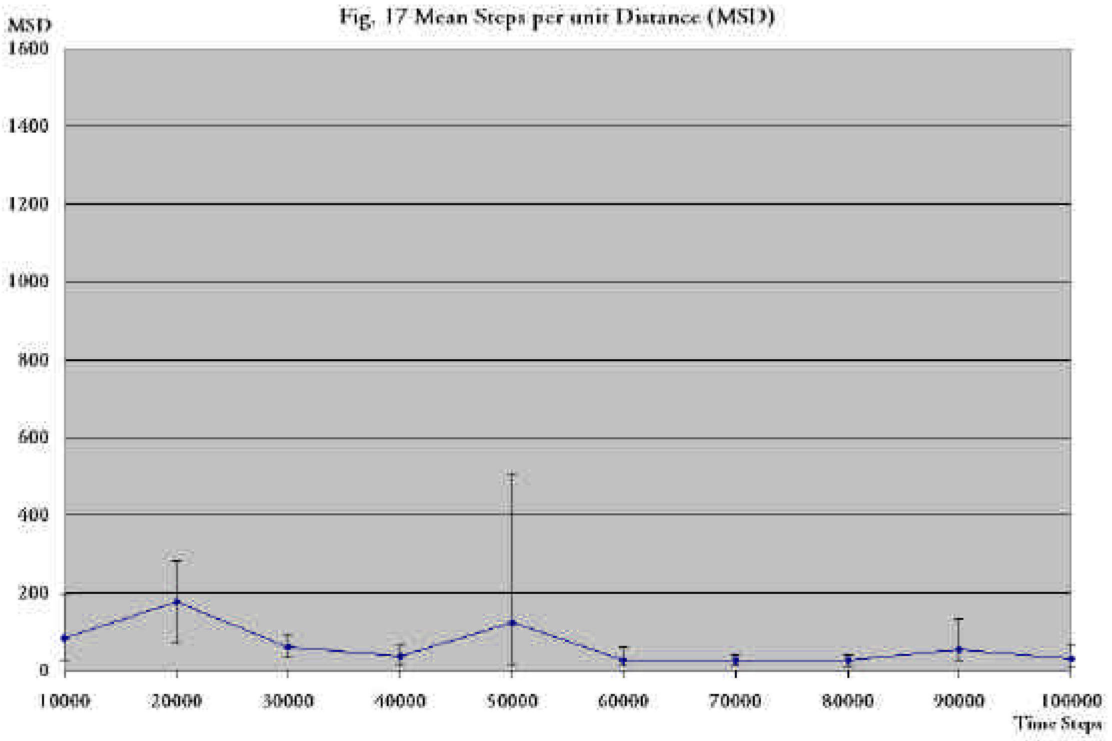
Parameters Initialization

Parameters in Sensorimotor Control Algorithm (Section 4.2)
$\mathcal{K} = 15$, $b = 1.0$, $g = 0.1$, $c_{min} = -20$, $c_{max} = 20$.

Parameters in Unsupervised Online Learning Algorithm (Section 4.3)
$\mathcal{N} = 225$, $X = 15$, $Y = 15$, $D_{max} = 0.02$, $a_{min} = -3.0$, $a_{max} = 3.0$, $h(0) = 1.0$, $h(t_{max}) = 0.2$, $s(0) = 1.2$, $s(t_{max}) = 0.1345$, $h'(0) = 1.0$, $h'(t_{max}) = 0.2$, $s'(0) = 0.6$, $s'(t_{max}) = 0.38$, $r = 0.08$.

Algorithm Performance





Observations and Analysis

We can see from Fig. 16 that there is a tremendous improvement in the MPE and the rate of convergence over online learning: single neuron update in the previous section. The network stabilizes after 40000 time steps at about 0.004 meters for MPE and 35 time steps per meter for MSD.

Presence of Boundary Bias Phenomenon

As mentioned in Section 4.5, our online unsupervised learning algorithm will experience a boundary bias problem. The phenomenon caused can be easily discerned by observing the movements of the weight sensory vectors $\mathbf{w_r}$ of the neurons in the local perceptual space. Fig. 18 to Fig. 24 attempt to show the $\mathbf{w_r}$ of these neurons in the x, y plane expressed in meters.

Fig. 18 Local Perceptual Space (x,y plane) at 0 time steps

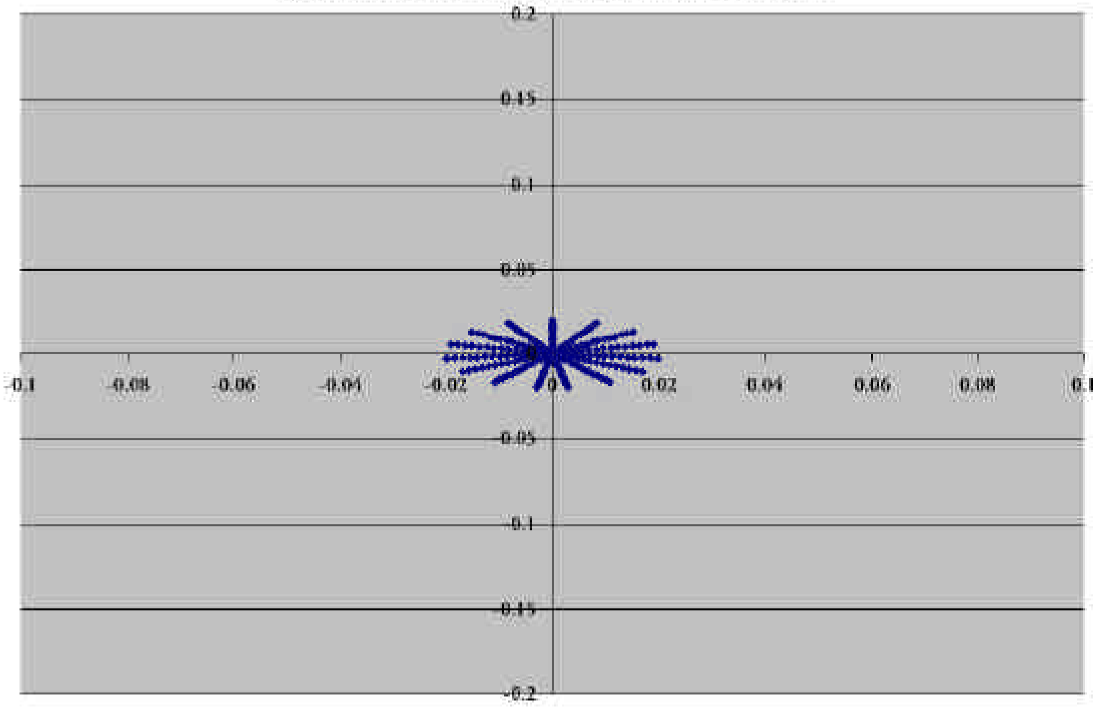


Fig. 19 Local Perceptual Space (x,y plane) at 6000 time steps

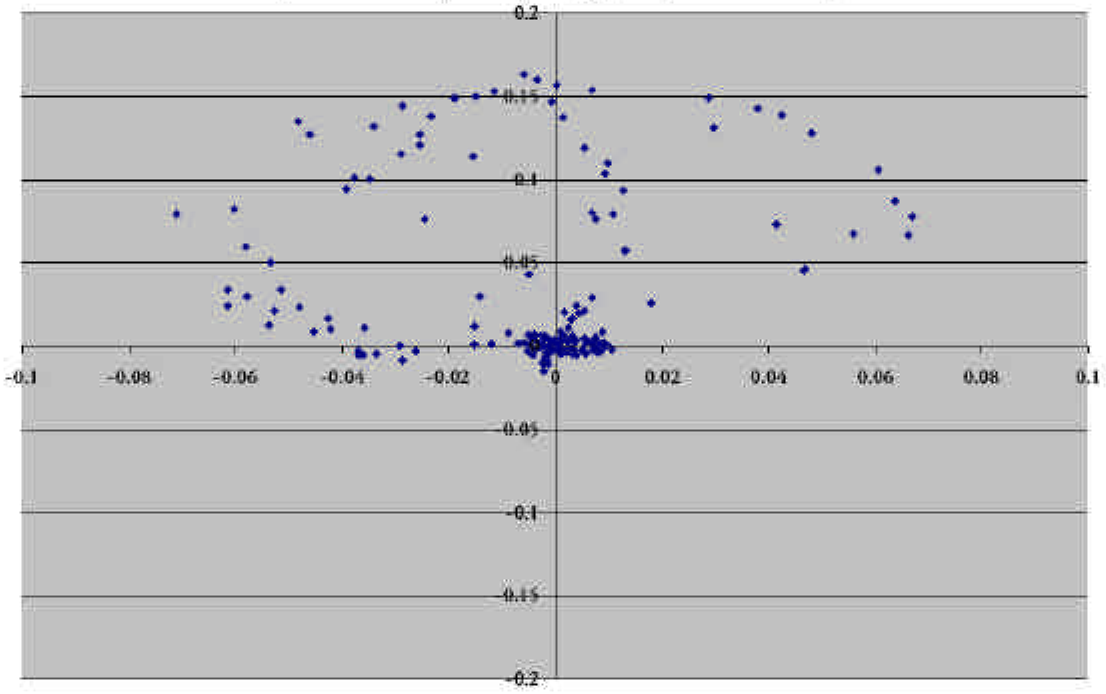


Fig. 20 Local Perceptual Space (x,y plane) at 14000 time steps

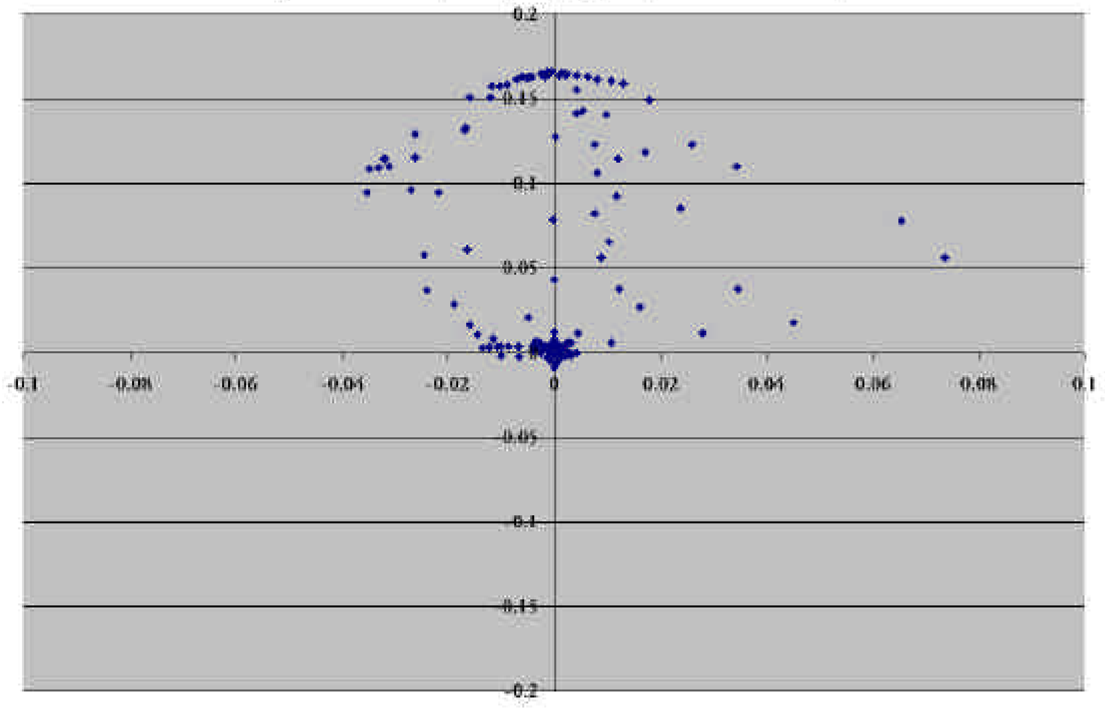


Fig. 21 Local Perceptual Space (x,y plane) at 26000 time steps

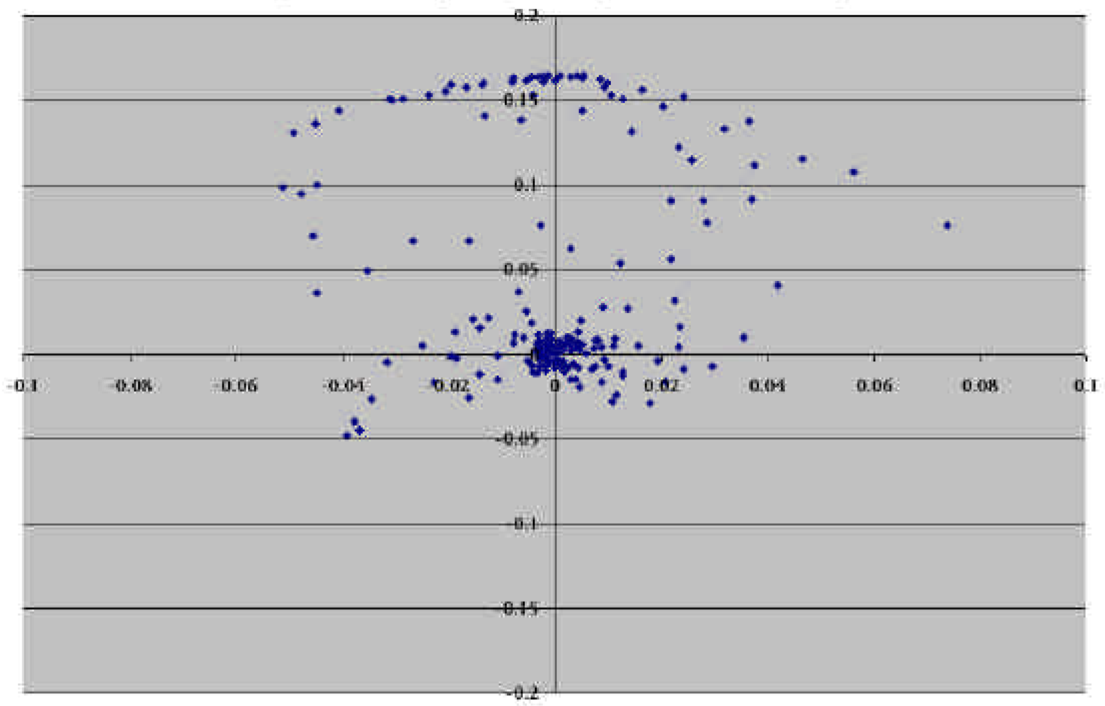


Fig. 22 Local Perceptual Space (x,y plane) at 34000 time steps

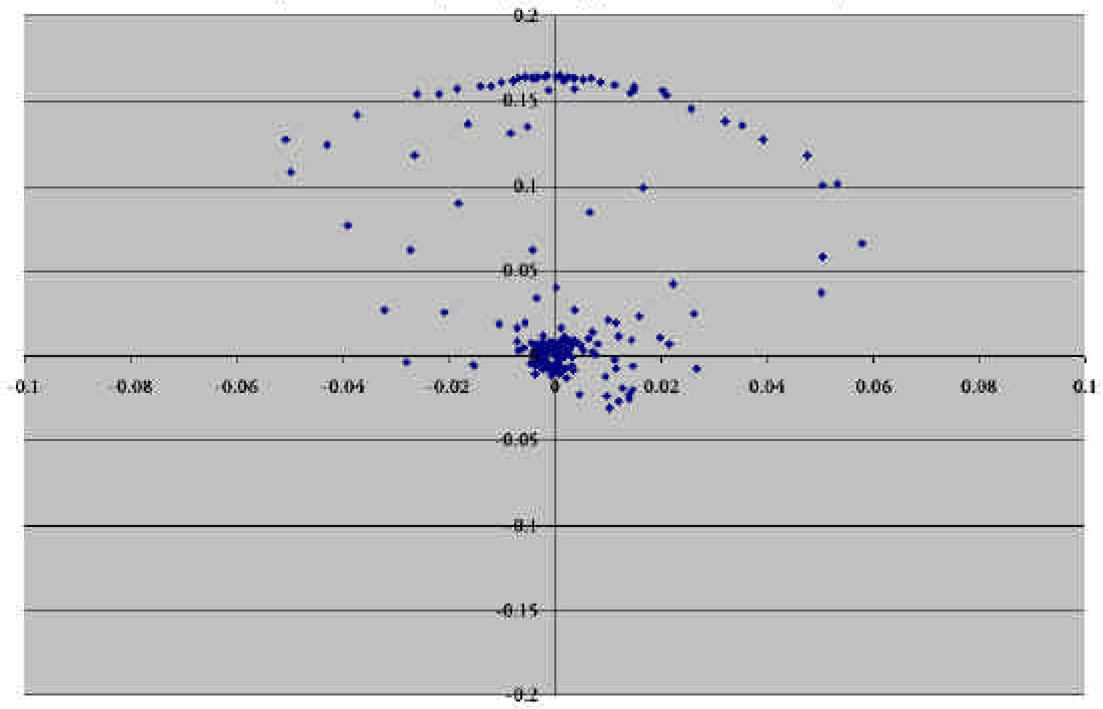


Fig. 23 Local Perceptual Space (x,y plane) at 50000 time steps

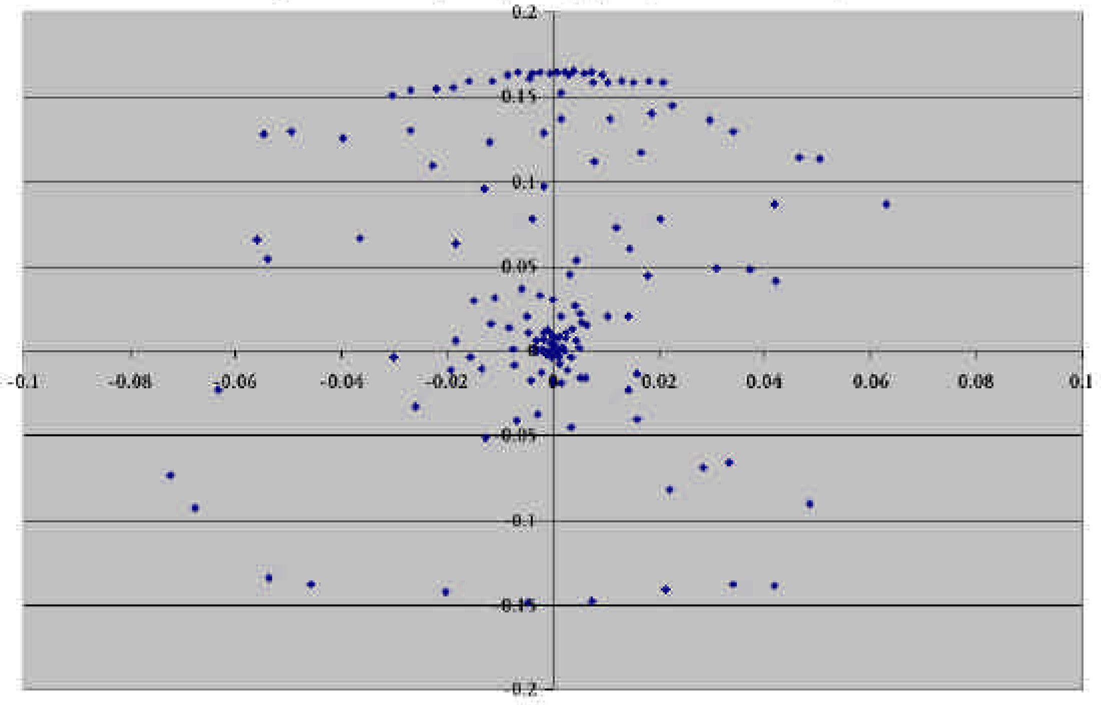
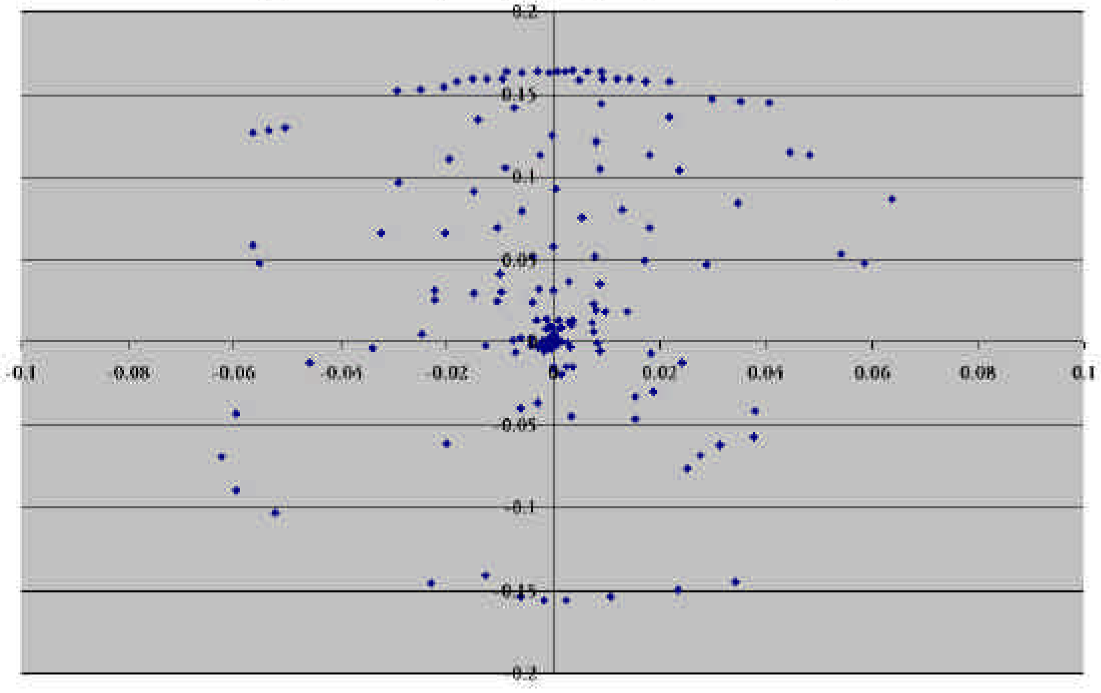


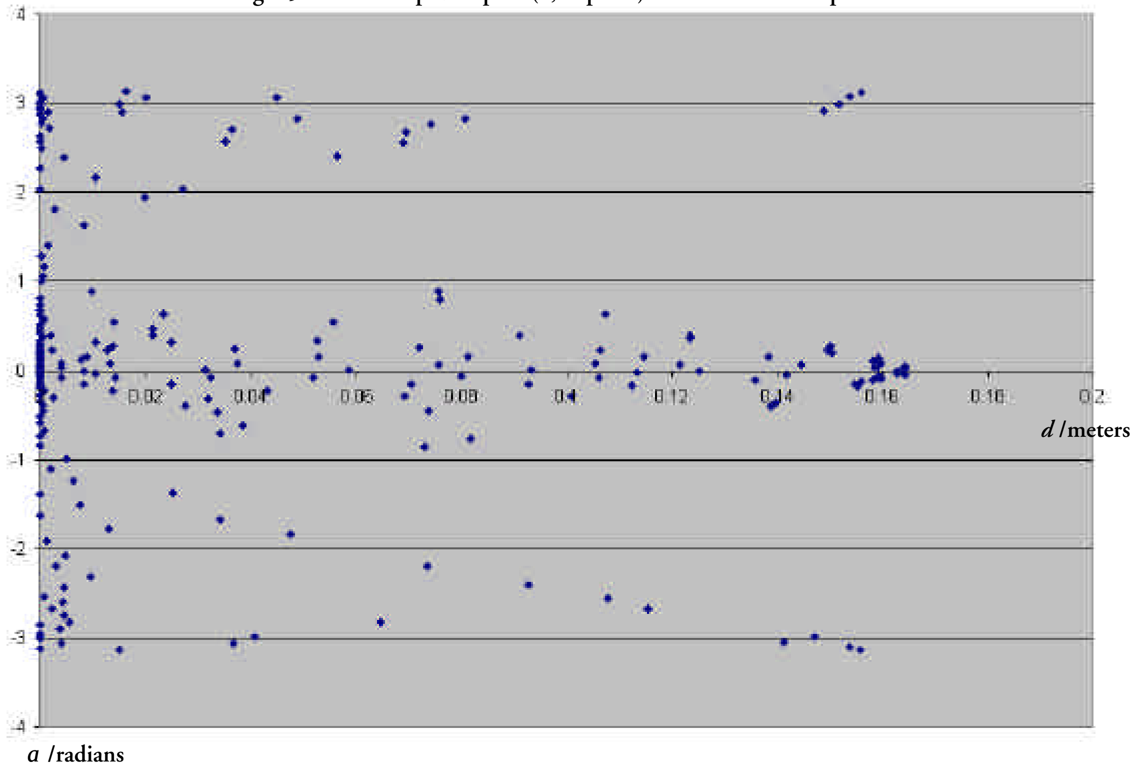
Fig. 24 Local Perceptual Space (x,y plane) at 100000 time steps



The weight sensory vectors of the neurons initially expand out to fill the local perceptual space at 6000 time steps and then collapse towards the center at 14000 time steps. By

26000 time steps, they expanded out again and at 34000 time steps, there is only a slight collapse. By this time, a major collapse is prevented due to a large enough sample set at the boundary and the reduction of the Gaussian bandwidth and learning rate. Finally, the network stabilizes its local perceptual space at about 50000 time steps. Fig. 25 shows the stabilized state of the local perceptual space at 100000 time steps in *d*, *a* plane. We can observe from Fig. 24 and Fig. 25 that there is a major concentration of weight sensory vectors at the origin where *d* is approximately 0. This is necessary for fine positioning to reduce the MPE to a minimum. Majority of the neurons are clustered in the region where *a* lies between –1 radian and 1 radian. This is required for a low MSD because the robot can move in a more or less straight path to the target, which implies shortest distance and delay to target.

Fig. 25 Local Perceptual Space (*d*, *a* plane) at 100000 time steps



5.4 Pre-learning : Single Neuron Update

This test case is similar to the test case in Section 5.2 with an initial pre-learning or motor babbling phase. Since it is single neuron update, local linear smoothing is not employed. K-means [64] and normal Batch Gradient Descent [18] are used for batch processing during the motor babbling phase. Refer to Section 4.6 for more details.

Parameters Initialization

Parameters in Motor Babbling (Section 4.4)
$\mathcal{T}$= 1681

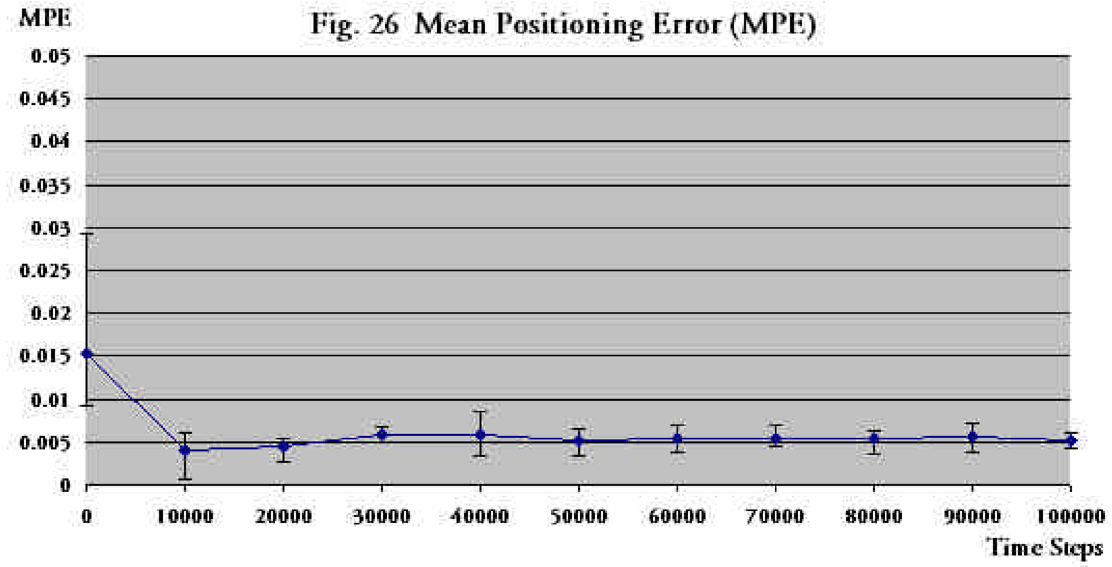
Parameters in Sensorimotor Control Algorithm (Section 4.2)
$\mathcal{K}$= 15, b = 1.0, g = 0.1, c_{min} = -20, c_{max} = 20.

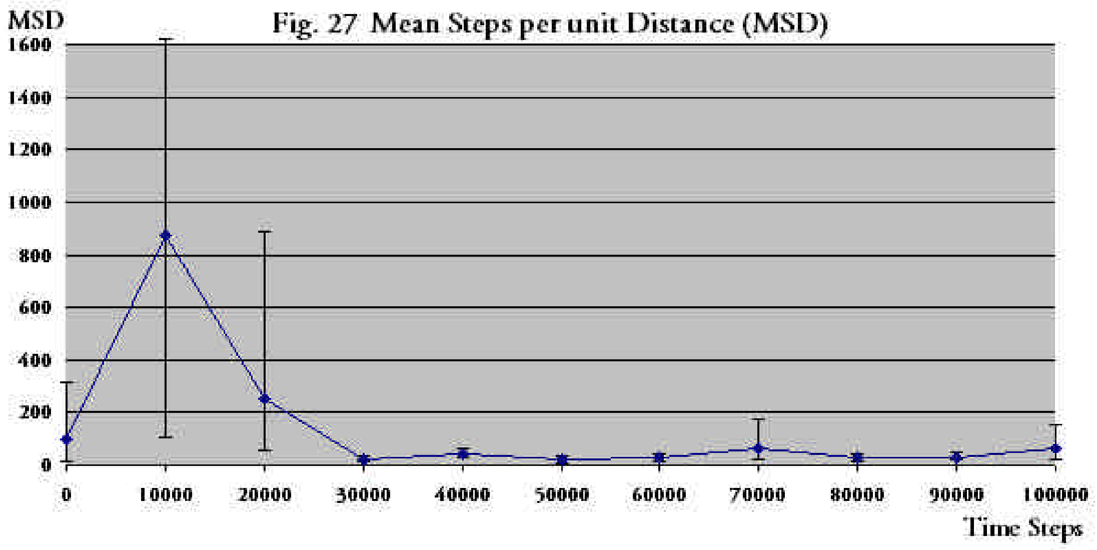
Parameters in Unsupervised Online Learning Algorithm (Section 4.3)
$\mathcal{N}$= 225, X = 15, Y = 15, $h(0)$ = 1.0, $h(t_{max})$ = 0.2, $h'(0)$ = 1.0, $h'(t_{max})$ = 0.2, r = 0.08.

Parameters in K-Means Algorithm (Section 4.6)
Iterations = 50.

Parameters in Batch Gradient Descent Algorithm (Section 4.6)
Iterations = 250, r = 0.08, j = 0.08.

Algorithm Performance





Observations and Analysis

From Fig. 26 and Fig. 27, the motor babbling phase at time step 0 creates a MPE of about 0.015 meters and 100 time steps per meter for MSD. The network stabilizes after 30000 time steps at about 0.0055 meters for MPE and 35 time steps per meter for MSD, which is better than online learning: single neuron update in terms of convergence rate. Thus, we can conclude that motor babbling can improve the convergence rate tremendously but is not necessary for successful online learning through single neuron update.

5.5 Pre-learning : Neighborhood Update

This test case is similar to the test case in Section 5.3 with an initial pre-learning or motor babbling phase. Local Linear Smoothing is employed in Batch Map and Batch Gradient Descent during the motor babbling phase.

Parameters Initialization

Parameters in Motor Babbling (Section 4.4)
$\mathcal{T}$ = 1681
Parameters in Sensorimotor Control Algorithm (Section 4.2)
$\mathcal{K}$ = 15, b = 1.0, g = 0.1, c_{min} = -20, c_{max} = 20.

Parameters in Unsupervised Online Learning Algorithm (Section 4.3)
$\mathcal{N} = 225$, $X = 15$, $Y = 15$, $h(0) = 1.0$, $h(t_{max}) = 0.2$, $s(0) = 1.2$, $s(t_{max}) = 0.1345$, $h'(0) = 1.0$, $h'(t_{max}) = 0.2$, $s'(0) = 0.6$, $s'(t_{max}) = 0.38$, $r = 0.08$.

Parameters in Batch Map Algorithm for Pre-Learning with Local Linear Smoothing (Section 4.6)
Iterations = 50, $s(0) = 0.8$, $s(50) = 0.1345$.

Parameters in Batch Gradient Descent Algorithm for Pre-Learning with Local Linear Smoothing (Section 4.6)
Iterations = 250, $r = 0.08$, $j = 0.07$, $s'(0) = 0.6$, $s'(250) = 0.38$.

Algorithm Performance

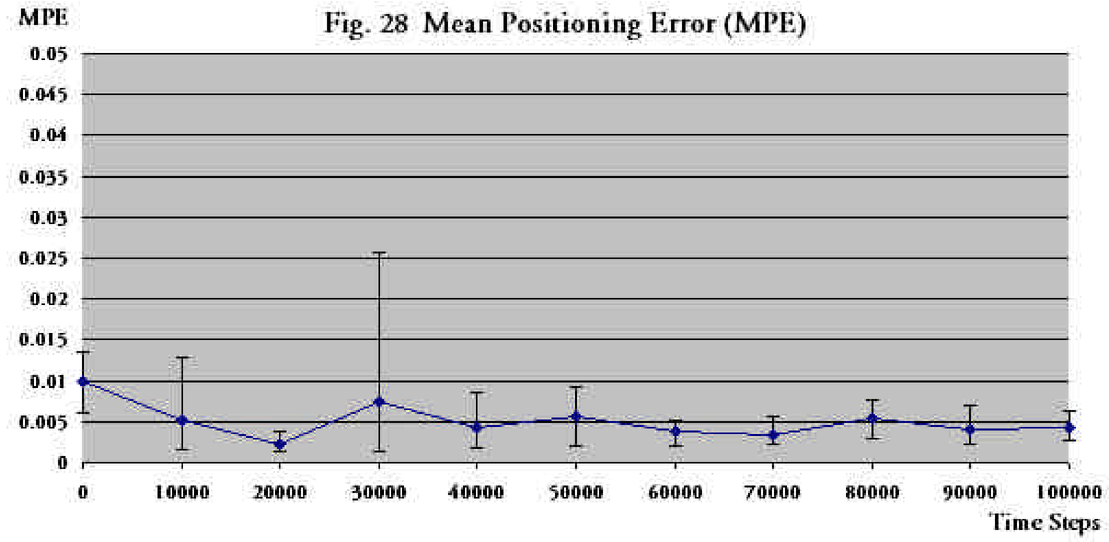


Fig. 28 Mean Positioning Error (MPE)

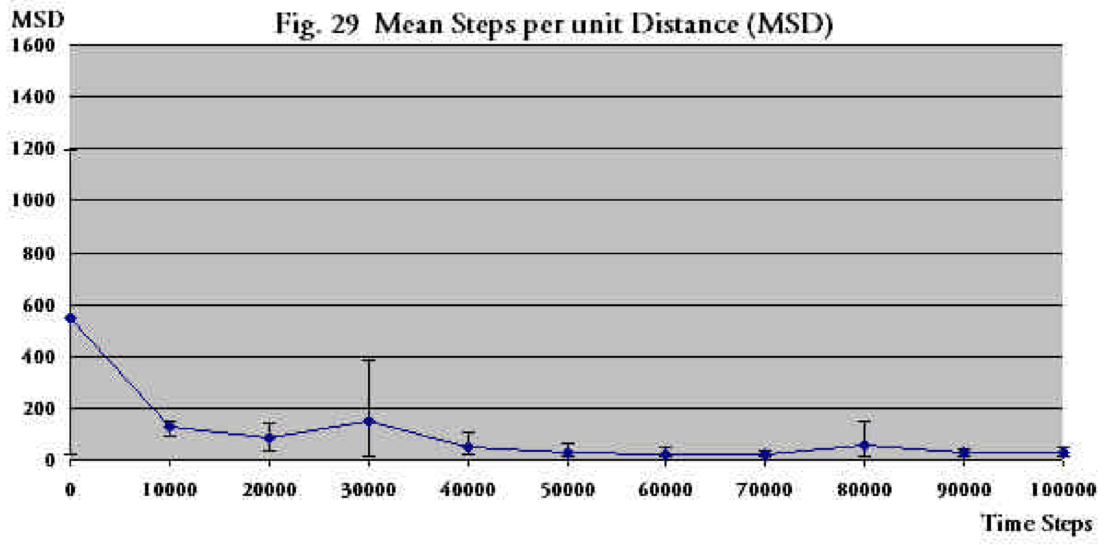


Fig. 29 Mean Steps per unit Distance (MSD)

Observations and Analysis

We can see from Fig. 28 and 29 that the network stabilizes after 40000 time steps at about 0.004 meters for MPE and 35 time steps per meter for MSD. There is no improvement over online learning: neighborhood update. Thus, motor babbling is considered redundant if neighborhood update is in play.

5.6 Batch Learning : Neighborhood Update

This test case uses Batch Map and Batch Gradient Descent with Local Linear Smoothing. 500 training samples are collected for each batch training process.

Parameters Initialization

Parameters in Batch Map Algorithm with Local Linear Smoothing (Section 4.6)

$\mathcal{N}$= 225, X = 15, Y = 15, D_{max} = 0.02, a_{min} = -3.0, a_{max} = 3.0, $\mathcal{K}$= 15, b = 1.0, g = 0.1, c_{min} = -20, c_{max} = 20, $\mathcal{T}$= 500, Iterations = 200, $s(0)$ = 0.7, $s(t_{max})$ = 0.1345.

Parameters in Batch Gradient Descent Algorithm with Local Linear Smoothing (Section 4.6)

$\mathcal{T}$= 500, Iterations = 200, r = 0.08, j = 0.001, $s'(0)$ = 0.6, $s'(t_{max})$ = 0.38.

Algorithm Performance

Fig. 30 Mean Positioning Error (MPE)

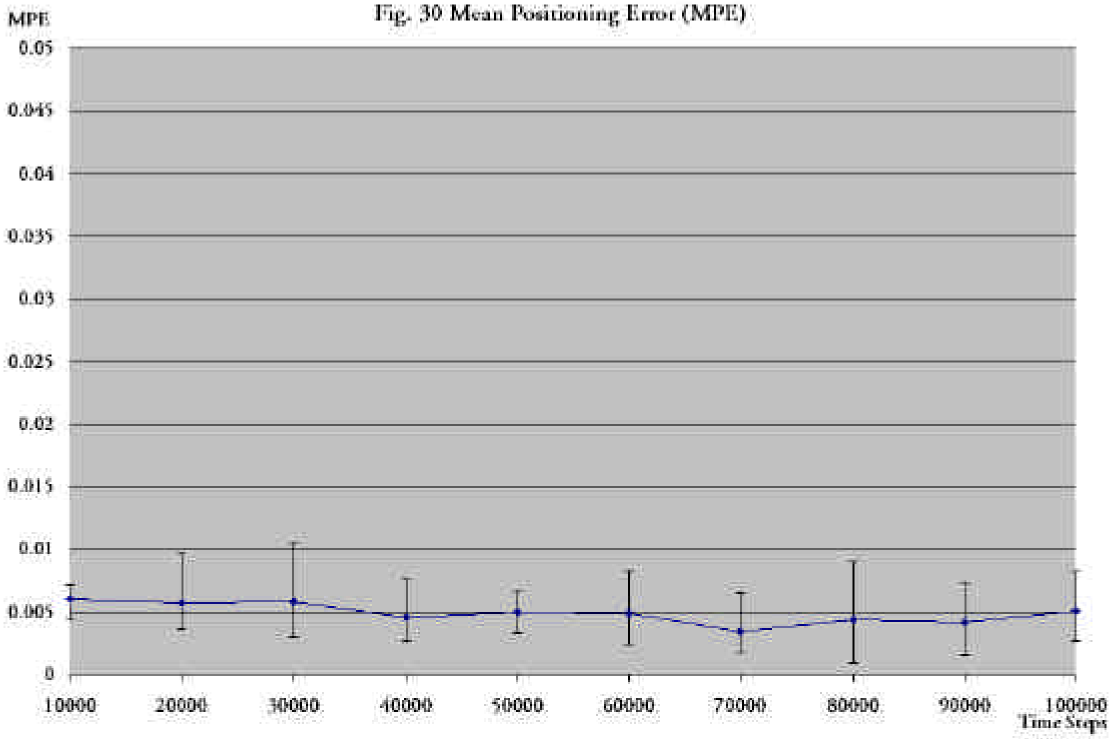
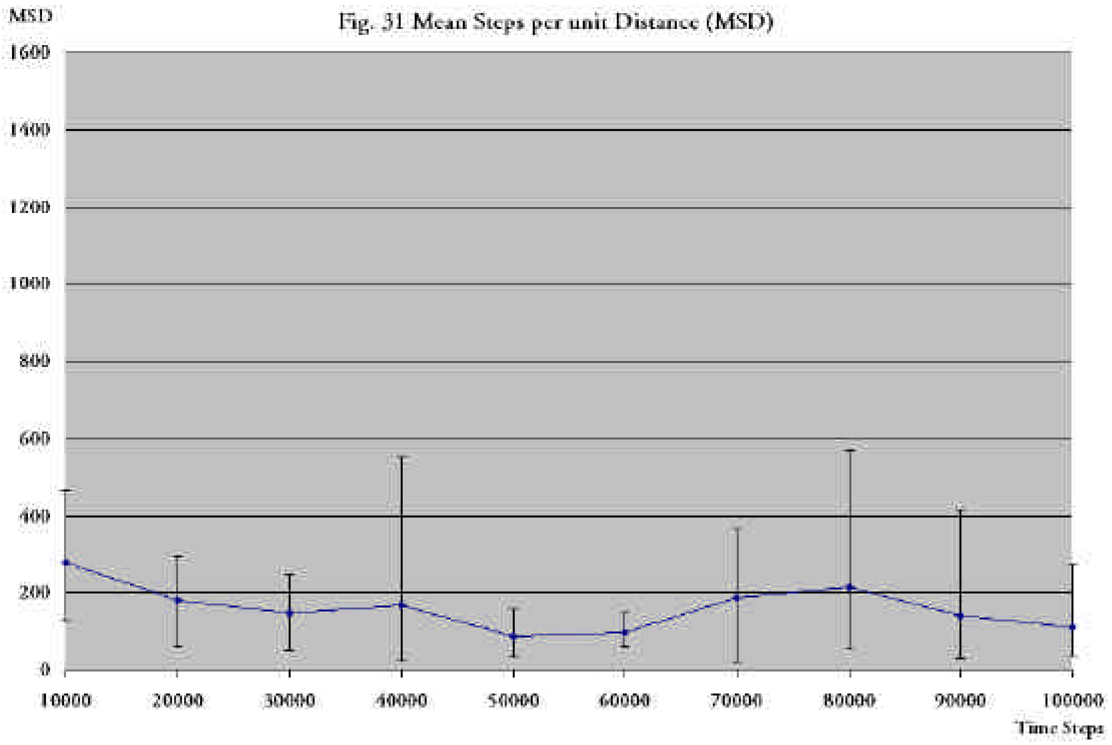


Fig. 31 Mean Steps per unit Distance (MSD)



Observations and Analysis

We can see from Fig. 30 and Fig. 31 that the network stabilizes extremely fast at 10000 time steps with MPE of 0.005 meters and 160 time steps per meter for MSD.

Absence of Boundary Bias Phenomenon

As mentioned in Section 4.5, Local Linear Smoothing can eliminate the boundary bias. Thus, the phenomenon does not exist, as proven in Fig. 32 to Fig. 37. At time step 0, the neurons' weight sensory vectors are initialized to that in Fig. 18.

Fig. 32 Local Perceptual Space (x,y plane) at 6000 time steps

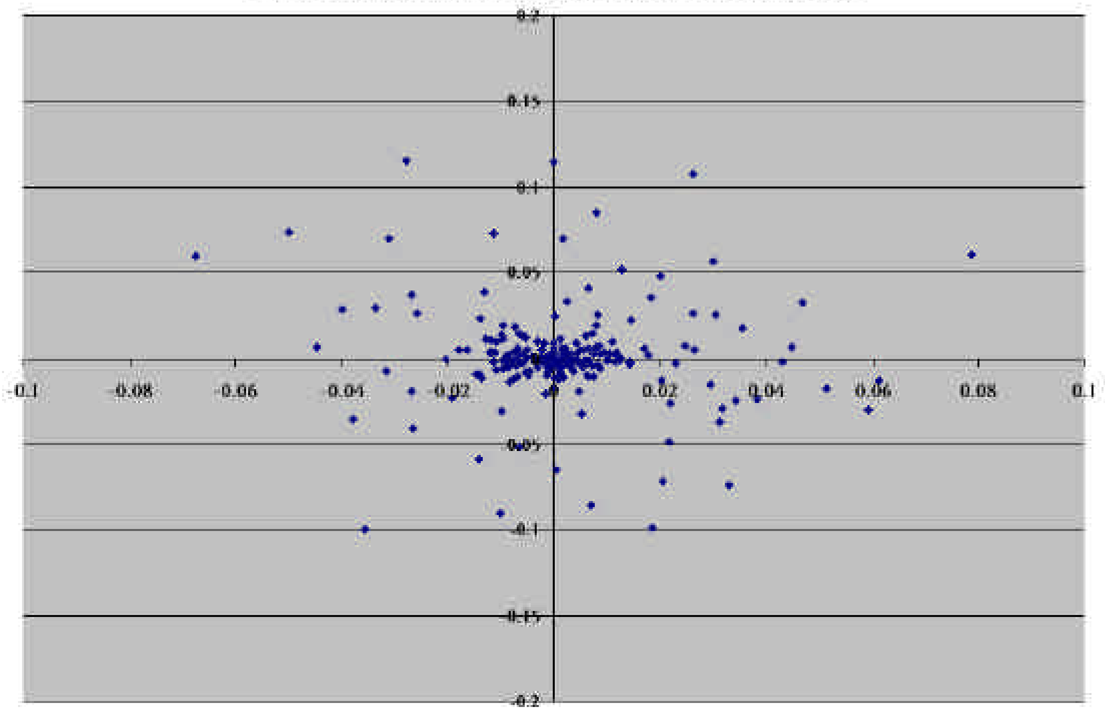


Fig.33 Local Perceptual Space (x,y plane) at 14000 time steps

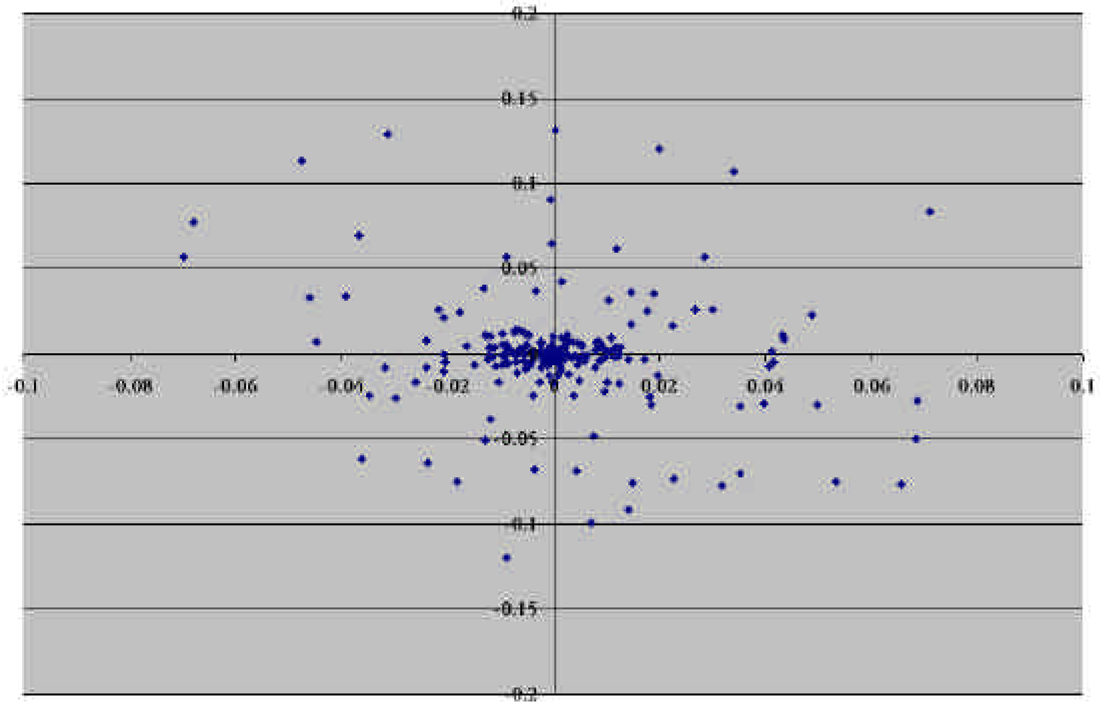


Fig. 34 Local Perceptual Space (x,y plane) at 26000 time steps

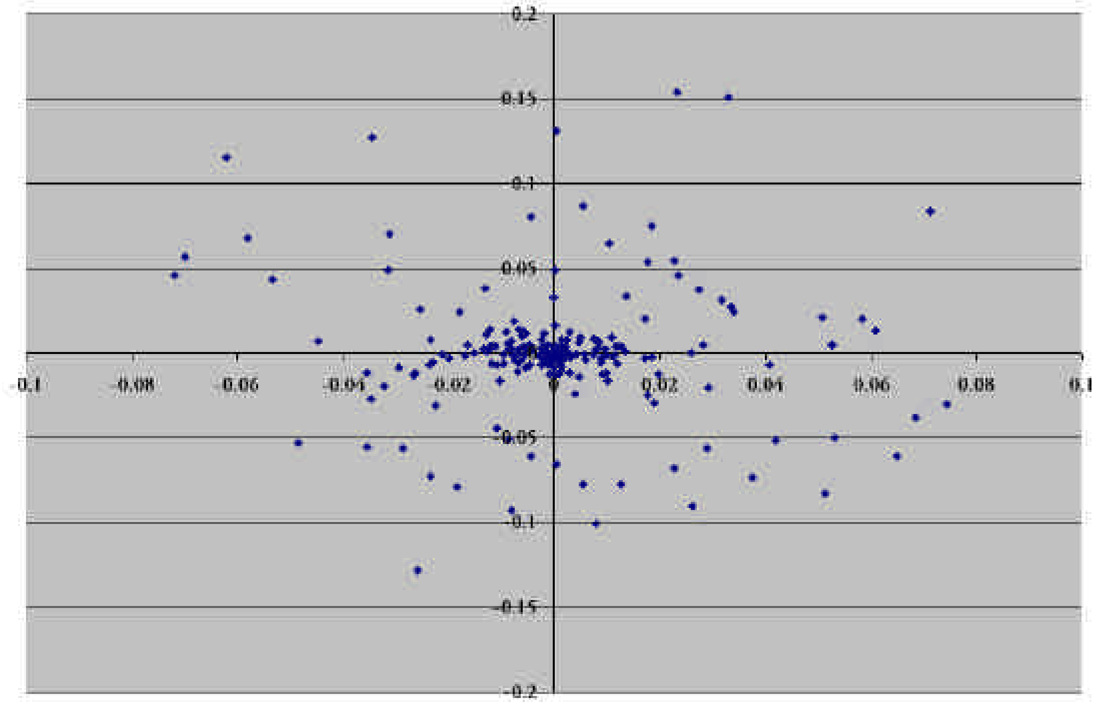


Fig. 35 Local Perceptual Space (x,y plane) at 34000 time steps

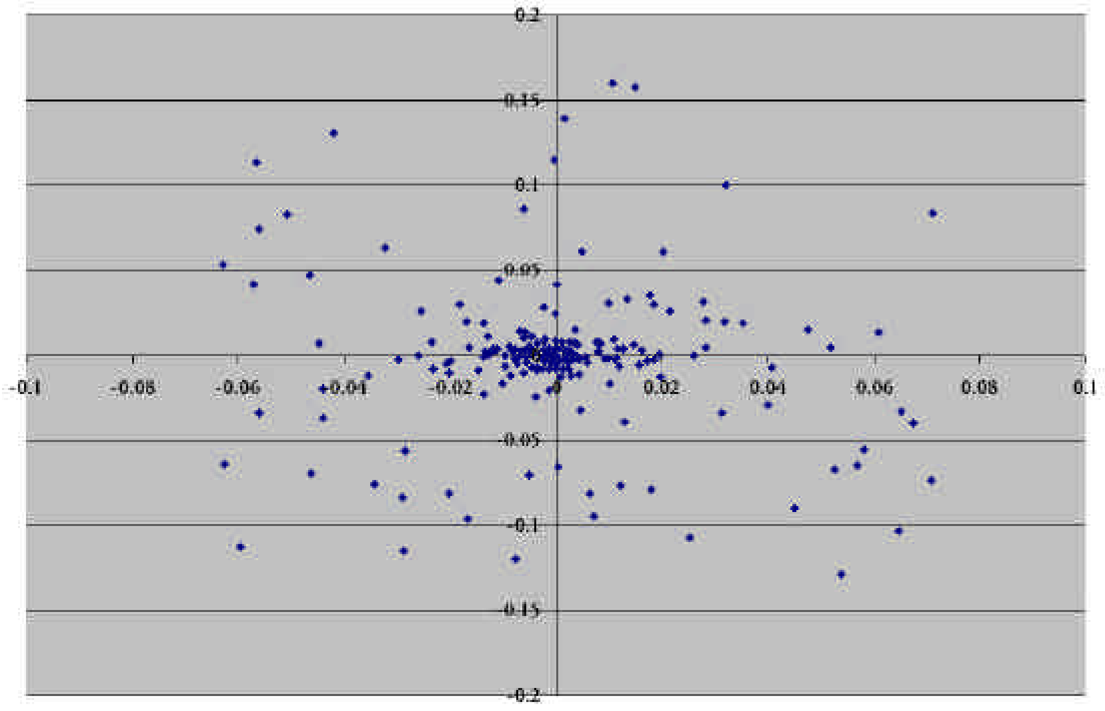


Fig. 36 Local Perceptual Space (x,y plane) at 50000 time steps

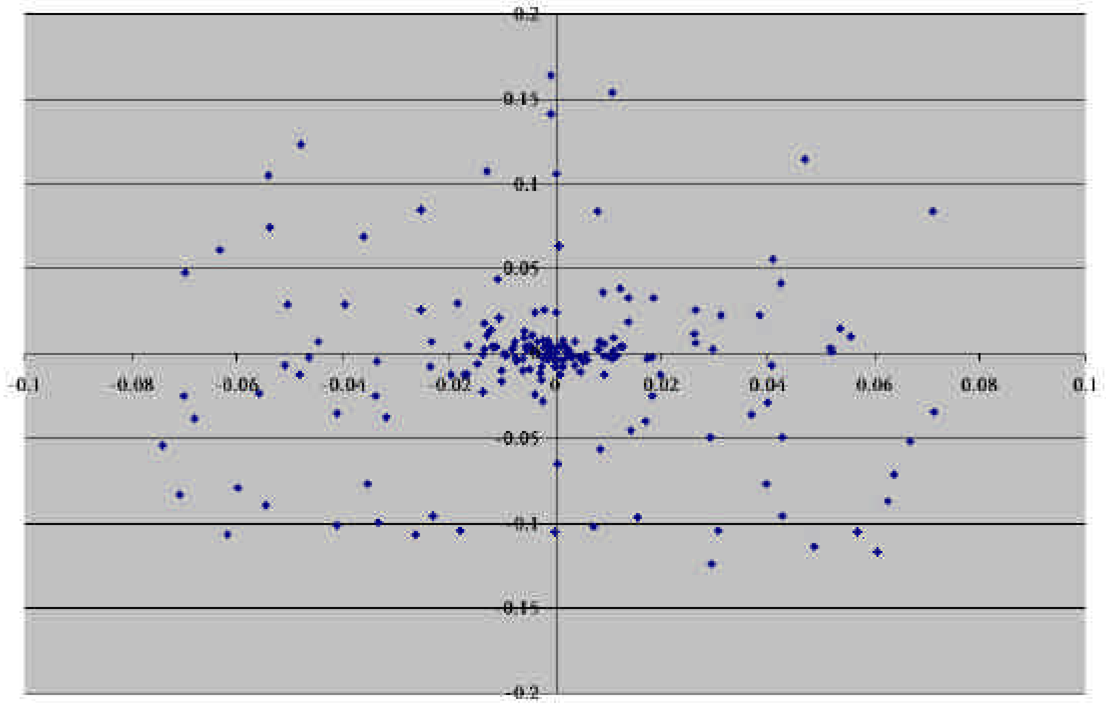
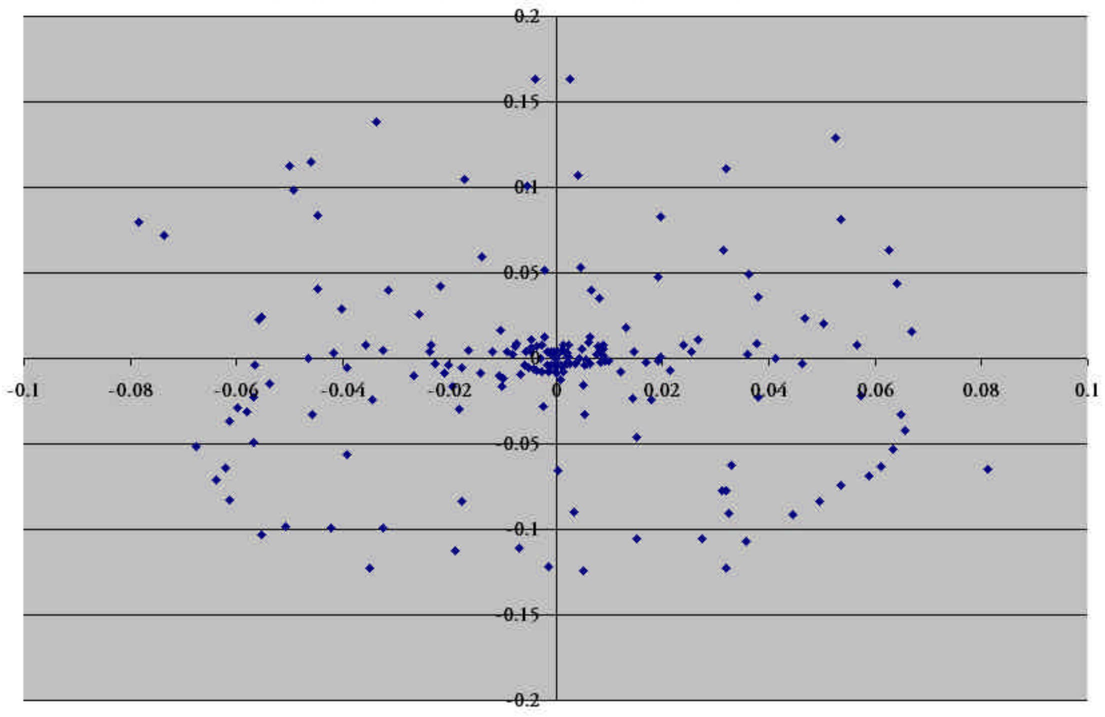
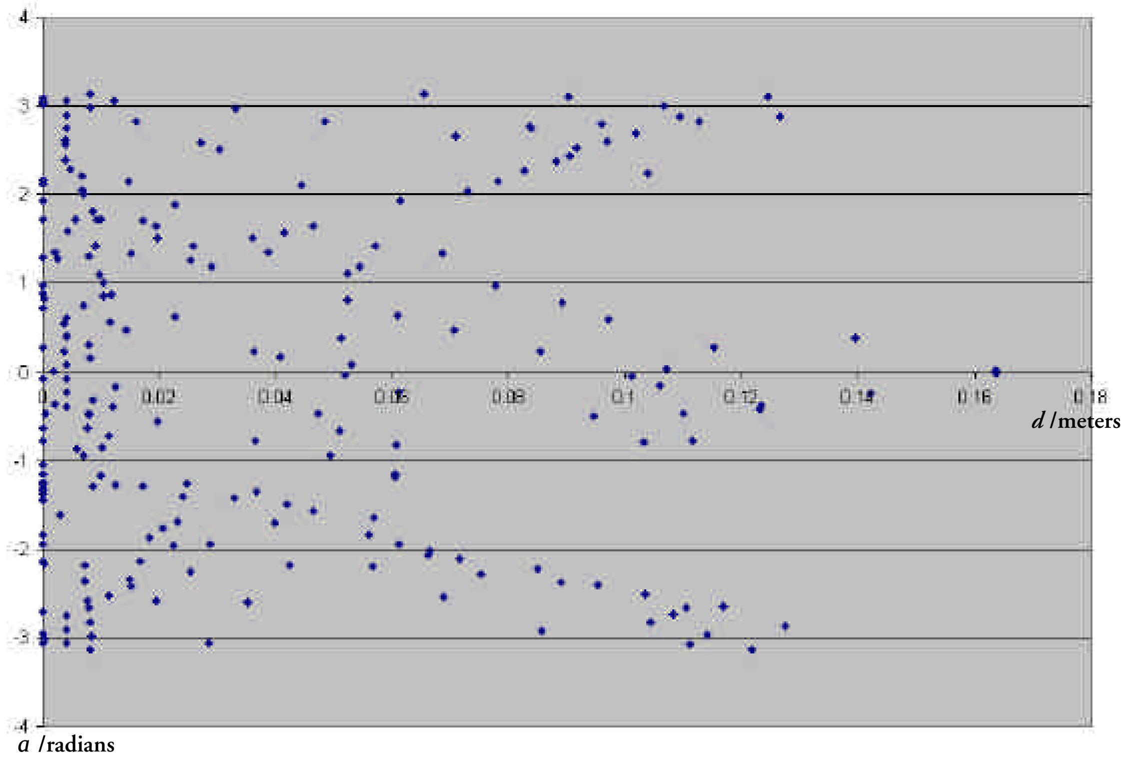


Fig. 37 Local Perceptual Space (x,y plane) at 100000 time steps



By 6000 time steps, the weight sensory vectors have already expanded out by quite a fair amount. Hence, we can observe a faster rate of convergence for local linear smoothing. Fig. 38 shows the stabilized state of the local perceptual space at 100000 time steps in *d*, ***a*** plane. We can observe from Fig. 37 and Fig. 38 that there is a major concentration of weight sensory vectors at the origin where *d* is approximately 0. This is necessary for fine positioning to reduce the MPE to a minimum.

Fig. 38 Local Perceptual Space (d, a plane) at 100000 time steps



The rest of the neurons are well spread in the local perceptual space, instead of clustering in the front region $a \in (-1, 1)$, which is the case for online learning: neighborhood update. This accounts for a higher MSD since no emphasis is placed on learning to move in a straight path. Why is this so? The robot has its first batch training session after 500 time steps. These initial 500 training samples emphasize on stabilizing at the randomly generated target location, which implies fine motion. By subsequent reinforcements, the robot continues with this emphasis. There are also evidences, during the learning process in our simulation runs, showing that the robot reaches much less random target locations, as compared to that of online learning: neighborhood update. We want to test our claim. Therefore, we increase the batch number $\mathcal{T}$ to 2000 and run some simulations. The MSD results for $\mathcal{T}$= 2000 are presented in Fig. 39.

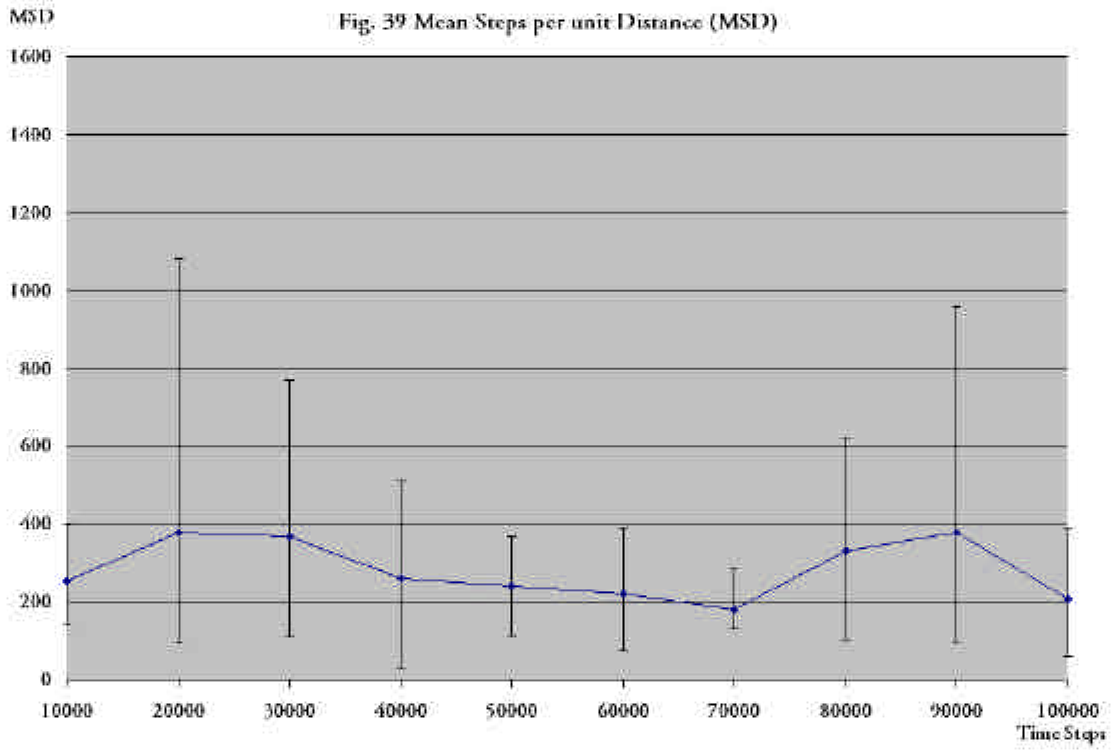


Fig. 39 shows an average MSD of 280 time steps per meter, which is 120 time steps more than that of $\mathcal{T} = 500$. This means that more time is used to stabilize at a target location. Thus, our claim is substantiated.

5.7 Model Performance on Khepera Robot

As mentioned in Section 3.3, without a vision system, we can only test the real Khepera robot on the basis of 1 time step of 1024 ms. To do so, we employ the remote control setup described in Section 3.1.3. The khep_core package [61] is used to interface with the robot; it is a package of C++ classes that allows one to communicate simply and effectively with a Khepera robot via the serial port of a PC. The weights of the trained SOM in Section 5.3 Online Learning: Neighborhood Update and in Section 5.6 Batch Learning: Neighborhood

Update are used for testing. Fig. 40 tabulates the results of the positioning errors at various possible locations in 1 time step.

d / meters	*a* / radians	ONLINE LEARNING (Section 5.3) Self-Positioning Error	BATCH LEARNING (Section 5.6) Self-Positioning Error
0.100	0.000	0.0102	0.0045
0.127	0.197	0.0122	0.0210
0.127	-0.197	0.0054	0.0177
0.100	3.142	0.0143	0.0231
MPE (*d* >= 0.1)		**0.0105**	**0.0166**
0.050	0.000	0.0000	0.0000
0.071	0.785	0.0248	0.0112
0.071	0.785	0.0175	0.0161
0.050	1.571	0.0120	0.0367
0.050	-1.571	0.0135	0.0120
0.050	3.142	0.0211	0.0000
0.071	2.356	0.0305	0.0269
0.071	-2.356	0.0242	0.0134
MPE (*d* <= 0.71)		**0.0180**	**0.0145**

Fig. 40 Positioning errors of Khepera Robot at various locations in 1 time step.

The mean positioning errors for *d* >= 0.1 and *d* <= 0.71 for the online and batch learning are as expected. From Section 5.3 Fig. 24, we can observe a dense clustering for long distances (*d* >= 0.1) on the robot trained by online learning. Hence, we would expect a better accuracy. From Section 5.6 Fig. 37, we witness a sparse clustering for long distances (*d* >= 0.1) on the robot trained by batch training. This results in a greater error. But for shorter distances (*d* <= 0.71), this same robot is more accurate because it emphasizes on fine motion during training as explained in Section 5.6. The robot with online learning has a slightly higher error due to less emphasis in fine motion. Note that if these values can be further refined if iterative sensory feedback is available.

For the simulations described in this chapter, no obstacles are assumed, not even walls. Thus, an infinite workspace may be required to train the robot. This is not practical. If we want the robot to start learning in an enclosed, obstacle-ridden environment, can it be done? The next chapter addresses this issue.

Chapter 6

LEARNING IN AN OBSTACLE-RIDDEN ENVIRONMENT

6.1 A Biological Approach : Escape Tactics of a Fly

Have you ever tried to chase a fly with your bare hands? You may have wondered how such a small insect can develop such efficient escape tactics. Where can it process the visual information that is coming to it from its eyes, if it barely has a brain? The answer lies in the visual system itself. Animals of all sizes and types show a remarkable capability to obtain very complicated behaviors with simple systems. What is their secret? Can it give us some insight on developing some autonomous complex behaviors for mobile robots?

The secret lies in sensor adaptation and short sensorimotor loops. By sensor adaptation we understand that the animal sensors are perfectly adapted for a given task. For instance, the fly uses the light detectors in its panoramic eyes to detect motion in the scene. It becomes blind if there is no motion. Thus, it needs to move constantly to perceive its environment. Short sensorimotor loops are necessary to have a real-time response, something critical in a dynamic environment. To achieve this, animals have very short connections between sensors and actuators, which implies that most of the visual information is not processed at the highest levels of the nervous system, which is the brain. The biological solution is to process the information on the retinas themselves, therefore sending to the next layer only the relevant information.

Now suppose that a newborn fly has an innate ability to avoid obstacles using its visual sensors and it is desperate to learn homing through its smell sensors so that it can home in on potential food sources and steer clear of moving bodies such as our hands. This situation is analogous to a mobile robot, with obstacle avoidance capabilities, trying to autonomously learn its homing behavior in an obstacle-ridden environment. Using the above-mentioned concepts of sensor adaptation and short sensorimotor loops, we develop a behavior coordination mechanism, which fuses homing and obstacle avoidance, to enable this.

6.2 Obstacle Avoidance Behavior

Before we introduce the behavior coordination mechanism, we need a simple, pre-defined obstacle avoidance behavior, which exhibits high real-time reactivity. Braitenberg type-3C vehicle [56] matches our description. Fig. 41 illustrates how it works.

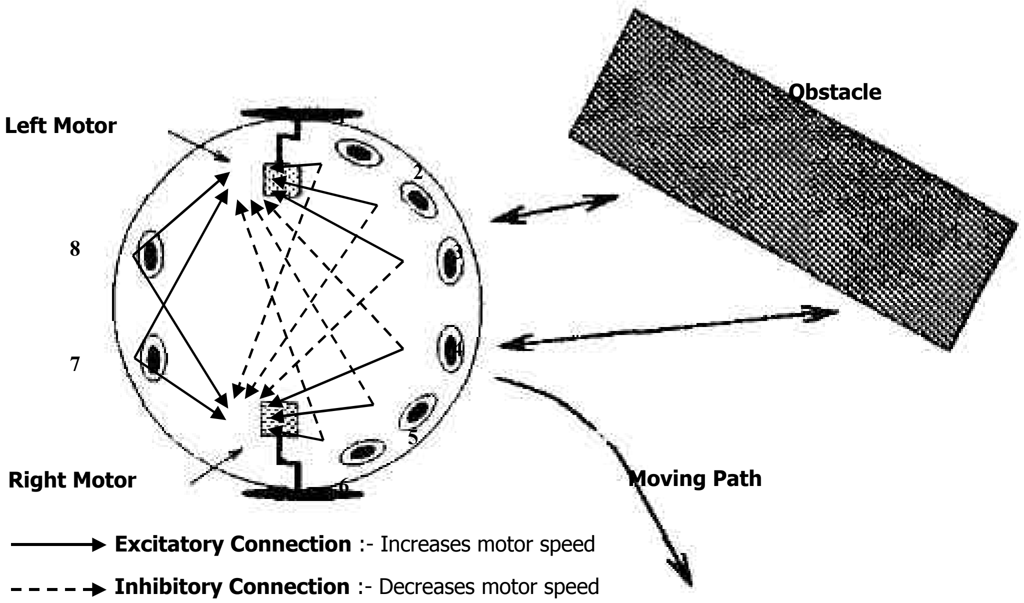

Obstacle
Left Motor
2
3
8
7
4
5
6
Right Motor
Moving Path
Excitatory Connection :- Increases motor speed
Inhibitory Connection :- Decreases motor speed

Fig. 41 Braitenberg type-3C vehicle. The front IR/light sensors have uncrossed excitatory connections and crossed inhibitory connections to the two motors while the back sensors have excitatory connections to both motors. The proximity of any sensors to an obstacle is reflected in its signal strength. For example, sensors 2 and 3 have extremely high signal strengths due to a nearby obstacle. The change in motor speed of each wheel is dependent on the signal strength of each sensor and its corresponding connection type and weight. For example, excitatory connections of sensors 2 and 3 drive the left wheel forward by increasing the left motor speed and their inhibitory connections drive the right wheel backward by decreasing the right motor speed. The robot therefore turns to the right.

We can observe from Fig. 41 that the eight IR proximity/light sensors surrounding the Khepera robot serve as the artificial compound eye of the fly to avoid both static and dynamic obstacles in its path.

6.3 A Behavior Coordination Mechanism : Command Fusion

To enable learning in an obstacle-ridden environment, a mechanism known as command fusion is implemented to coordinate the homing and obstacle avoidance behaviors. Command fusion is concerned with "how to combine results from different behaviors into one command to be sent to the robot's actuators" [47]. This technique proceeds in three steps [57]:

1. **Action Recommendations** :- A module is implemented to generate recommended actions according to some behavioral criteria such as homing and obstacle avoidance.
2. **Behavior Aggregation** :- The actions recommended by behaviors are combined according to some rule such as summation.
3. **Action Selection** :- An appropriate action is selected based on the combined recommendations such as the output of summation in step 2.

Fig. 42 shows the command fusion of the homing and obstacle avoidance behaviors in a mobile robot to imitate a fly.

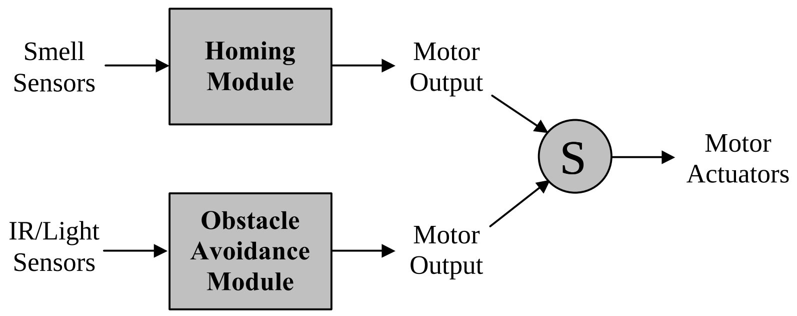


Fig. 42 Command fusion of homing and obstacle avoidance behaviors.

The smell and IR/light sensor readings are always collected simultaneously at a fixed time interval to produce a resultant motor output to move the robot wheels. Recall from Section 6.1 that a short sensorimotor loop is desired in order to have quick real-time response to the environment. Thus, the time step size t in the homing module (otherwise known as sensorimotor controller in Chapter 4) is reduced to a rate where the incremental movement space of the robot in one time step coincides with that of the IR/light sensory space. In fact, this change is critical to the success of the autonomous learning mechanism in Section 4.3. Every training sample fed into the learning mechanism comprises of a motor output vector and a corresponding observed motion vector of the robot in time t. In Fig. 42, the resultant motor output to the actuators from the summation is the one that produces the observed motion instead of the motor output from the homing module. If the incremental movement space in one time step exceeds that of the IR/light sensory space, the non-overlapped space may contain an obstacle to obstruct the actual motion path generated by motor output vector. This causes a foreshortening in the observed motion and creates an erroneous training sample. Fig. 43 gives a pictorial illustration of this problem. This command fusion mechanism is implemented and tested in the simulator to validate its performance.

6.4 Simulation Results and Analysis

The experimental setup is the same as that in Section 5.1.2. New constraints are added; 4 walls, 7 static obstacles (pillars and spheres) and 2 moving obstacles (Khepera robots) are placed in the enclosed environment together with our Khepera. These are shown in Fig. 44.

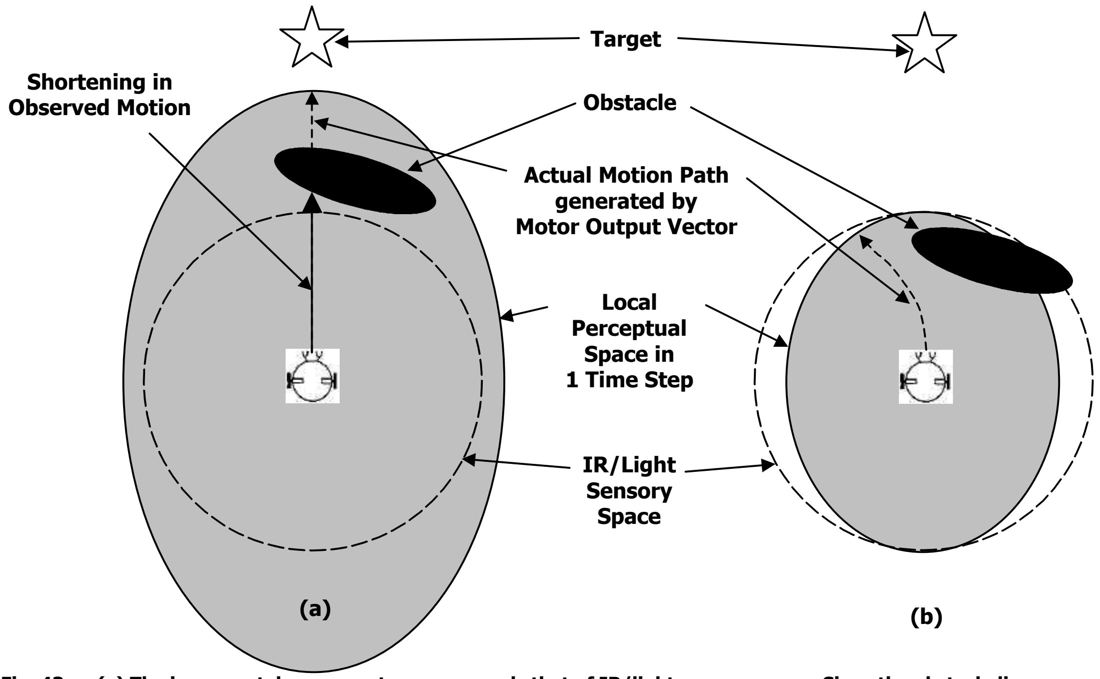

Target
Shortening in Observed Motion
Obstacle
Actual Motion Path generated by Motor Output Vector
Local Perceptual Space in 1 Time Step
IR/Light Sensory Space
(a)
(b)

Fig. 43 **(a) The incremental movement space exceeds that of IR/light sensory space. Since the obstacle lies outside the IR/light range, it is undetected. When the robot executes the actual motion path generated by the motor output vector, its motion is obstructed and shortened to the observed motion due to lack of obstacle avoidance. The observed motion is a wrong correspondence to the motor output vector, thus creating an erroneous training sample.**
(b) The incremental movement space lies within that of IR/light sensory space. Since the obstacle lies inside the IR/light range, it is detected. Thus, the motor output vector accounts for the obstacle avoidance behavior by steering to the left. The observed motion is the actual motion path generated by the same motor output vector. The training sample is therefore valid.

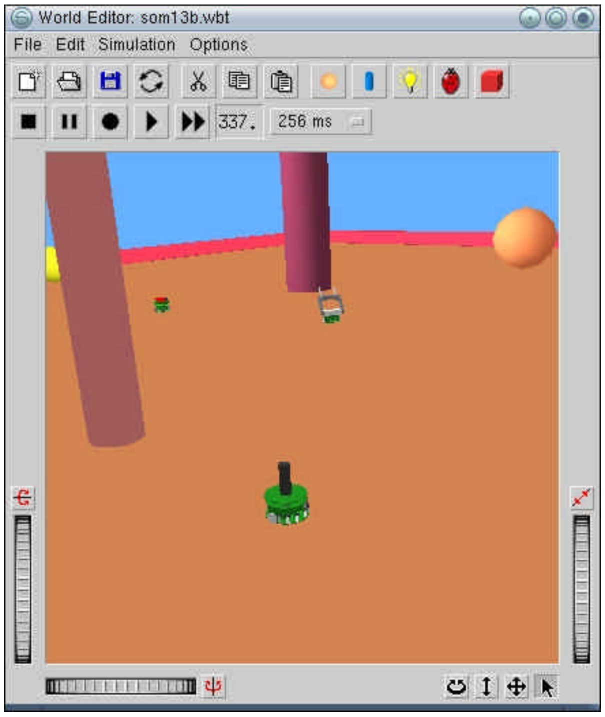


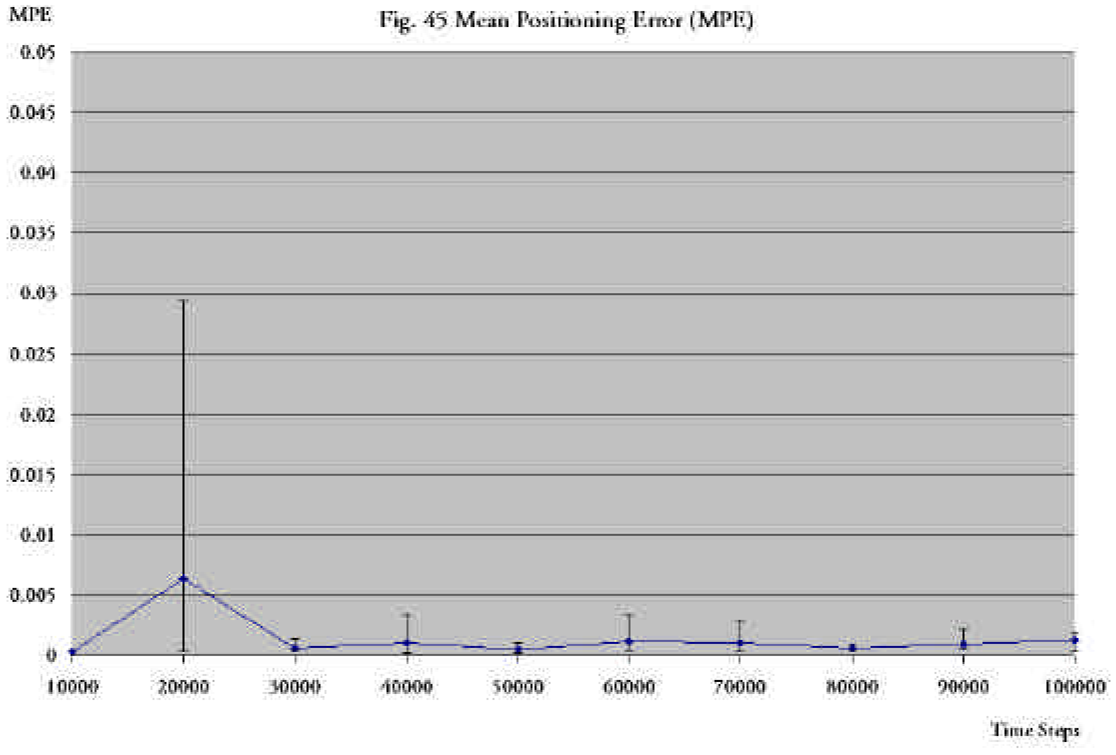
Fig. 44 **Our Khepera robot is the one with a black pole. The two other Kheperas in the background act as moving obstacles. They will move forward all the time. Upon encountering obstacles, Braitenberg vehicle type 3-C obstacle avoidance behavior will be activated in them. We can also see static obstacles like pillars, spheres and pink walls that enclose this environment.**

Parameters Initialization

Parameters in Sensorimotor Control Algorithm (Section 4.2)
$\mathcal{K} = 30$, $b = 1.0$, $g = 0.001$, $c_{min} = -20$, $c_{max} = 20$.

Parameters in Unsupervised Online Learning Algorithm (Section 4.3)
t = 256ms, $\mathcal{N} = 225$, $X = 15$, $Y = 15$, $D_{max} = 0.005$, $a_{min} = -3.0$, $a_{max} = 3.0$,
$h(0) = 1.0$, $h(t_{max}) = 0.2$, $s(0) = 0.8$, $s(t_{max}) = 0.1345$, $h'(0) = 1.0$, $h'(t_{max}) = 0.2$,
$s'(0) = 0.55$, $s'(t_{max}) = 0.38$, $r = 0.08$.

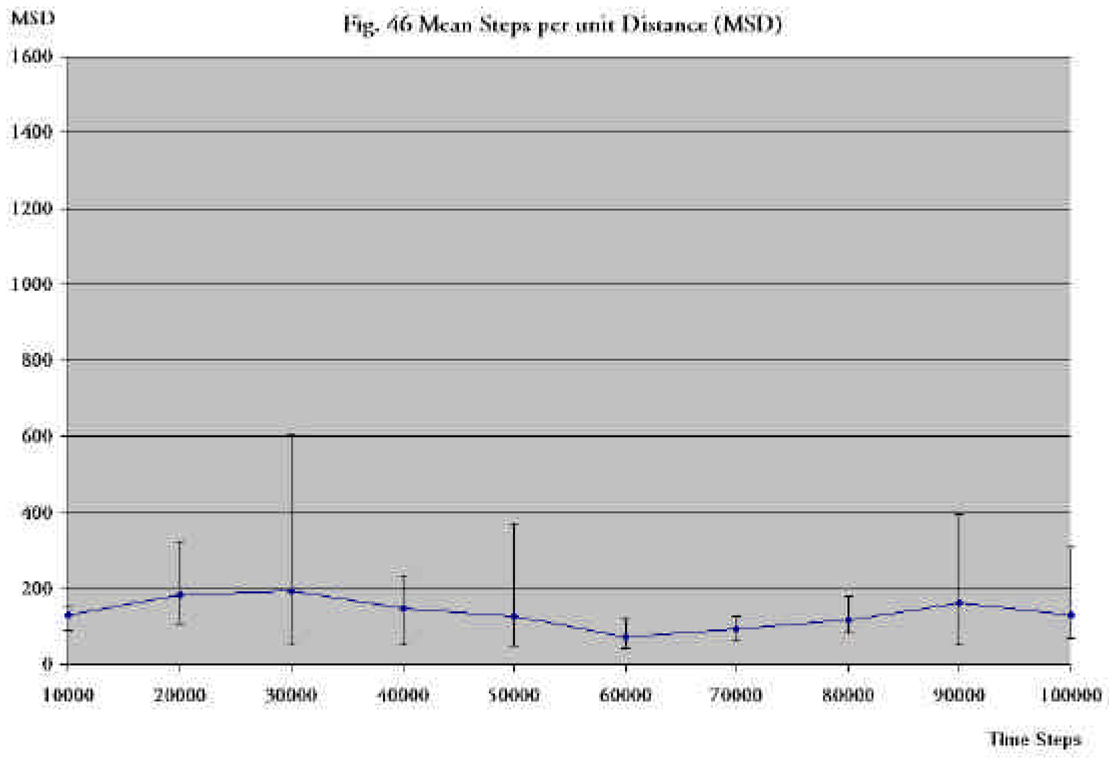
Algorithm Performance



Observations and Analysis

We can see from Fig. 45 and Fig. 46 that the network stabilizes at about 40000 time steps with MPE of 0.001 meters and 130 time steps per meter for MSD. The MPE is extremely small because we use the same number of neurons $\mathcal{N} = 225$ to cover a smaller local perceptual space. Higher accuracy would therefore be expected. Our aim is thus achieved.

Fig. 46 Mean Steps per unit Distance (MSD)



Limitations

1. The Khepera robot has 6 IR sensors in the front and only 2 IR sensors at the back. Therefore, there is a possibility that the Khepera robot may move diagonally backwards and bump into an obstacle without sensing it. If we look at Fig. 41, we can see that those directions do not have IR sensors. If the robot has IR sensors all around its body, this situation would never be encountered.
2. The reactive obstacle avoidance behavior employed in our command fusion mechanism cannot be used against complex concave obstacles. The range of the IR sensors is just too short; it would already be too late when they sense these obstacles. Other behaviors like wall following or backtracking must be incorporated into our architecture to overcome this barrier.

Chapter 7

DISCUSSION, FUTURE WORK AND CONCLUSION

7.1 Comparison with other Approaches

Concurrency of Learning and Performance Phases

As shown in the simulation results from Chapter 5, we achieve the concurrency of online learning and target reaching. This notion has never been mentioned by C. Versino and L.M. Gambardella [7] or E. Zalama et al. [5, 60].

Segmentation of Input Space

With an irregular, self-organizing topology, our method is able to reduce the self-positioning error through fine motion and also perform target reaching with minimum delay. In Section 5.3, our online learning algorithm has proven so, with the help of neighborhood update. This makes our method much more superior than E. Zalama et al. [5, 60] since it can automatically re-shuffle the neurons' weights to maximize the coverage in the input and output space to achieve a good neuron to space ratio and fine performance as well.

Immediate Output of Network

The robot control architecture proposed by Versino and L.M. Gambardella [7] and E. Zalama et al. [5, 60] maps the sensory input space directly to the motor output space. But we choose the control parameters to represent our output space. What do we take on this perspective? Are there any advantages to this notion?

Let us now try to visualize the internal works of Versino's SOM and our SOM. Version's SOM maps each local region in the input sensory space through a neuron to a discrete motor output. But our SOM maps each local region through a neuron to a region of motor output space. This is achieved by the control parameters matrix in each neuron. In this manner, we can get different motor outputs, which correspond to different sensory inputs lying in the same local region. In fact, our method attempts to find a local linear fit in that region while Versino's method finds a local constant. As mentioned in Section 4.5, local linear fit makes a better approximation [24].

Hence for each neuron **s**,

$$\mathbf{c}(\mathbf{u}) = \mathbf{c}(\mathbf{w}_s) + \mathbf{M}_s(\mathbf{u} - \mathbf{w}_s) \tag{42}$$

where **c** is the motor output vector, **u** is the input sensory vector, $\mathbf{w}_s$ is the weight sensory vector and $\mathbf{M}_s$ is the control parameters matrix for neuron **s**. To find $\mathbf{c}(\mathbf{u})$, three weights $\mathbf{w}_s$, $\mathbf{c}(\mathbf{w}_s)$ and $\mathbf{M}_s$ have to be estimated. Maintaining these three weights would require a lot of computational time, learning time and storage space. The scalability of the network would also be dampened. We can cut down to two weights $\mathbf{w}_s$ and $\mathbf{M}_s$ by doing an approximation.

$$\mathbf{c}(\mathbf{w}_s) \approx \mathbf{M}_s \mathbf{w}_s \tag{43}$$

This means that with accurate weights of $\mathbf{w}_s$ and $\mathbf{M}_s$, $\mathbf{c}(\mathbf{w}_s)$ would be redundant. Equation (42) will be reduced to

$$\mathbf{c}(\mathbf{u}) \approx \mathbf{M}_s \mathbf{u} \tag{44}$$

7.2 Future Work

Generic Nature of Sensorimotor Controller

Our proposed method can be tested on other forms of robots to validate its generic nature.

Real robots are expensive. Alternatively, we can use a mobile robot simulator, which is capable of modeling any forms of robots. Webots 3.0 is one such simulator but it is still in its making. Fig. 47 and Fig. 48 show the types of robots available in this simulator.

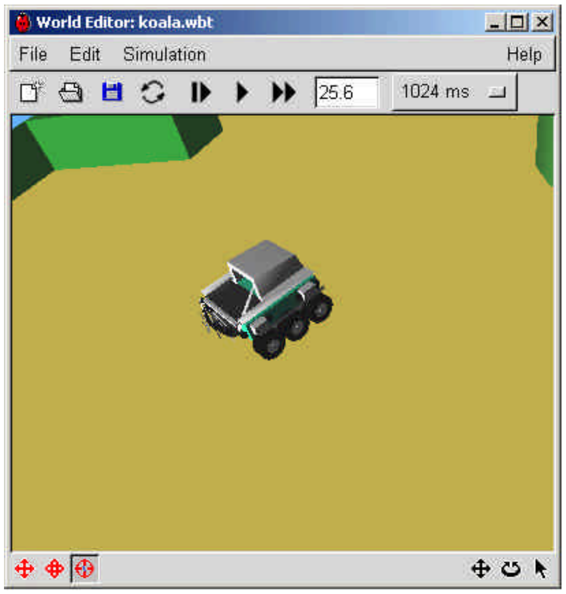


Fig. 47 Webots 3.0: Virtual Koala Robot.

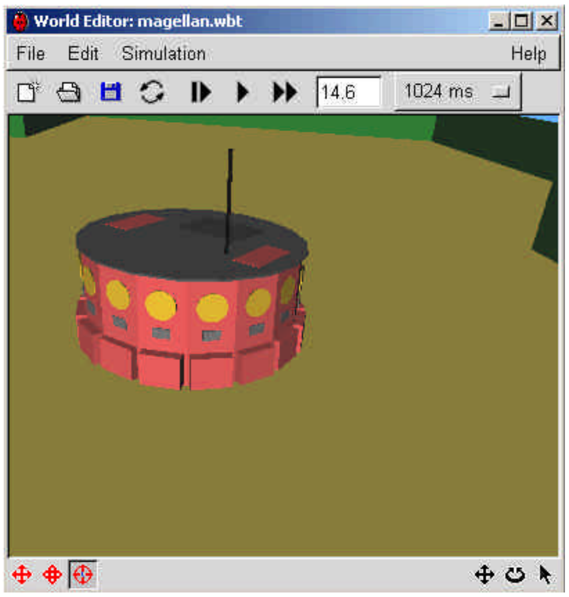


Fig. 48 Webots 3.0: Virtual Magellan Robot.

Localization

As mentioned in Section 2.1.1, dead reckoning causes the self-positioning error to accumulate over time. Our method has already eliminated the translational error. The rotational error can be accounted for if we let the robot learn a new component known as the robot heading in our SOM. When a reference landmark (one that does not change its relative position due to robot motion) is available, the robot can use this landmark to learn its own heading. If this reference landmark goes missing, the robot can use this learned heading to perform localization. This localization behavior has been detected in certain species of bees; during rainy or cloudy days, they can still navigate reasonably well.

Cognitive Map

A map-building module can be mounted on top of our current sensorimotor controller. This higher-level module defines the exploration strategies, graph structure and place or landmark recognition. Several critical problems can be encountered here. For example, how does the robot know whether it has traversed to or near a previously explored place or landmark? Particularly, if the robot is "kidnapped" and transported to another location in the map, can it find out by autonomous means its current location? Another problem would be that of perceptual aliasing; how does a robot differentiate two nodes with similar sensory signatures?

7.3 Conclusion

I will conclude by listing my contributions.

Learning Method

Our SOM network has autonomously learned the association between the sensory input and the motor output through the robot's motion. This whole self-organizing process is automatic and gains from several advantages mentioned in Section 7.1.

Motor Control Performance

The mobile robot, equipped with our trained SOM, is able to reduce its self-positioning error by performing fine motion and through iterative sensory feedback. The introduction of neighborhood update further reduces the error. The robot is also able to reach a designated target location with minimum delay.

Noise Tolerance

Our simulations performed on Webots 2.0 are subject to 10% white noise modeled in the motor output and the IR sensors. Thus, our SOM network has proven to be robust against noise and still achieve the two above-mentioned objectives. This also means that our SOM network can adapt to different environments.

Batch Training with Local Linear Smoothing

We have introduced the notion of batch training with local linear smoothing into the field of mobile robotics. I’ve yet to come across an article that does so.

REFERENCES

[1] Albus J.S., A Theory of Cerebellar Functions, *Math. Biosci.*, 10:25-61, 1971.
[2] Kohonen T., *Self-Organizing Maps*, Springer, Berlin, Heidelberg, 1995. (2nd Extended Ed., 1997).
[3] Kuperstein M. and Rubinstein J., Implementation of an Adaptive Neural Controller for Sensory-Motor Coordination. *IEEE Control Systems Magazine*, 9(3): 25-30, 1989.
[4] Piaget J., *The Origins of Intelligence in Children*, New York: International University Press, translated by M. Cook, 1952.
[5] Zalama E., Gaudiano P. and López Coronado J., A Real-time, Unsupervised Neural Network for the Low-level Control of a Mobile Robot in a Non-stationary Environment, *Neural Networks*, 8(1):103-123, 1995.
[6] Rao R.P.N. and Fuentes O., Perceptual Homing by an Autonomous Mobile Robot using Sparse Self-Organizing Sensory-Motor Maps, *In Proceedings of World Congress on Neural Networks (WCNN)*, II:380-383, 1995.
[7] Versino C. and Gambardella L.M., Learning the Visuo-Motor Coordination of a Mobile Robot by using the Invertible Kohonen Map, *In Proceedings of International Workshop on Neural Networks (IWANN95)*, Malaga, Spain, June 1995.
[8] Ritter H., Martinetz T. and Schulten K., *Neural Computation and Self-Organizing Maps: An Introduction*. Addison-Wesley, Reading, MA, 1992.
[9] Walter J.A. and Schulten K.J., Implementation of Self-Organizing Neural Networks for Visuo-Motor Control of an Industrial Robot. *IEEE Transactions on Neural Networks*, 4(1):86-95, 1993.
[10] Massone L., Sensorimotor learning, in *The Handbook of Brain Theory and Neural Networks*, M.A. Arbib, Ed., Cambridge, M.A. MIT Press, to appear.
[11] Poggio T. and Girosi F., Networks for Approximation and Learning, *Proc. IEEE*, 78:1481-1497, 1990.
[12] McIlwain J.T., Distributed spatial coding in the superior colliculus: A review, *Visual Neuroscience*, 6:3-13, 1991.
[13] Widrow B. and Hoff M.E., Adaptive Switching Circuits, *WESCON Conv Record*, 4:96-104, 1960.
[14] van der Smagt P., Groen F.C.A. and F. van het Groenewoud F., The Locally Linear Nested Network for Robot Manipulation. In *Proceedings of the IEEE International Conference on Neural Networks*, 2787-2792, 1994.
[15] Jansen A., van der Smagt P. and Groen F.C.A., Nested Networks for Robot Control, In A. F. Murray, editor, *Applications of Neural Networks*, 221-239, Kluwer Academic Publishers, Dordrecht, the Netherlands, 1995.
[16] Kohonen T., Things you haven't heard about the Self-Organizing Map. *In Proc. ICNN'93, Int. Conf. on Neural Networks*, pages 1147-1156, Piscataway, NJ, IEEE Service Center, 1993.
[17] Kohonen T., Kaski S., Lagus K., Salojarvi J., Honkela J., Paatero V., Saarela A., Self Organization of a Massive Document Collection (Draft), *IEEE Transactions on Neural Networks*, 11(3), 2000.
[18] Mitchell T.M., *Machine Learning*, McGraw-Hill, 1997.
[19] Mulier F. and Cherkassky V., Self-Organization as an Iterative Kernel Smoothing Process, *Neural Computation*, 7:1165-1177, 1995.
[20] Fan J., Design-adaptive Nonparametric Regression, *Journal of the American Statistical Association*, 87(420):998-1004, 1992.
[21] Wand M.P., and Jones M.C., *Kernel Smoothing,* London: Chapman & Hall, 1995.
[22] Su M.C., Liu T.K. and Chang H.T., An Efficient Initialization Scheme for the Self-Organizing Feature Map Algorithm, *In IJCNN'99, International Joint Conference on Neural Networks, Proceedings*, 3:1906-1910, Piscataway, N.J. IEEE Service Center, 1999.
[23] Su M.C. and Chang H.T., Fast Self-Organizing Feature Map Algorithm, *IEEE Transactions on Neural Networks*, 11(3):721-733, 2000.
[24] T. Hastie and C. Loader, Local Regression: Automatic Kernel Carpentry, *Statist. Science*, 8:120-143, 1993.
[25] Michel O., Webots: a Powerful Realistic Mobile Robots Simulator, *Proceedings of the Second International Workshop on RoboCup*, LNAI, SpringerVerlag, http://www.cyberbotics.com/, 1998.
[26] Mondada F., Franzi E. and Ienne P., Mobile Robot Miniaturization: A Tool for Investigation in Control Algorithms, *Proceedings of 3rd International Symposium on Experimental Robotics*, Kyoto, Japan, 1993, Springer Verlag, London, 501-513, 1994.
[27] Trullier O., Wiener S.I., Berthoz A. and Meyer J.A., Biologically Based Artifical Navigation Systems: Review and Prospects, *Progress in Neurobiology*, 51:483-544, 1997.
[28] Franz M.O. and Mallot H.A., Biomimetic Robot Navigation, *Robotics and Autonomous Systems*, 30: 133-153, 2000.
[29] Lagoudakis M.G. and Maida A.S., Neural Maps for Mobile Robot Navigation, *Proceedings of the 1999 International Joint Conference on Neural Networks*, Washington, D.C., July 1999.
[30] Salomon R., Self-Organizing Neural Network Controllers for Mobile Robots, In C.H. Dagli, M. Akay, O. Ersoy, B.R. Fernandez, and A. Smith (Eds.), *Proceedings of the Artificial Neural Networks in Engineering Conference (ANNIE'97)*, 581-586, ASME Press, New York, 1997.
[31] Leow W.K., Computational Studies of Exploration by Smell, *Adaptive Behavior*, 6(3/4):409-432, 1998.
[32] Borenstein J., Everett B. and Feng L., *Navigating Mobile Robots: Systems and Techniques*, A. K. Peters, Ltd., Wellesley, MA, 1996.
[33] Huber S.A., Mallot H.A. and Bulthoff H.H., *Modeling Biological Sensorimotor Control with Genetic Algorithms*, Technical Report No. 060, Max-Planck-Institute for Biological Cybernetics, Tuebingen, Germany, May 1998.
[34] Mataric M.J., Integration of Representation Into Goal-Driven Behavior-Based Robots, *The Artificial Life Route to Artificial Intelligence: Building Embodied, Situated Agents,* L. Steels and R. Brooks, eds., Lawrence Erlbaum Associates, Hillsdale, 165-186, 1995.
[35] Garcia-Alegre M.C. and Recio F., Basic Agents for Visual/Motor Coordination of a Mobile Robot, Proceedings of the first international conference on Autonomous Agents, 429-434, 1997.

[36] Martinetz T., Ritter H. and Schulten K., Learning of visuomotor coordination of a robot arm with redundant degrees of freedom. *In Proc. ICNC-90*, Dusseldorf, 431-434, North Holland, Amsterdam, March 1990.
[37] Martinetz T., Ritter H. and Schulten K., Three-dimensional neural net for learning visuomotor coordination of a robot arm, *IEEE Trans. Neural Networks*, 1(1):131-136, March 1990.
[38] Hagen S.T., l'Ecluse D. and Krose B., Q-Learning for Mobile Robot Control, *In Proc. 11th BNAIC*, 203-210, Maastricht, Nov. 1999.
[39] Rome E., Hertzberg J., Christaller Th., Kirchner F., and Licht U., Towards Autonomous Sewer Robots: The MAKRO Project, *J. Urban Water*, 1:57-70, 1999.
[40] Collett T.S., Insect Navigation En Route to the Goal: Multiple Strategies for the Use of Landmarks, *Journal of Experimental Biology*, 199:227-235, 1996.
[41] Brooks R.A., A Robust Layered Control System for a Mobile Robot, *IEEE J. Robotics and Automation*, 2(1):14-23, 1986.
[42] Maes P. and Brooks R.A., Learning to Coordinate Behaviors, *in Proceedings AAAI-91*, Boston, MA, 796-802, 1990.
[43] Mataric M.J., *A Distributed Model for Mobile Robot Environment-Learning and Navigation*, Technical Report AI-TR-1228, MIT Artificial Intelligence Laboratory, 1990.
[44] Yamagauchi B. and Beer R., Spatial Learning for Navigation in Dynamic Environments, *IEEE Transactions on Systems, Man and Cybernetics-Part B*, Special Issue on Learning Autonomous Robots, 26:496-505, 1996.
[45] Mataric M.J., Behavior-Based Control: Examples from Navigation, Learning, and Group Behavior, *Journal of Experimental and Theoretical Artificial Intelligence,* special issue on Software Architectures for Physical Agents, 9(2-3):323-336, H. Hexmoor, I. Horswill, and D. Kortenkamp, eds., 1997.
[46] Stoytchev A. and Arkin R.C., *Combining Deliberation, Reactivity and Motivation in the Context of a Behavior-Based Robot Architecture*, Mobile Robot Laboratory, College of Computing, Georgia Institute of Technology, Sep. 2000.
[47] Saffioti A., Fuzzy Logic in Autonomous Robotics: Behavior Coordination, *Procs. Of the 6th IEEE Int. Conf. On Fuzzy Systems*, Barcelona, SP, 573-578, July 1997.
[48] Krose B.J.A. and Eecen M., A Self-Organizing Representation of Sensor Space for Mobile Robot Navigation, Proc. of the IEEE/RSJ/GI *Int. Conf. on Intelligent Robots and Systems IROS'94* , Munich, Germany, 9-14, 1994.
[49] Walker A., Hallam J. and Willshaw D., Bee-havior in a Mobile Robot: The Construction of a Self-Organized Cognitive Map and its Use in Robot Navigation within a Complex, Natural Environment, *IEEE International Conference on Neural Networks*, 3:1451-1456, 1993.
[50] Chase T.A., White M., Janet J.A., Gutierrez-Osuna R. and Luo R.C., Global Self-Localization for Autonomous Mobile Robots using Self-Organizing Kohonen Neural Networks, *In Proc. of 1995 IEEE Int. Conf. on IROS*, 3:504-509, 1995.
[51] Kohonen T., Hynninen J., Kangas J. and Laaksonen J., *SOM PAK: The Self-Organizing Map Program Package*, Report A31, Helsinki University of Technology, Laboratory of Computer and Information Science, January 1996.
[52] Fritzke B., Some Competitive Learning Methods, http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/JavaPaper/, April 1997.
[53] Kaski S., Kangas J., Kohonen T. Bibliography of Self-Organizing Map (SOM) Papers: 1981-1997, Neural Computing Surveys 1, 102-350, 1998. http://www.icsi.berkeley.edu/~jagota/NCS
[54] Kohonen T., 4311 Works that have been based on the Self-Organizing Map (SOM) method developed by Kohonen, Neural Networks Research Centre, Helsinki University of Technology, 2000. http://www.cis.hut.fi/nnrc/refs
[55] Hofner C. and Schmidt G., Path Planning and Guidance Techniques for an Autonomous Mobile Cleaning Robot, *J. Robotics and Autonomous Systems*, 14:199-212, 1995.
[56] Braitenberg V., *VEHICLES – Experiments in Synthetic Psychology*, MIT Press, Cambridge, MA, 1984.
[57] Pirjanian P., *Behavior Coordination Mechanisms -- State-of-the-art*, Technical Report IRIS-99-375, Institute of Robotics and Intelligent Systems, School of Engineering, University of Southern California, October 1999. http://iris.usc.edu/~irislib
[58] Sahin E. and Gaudiano P., KITE: The Khepera Integrated Testing Environment, *Proceedings of the First International Khepera Workshop*, Paderborn, Germany, 199-208, 1999. http://www.starlab.org/neurobotics/
[59] Webb B., Robots, Crickets and Ants: Models of Neural Control of Chemotaxis and Phototaxis, *Neural Networks*, 11:1479-1496, 1998.
[60] Gaudiano P., Zalama E. and López Coronado J., An Unsupervised Neural Network for Low-level Control of a Mobile Robot: Noise Resistance, Stability, and Hardware Implementation, *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 26:485-496, 1996.
[61] Schonbein, W., Khep_Core Package Version 0.5, PNP Robot Lab, Department of Philosophy, Washington University, March 2000. http://artsci.wustl.edu/~philos/pnp/robotlab/robotlab.html
[62] Sarle W.S., *Neural Network FAQ, part 1 of 7: Introduction,* periodic posting to the Usenet newsgroup comp.ai.neural-nets, ftp://ftp.sas.com/pub/neural/FAQ.html, ed. 1997.
[63] Press W.H., Teukolsky S.A., Vetterling W.T. and Flannery B.P., *Numerical Recipes in C, The Art of Scientific Computing*, 2nd ed., Cambridge Univ. Press, 1992.
[64] Makhoul J., Roucos S., Gish H., Vector Quantization in Speech Coding, *Proc. IEEE*, 73:1551-1558, 1985.