

國立交通大學

電機與控制工程學系

博士論文

自我組織特徵映射網路於最佳化問題及其應用

An SOM-Based Algorithm for Optimization
and Its Applications

研究生：陳一元

指導教授：楊谷洋 教授

中華民國九十七年七月

自我組織特徵映射網路於最佳化問題及其應用

An SOM-Based Algorithm for Optimization
and Its Applications

研究生：陳一元

Student : Yi-Yuan Chen

指導教授：楊谷洋

Advisor : Kuu-Young Young



A Dissertation
Submitted to Department of Electrical and Control Engineering
College of Electrical and Computer Engineering
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy
in
Electrical and Control Engineering

July 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年七月

自我組織特徵映射網路於最佳化問題及其應用

An SOM-Based Algorithm for Optimization and Its Applications

研 究 生：陳一元

指 導 教 授：楊谷洋 博士

國立交通大學電機與控制工程學系



摘要

自我組織特徵映射神經網路 (SOM) 已經廣泛地應用在靜態資料處理與動態資料的分析，但利用 SOM 解決最佳化的問題的研究非常少。目前以 SOM 為基礎的最佳化演算法對動態系統最佳化的效能還有待改進，所以在本論文中提出一自我組織特徵映射神經網路最佳化演算法 (SOMS) 應用於靜態與動態最佳化問題。為了更進一步擴展它的收尋能力，也提出一個新的鍵結值的調整規則以達到動態調整 SOM 的鄰域函數。在論文中我們也利用 SOMS 演算法發展一智慧型雷達預估器，可在很短的時間週期內對於只有很少的資料被雷達接收的情形下，估測目標物的運動軌跡。除此之外，當最佳化問題存在多個最佳解時，利用一個新的 Niching 方法 (即決策型競爭機制)，我們也提出一個 Niching 型自我組織特徵映射神經網路最佳化演算法 (NSOMS)。為了提高學習的效能且同時可以讓最佳解的分佈結構顯現在二維的輸出空間，我們提出一新的神經元鍵結值與座標位置的調整規則，由於新的調整規則的設計簡單而且只用到兩個學習參數分別於神經元鍵結值與座標位置上，所以提出的 NSOMS 可以很容易地應用在不同的最佳化問題上。我們以模擬的方式來驗證此方法的可行性，並與傳統的卡曼濾波器 (KF)，基因演算法 (GA)，與其他 SOM 最佳化演算法進行比較。

An SOM-Based Algorithm for Optimization and Its Applications

Student : Yi-Yuan Chen

Advisor : Dr. Kuu-Young Young

Department of Electrical and Control Engineering
National Chiao Tung University



Abstract

The self-organizing map (SOM), as a kind of unsupervised neural network, has been used for both static data management and dynamic data analysis. To further exploit its search abilities, in this dissertation we propose an SOM-based search algorithm (SOMS) for optimization problems involving both static and dynamic functions. Furthermore, a new SOM weight updating rule is proposed to enhance the learning efficiency; this may dynamically adjust the neighborhood function for the SOM in learning system parameters. Based on the SOMS, we develop an intelligent radar predictor to achieve accurate trajectory estimation under the strict time constraint due to only few data are available in every short time period. Moreover, when an optimization problem has many different optimal solutions, a new niche method (deterministic competition mechanism) to extend SOM-based search algorithm (NSOMS) has been proposed for identification of multiple optimal solutions. The proposed NSOMS network structure is able to find multiple different optimal solutions and visualize distribution and structure of optimal solutions, allowing us to easily classify the optimal solutions into clusters. As a demonstration, the proposed NSOMS is applied for function optimization in a multimodal domain and also dynamic trajectory prediction involving multiple targets, with its performance compared with the genetic algorithm (GA) and other SOM-based optimization algorithms.

Acknowledgements

以最誠摯的心感謝我的指導教授楊谷洋博士，在我攻讀碩士及博士學位期間，在學術研究領域上給予我最專業的指導與諄諄教誨，使我的博士論文得以順利完成。此外我也從楊谷洋博士身上學習到對於研究應有的態度，懷抱著熱忱，加以認真、踏實、努力的精神、持續不輟的耐心，才能在學術研究這條路上堅持理想，朝著目標邁進。也感謝蘇木春教授、林進燈教授、周志成教授、林昇甫教授、陶金旭教授，以及黃士殷教授，撥冗參與我的論文口試，你們的不吝指導與寶貴的建議，使我獲益良多，並不斷砥礪自己精益求精。

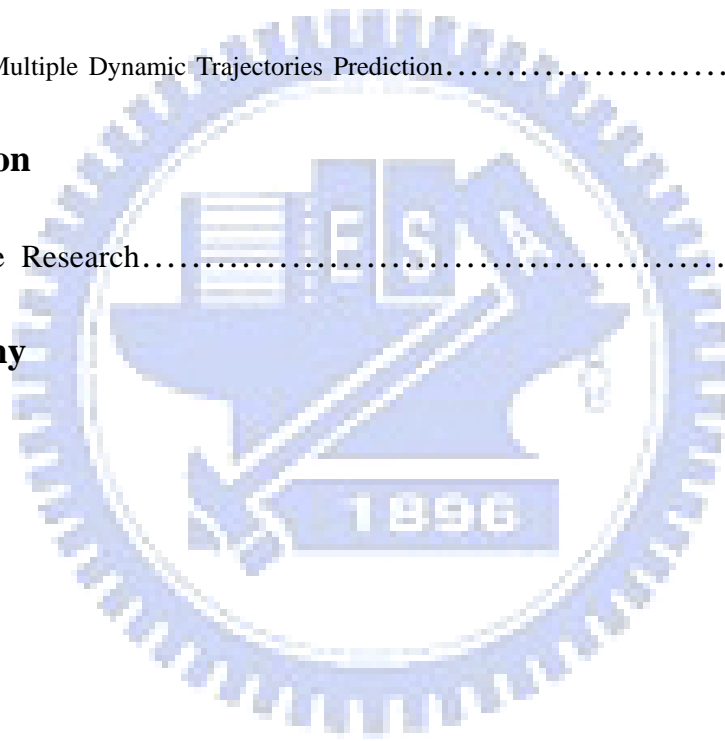
感謝人與機器實驗室的學弟們，在論文研究上給予最佳的建議與討論，使我的研究可以更加事半功倍。感謝我的父母、弟弟、太太、兒女，在我求學的路途上給予我最堅定的支持與鼓勵，使我能堅持目標繼續奮鬥，因為你們的關懷與愛護，所以我能心無旁騖的完成學業，也讓我感受到家人的關懷是支持我在求學路上最堅強、最溫暖的力量。

求學生涯告一段落，但這不代表學習的結束，而是另一階段的開始。感謝所有愛護我的人對我的支持與鼓舞，我將帶著你們給我的信心，在學術研究的路上繼續努力！
謹以此論文獻給愛我的師長、家人、朋友們！謝謝大家。

Contents

Chinese Abstract	i
English Abstract	ii
Acknowledgements	iii
Contents	iv
List of Tables	vi
List of Figures	vii
1 Introduction	1
2 Intelligent Radar Predictor	5
2.1 Dynamic Trajectory Prediction Based on Self-Organizing Map.....	9
2.2 SOM Implementation.....	12
2.3 Simulation.....	18
3 SOM-Based Algorithm for Optimization	28
3.1 Proposed SOM-Based Search Algorithm (SOMS).....	29
3.2 Proposed Weight Updating Rule.....	31
3.3 Applications.....	37
3.3.1 Function Optimization.....	38
3.3.2 Dynamic Trajectory Prediction.....	43

4 Niching SOM-Based Search Algorithm (NSOMS)	52
4.1 Niching Method.....	53
4.2 Proposed Niching SOMS Weight Updating Rule.....	57
4.3 Visualization of Distribution of Optimal Solutions.....	62
4.4 Applications.....	64
4.4.1 Function Optimization of a Multimodal Domain.....	64
4.4.2 Multiple Dynamic Trajectories Prediction.....	72
5 Conclusion	91
5.1 Future Research.....	92
Bibliography	93
Vita	99



List of Tables

4.1 Comparison results for NSOMS and RCS-PSM on the 4 test functions.....	69
4.2 Comparison results for NSOMS, SOMSO-1, and SOMSO-2 on the dynamic trajectory prediction.....	87



List of Figures

2.1	A conceptual diagram of an air-defense radar system.....	7
2.2	System organization of the proposed intelligent radar predictor.....	11
2.3	The structure and operation in the SOM.....	11
2.4	The movement of the weight vector in the two-dimensional space: (a) $\underline{w}_j^s \neq \underline{w}_j^*$ and (b) $\underline{w}_j^s = \underline{w}_j^*$	13
2.5	Simulation results for trajectory prediction using the SOM and the Kalman filter with good estimates of both the initial condition and noise distribution: (a) the ideal and measured TBM trajectories, (b) the estimated position error by using the SOM and Kalman filter, and (c) the movement of the weight vector \underline{w}_j during the SOM learning process.....	21
2.6	Simulation results for trajectory prediction using the SOM and the Kalman filter with a good estimate of the initial condition but bad estimate of the noise distribution: (a) the ideal and measured TBM trajectories, (b) the estimated position error by using the SOM and Kalman filter, and (c) the movement of the weight vector \underline{w}_j during the SOM learning process.....	23
2.7	Simulation results for trajectory prediction using the SOM and the Kalman filter with a bad estimate of the initial condition but good estimate of the noise distribution: (a) the ideal and measured TBM trajectories, (b) the estimated position error by using the SOM and Kalman filter, and (c) the movement of the weight vector \underline{w}_j during the SOM learning process.....	24
2.8	Simulation results for trajectory prediction using the SOM and the Kalman filter with bad estimates of both the initial condition and the noise distribution: (a) the	

ideal and measured TBM trajectories, (b) the estimated position error by using the SOM and Kalman filter, and (c) the movement of the weight vector \underline{w}_j during the SOM learning process.....	25
2.9 Results by applying the SOM to predict the actual TBM trajectory.....	26
3.1 Conceptual diagram of the organized search in a 2-D SOM: the solution is (a) within the estimated range and (b) outside of the estimated range.....	30
3.2 Proposed SOM-based algorithm for optimization.....	33
3.3 Structure and operation of the SOM in the SOMS.	33
3.4 Center and width adjustment for the neighborhood function $G(w_{j,i}(k))$, when (a) $(w_{e_i} - w_{j^*,i}(k))^2 \geq \sigma_i^2$, and (b). $(w_{e_i} - w_{j^*,i}(k))^2 < \sigma_i^2$	36
3.5 Minimization of the 2-D Griewant function using the SOMS and GA with the optimal solution outside of the estimated region: (a) minimal function values $O(\underline{w}_{j^*}(k))$ during the learning process, (b) weight vector movement in the SOM, and (c) weight vector movement in the GA.....	41
3.6 Minimal function values $O(\underline{w}_{j^*}(k))$ during the learning process for the minimization of the 30-D Rosenbrock function using the SOMS, GA, and SOMO.....	42
3.7 Simulation results for dynamic trajectory prediction using the SOMS, SOMSO, and GA with a good estimate of the initial state: (a) the estimated position error in the X-direction and (b) the variation of the neighborhood function $F(\underline{w}_j(k))$ during the SOMS learning process.....	50
3.8 Simulation results for dynamic trajectory prediction using the SOMS, SOMSO, and GA with a bad estimate of the initial state.....	51
4.1 Optimization during learning process: (a) without the niching method, (b) with the niching method.....	55

4.2	Proposed Niching SOM-based search algorithm.....	56
4.3	Structure and operation of the SOM in the NSOMS.....	56
4.4	Three multimodal functions. (a) F1: uniform sine function, (b) F2: nonuniform sine function, and (c) F3: Shekel's Foxholes function.....	67
4.5	Convergence comparisons for F3 function: (a) the variation of the number of maintained peaks and (b) the variation of the maximum peak ratio during the NSOMS learning process.....	70
4.6	The results obtained by the NSOMS for F4 function: (a) projection result in the 2D neuron space, and (b) final neighborhood function values.....	71
4.7	Simulation results for the multiple trajectories prediction using the NSOM, SOMSO-1, and SOMSO-2 with a good estimate of the initial state: (a) the ideal and measured TBM trajectories, (b)-(d) the estimated initial state error by using the NSOM, SOMSO-1, and SOMSO-2.....	77
4.8	Final results obtained by the NSOMS for the multiple trajectories prediction: (a) projection result in 2D neuron space and (b) final neighborhood function values....	78
4.9	Simulation results for multiple trajectories prediction using the NSOM, SOMSO-1, and SOMSO-2 with a bad estimate of the initial state: (a) the ideal and measured TBM trajectories, (b)-(d) the estimated initial state error by using the NSOM, SOMSO-1, and SOMSO-2.....	81
4.10	Final results obtained by the NSOMS for the multiple trajectories prediction : (a) projection result in 2D neuron space and (b) final neighborhood function values.....	82
4.11	Performance for different network sizes using the NSOMS, SOMSO-1, and SOMSO-2.....	84
4.12	Performance for different parameters using the NSOMS, SOMSO-1, and SOMSO-2.	86
4.13	Performance of 5 runs with the same initial weights using the NSOMS, OMSO-1,	

and SOMSO-2.....89

4.14 Performance of 5 runs with the different initial weights using the NSOMS,
SOMSO-1, and SOMSO-2.....90



Chapter 1

Introduction

The self-organizing map (SOM), as a kind of unsupervised neural network, is performed in a self-organized manner in that no external teacher or critic is required to guide synaptic changes in the network [4, 22]. By contrast, for the other two basic learning paradigms in neural networks, supervised learning is performed under the supervision of an external teacher [14] and reinforcement learning involves the use of a critic that evolves through a trial-and-error process [3]; these other two also demand the input-output pairs as the training data. The appealing features of learning without needing the input-output pairs makes the SOM very attractive when dealing with varying and uncertain data. In its many applications, the SOM has been used for both static data management and dynamic data analysis, such as data mining, knowledge discover, clustering, visualization, text archiving, image retrieval, speaker recognition, mobile communication, robot control, identification and control of dynamic systems, local dynamic modeling, nonlinear control, and tracking moving objects [1, 2, 14, 22, 24, 31, 36, 37, 38, 42]. There have also been many approaches proposed to improve or modify the original SOM algorithm for different purposes [2, 13, 19, 37, 43, 45, 47]. However, from our survey, its search abilities have not been adequately

exploited yet [7, 12, 16, 29, 30, 40]. This need thus motivates us to propose an SOM-based search algorithm (SOMS) for both static and dynamic functions.

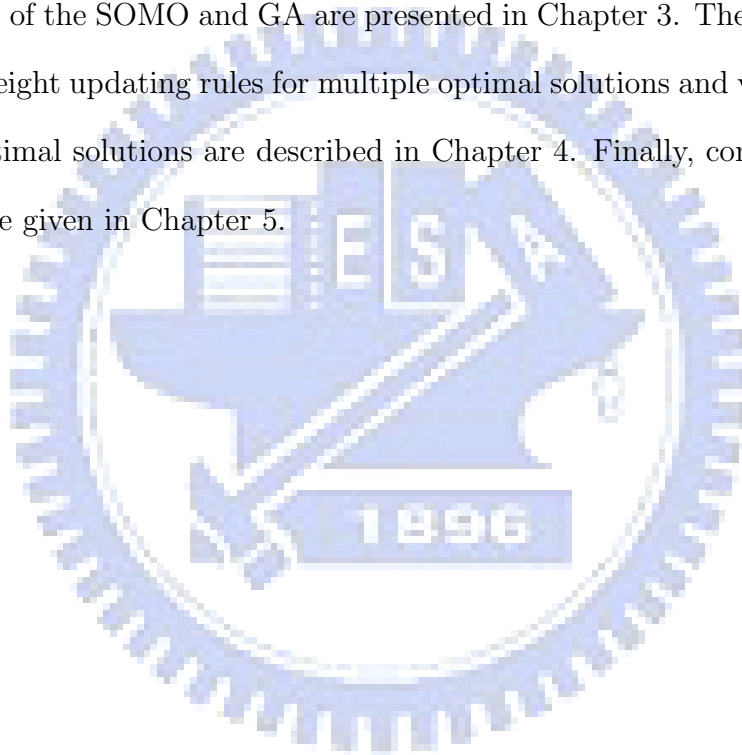
In recent years, some new research studies have turned to tackle the continuous optimization problems based on the self-organizing map. Michele et al. proposed an optimization method based on the Kohonen SOM evolution strategy (KSOM-ES) [29]. Su et al. proposed the SOM-based optimization algorithm (SOMO) [40]. An self-organizing and self-evolving agents (SOSENs) neural network that combines multiple simulated annealing algorithms (SAs) and SOM algorithm have also been proposed [44]. Our proposed SOMS will extend the application further to optimization problems involving dynamic functions. When searching for a dynamic function, the goal may be to look for a set of optimal parameters that lead to the desired performance of the dynamic system from limited measured data. In this dissertation we first apply the self-organizing map (SOM) to develop an intelligent radar predictor. With the few radar data read into the predictor in each time interval and a simplified dynamic model of the moving target, the SOM learns to estimate the initial state of the target trajectory in each learning cycle, and will gradually converge to the optimal initial state. To achieve high learning efficiency under such widely varying parameters, we propose a new weight updating rule which may dynamically adjust the shape and location of the neighborhood function for the SOM, in an individual basis, in learning the system parameters. Thus, the proposed SOMS should be able to execute both system performance evaluation and the subsequent search in a real-time manner.

Many optimization problems often have more than one optimal solutions in the feasible domain [15, 34, 41]. If more different optimal solutions can be found, it is advantageous for us to have the right of choice. Although many global optimization techniques based

on population evolutionary have been successfully applied to find the global optimum [12, 17, 21], they cannot be directly applied to search for multiple solutions through one search process in a multimodal domain. Mahfoud proposed the niching methods to identify multiple optima in a multimodal domain [28]. When many optimal solutions are obtained, how to classify the set of optimal solutions and select one useful solution is difficult, and in particular depends on the number of the optimal solutions and the size of dimension. Igarashi used GA to find many different optimal solutions repeatedly and applied the SOM for visualization and clustering of optimal solutions in 2-D output spaces [15]. The visualization of high-dimensional data is one of the well-known merits of the SOM. However, the SOM algorithm proposed in [15] was not applied to optimization. Although the SOM has been used to tackle the optimization problems [29, 40, 44] and its performances have been manifested better than other search algorithms such as SA [21], PSO [17], DE [25], and GA [12], their search abilities were still not well exploited in finding multiple optimal solutions. Meanwhile, it did not simultaneously provide visualization of the distribution of optimal solutions, either. Thus, we further propose a niching SOM-based search algorithm (NSOMS) for identification and visualization of the multiple optimal solutions.

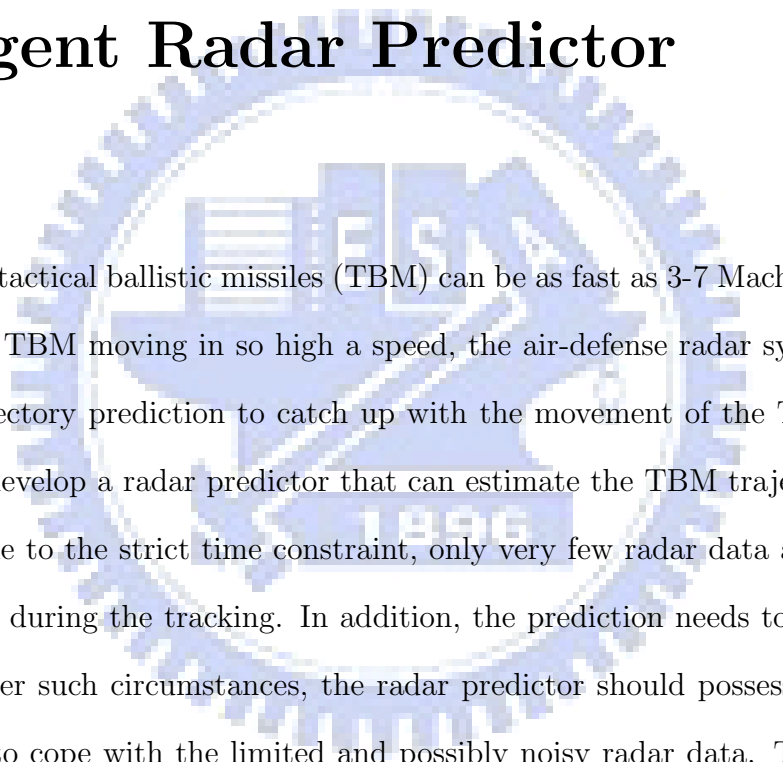
This new niching method is proposed to extend the SOMS by defining subpopulations (subspaces) in a multimodal domain. With the proposed niching weight-updating rule, the niche location located on the winning weight site will be moved to approach the real peak location of a multimodal domain gradually. Thus, with many different niches set, the NSOMS can be applied to searching for multiple optimal solutions. For visualization of distribution of optimal solutions, the concept of the double SOM (DSOM) [39], which updates the weight vectors together with the two-dimensional position vector of the neuron, is employed in the proposed NSOMS. The optimal solutions in the parameter space

are mapped onto a two-dimensional (2-D) neuron space. Through this map it allows us to classify the optimal solutions into clusters. We then apply proposed NSOMS to function optimization in a multimodal domain and multitarget tracking problem simultaneously with data sent from multiple sensors. The rest of this dissertation is organized as follows. The proposed intelligent radar predictor including SOM and the performance of the SOM compared with that of the Kalman filtering are discussed in Chapter 2. The proposed SOMS with dynamic weight updating rules and also the performance of the SOM compared with that of the SOMO and GA are presented in Chapter 3. The proposed NSOMS with the new weight updating rules for multiple optimal solutions and visualization of distribution of optimal solutions are described in Chapter 4. Finally, conclusions and some future works are given in Chapter 5.



Chapter 2

Intelligent Radar Predictor



Nowadays, the tactical ballistic missiles (TBM) can be as fast as 3-7 Mach. For successful tracking of the TBM moving in so high a speed, the air-defense radar system should be capable of trajectory prediction to catch up with the movement of the TBM. It is then imperative to develop a radar predictor that can estimate the TBM trajectory using the radar data. Due to the strict time constraint, only very few radar data are available for each prediction during the tracking. In addition, the prediction needs to be executed in real time. Under such circumstances, the radar predictor should possess certain degree of intelligence to cope with the limited and possibly noisy radar data. The challenge of developing an intelligent radar predictor for accurate trajectory estimation motivates the study in this dissertation.

Before discussing the proposed intelligent radar predictor, we first briefly describe an air-defense radar system, as shown in Figure 2.1 [23]. In Figure 2.1, a TBM is launched from location A; the radar system then tracks its trajectory, in addition to predicting the possible landing site, location B. Under successful tracking, the radar system can then

guide the intercepting missile to penetrate into the predicted trajectory of the incoming TBM, and destroy it as early as possible. From the different launching site relative to location B, location C or D, the intercepting missile may take different route to enter the predicted TBM trajectory, as shown in Figure 2.1. To let the intercepting missile follow the trajectory shown in Figure 2.1, missile guidance law is demanded. By commanding the acceleration of the missile proportional to the angular rate of some desired direction, the guidance law will turn the heading of the missile toward that direction as rapidly as possible [23, 46]. As the intruding missile is in such a high speed and small volume, the design of guidance law for the intercepting missile is very challenging.

Missile guidance can basically be divided into two stages: midway guidance and terminal guidance. During the midway guidance stage, the information from the ground radar is used to guide the intercepting missile. When the target trajectory can be precisely predicted, the intercepting missile may not need to chase after an extremely fast target, but just move toward the predicted target location. After the intercepting missile is led close to the target, the seeker, as an active radar equipped on the missile, may then take over and proceed with the terminal guidance. As the intruding TBM may be capable of escaping, delicate guidance laws need to be installed in the seeker to provide more complex maneuvering. It can be seen that control load for missile interception in both stages of guidance can be tremendously alleviated, if the radar system is able to estimate the TBM trajectory accurately. Meanwhile, trajectory prediction can also provide the possible TBM landing location, and thus be helpful in determining a proper location and direction to launch the intercepting missile.

One famous approach for trajectory prediction is the Kalman filtering, which has been widely used in predicting the movements of the satellites, airplanes, ships, etc. [32].

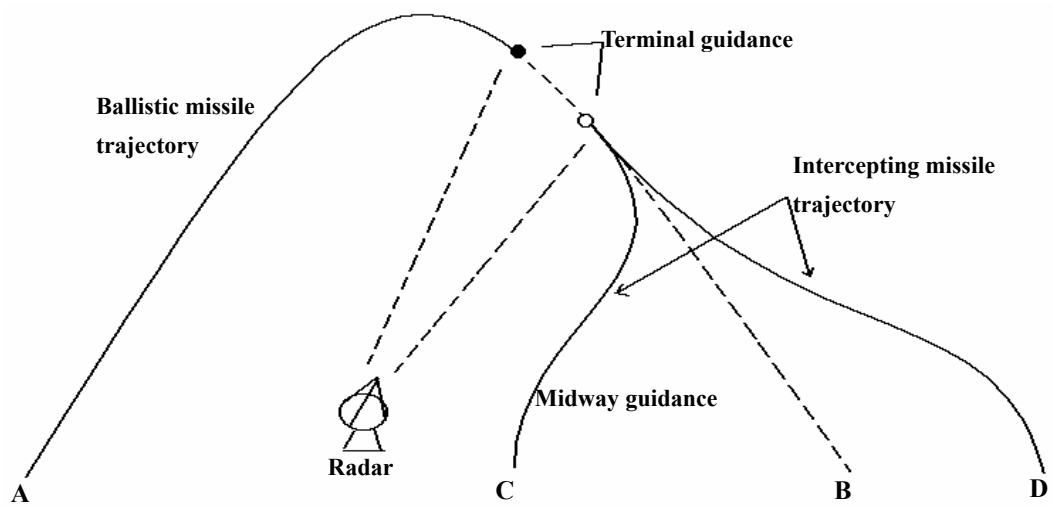


Figure 2.1 A conceptual diagram of an air-defense radar system.

By knowing the dynamic model of the moving target, the Kalman filter in general yields satisfactory performance when the statistics of the environmental noises and good guesses of the initial conditions can be obtained in advance. However, the Kalman filtering may not be suitable for unknown, noisy environments. To tackle the situation aforementioned, researches have been dedicated to build mathematical models, perform statistical data analysis, and make the Kalman filter more adaptive [5, 9, 27]. It is by no means an easy task to cope with the complexity involved in modeling, though. As an alternative, the learning mechanism has been used to assist the Kalman filter, since it is model-free and computational efficient after training. Among them, Roberts et al. proposed a neurofuzzy estimator to improve the Kalman filter initialization [33]. Chin proposed incorporating the neural network into the Kalman filter configuration to deal with the multi-target tracking problem [6].

The SOM, first introduced by Kohonen, transforms input vectors into a discrete map (e.g., a 2-D grid of neurons) in a topological ordered fashion adaptively [22, 37]. During each iteration of learning, the each neuron competes with each other to gain the opportunity to update its weight, and the vector that generates the output most close to the desired value (vector) is chosen as a winner. Because the SOM allows local interaction between neighboring neurons, the weights of the winner and also its neighbors are all updated. Through repeated weight modification, a cluster (or clusters) may form and become more and more compact until a final configuration develops. The SOM thus has a structure very suitable for parallel processing. We further exploit this parallelism and design an organized search accordingly. In other words, we take advantage of the SOM in its distribution of the neurons in a grid pattern and the presence of local interaction in between the grid.

The SOM in the proposed predictor is in fact used as a search mechanism, and the simplified TBM dynamic model as a reference for the SOM to approach the neighborhood of the TBM. The employment of the SOM in this way is different from those in most of its previous applications, and well exploits its capability in searching. The proposed intelligent radar predictor including SOM is described in next section.

2.1 Dynamic Trajectory Prediction Based on Self-Organizing Map

Figure 2.2 shows the system organization of the proposed intelligent radar predictor. For an incoming TBM, the predictor uses the measured position data sent from the radar to predict the TBM trajectory. The main module in this predictor is the SOM [22]. We may distribute the possible positions and velocities of the missile (as weight vectors) into the network in an organized fashion. Under this arrangement, the searches among the neurons are closely related through the grid, leading to a more rapid convergence. On the other hand, when the estimation is inaccurate, the search, still organized, may take longer time to converge to the optimal solution.

The adoption of the SOM to realize this intelligent radar predictor is that it does not demand input-output pairs for on-line prediction when facing unknown environments. Thus, the genetic algorithm (GA) may also be a possible alternative [12]. Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. It employs multiple concurrent search points called chromosomes and evaluates the fitness of each chromosome. The search procedure uses random choice as a tool to guide a highly exploitative search through a coding of a parameter space. We consider

the SOM is more suitable than the GA for this trajectory tracking problem. The reason is that the connection among the possible initial states for missile launching is not utterly random. The SOM better exploits the relationship between the initial state and its resultant TBM trajectory, leading to a somewhat guided search. In addition, the GA is in general more time-consuming. As the prediction needs to be accomplished within a limited amount of time, the efficiency of the network is crucial.

In this TBM tracking application, the SOM is used to estimate the initial position, velocity, and acceleration of the TBM using the measured position data from the radar. Thus, if the dynamic model of the TBM is available, the entire TBM trajectory can be derived using those estimated initial position, velocity, and acceleration as the initial state for the dynamic model. Through a learning process, the SOM determines a most probable initial state by comparing the measured position data with the predicted TBM position trajectories derived from a number of possible initial states selected from a predicted range. The process of how the SOM learns to estimate the initial state is as follows. First, a number of vectors, each of which contains a possible initial state, are selected and stored into the neurons of the SOM. During each time interval, the SOM sends these vectors to the dynamic model of the TBM to compute the corresponding trajectories. By comparing the predicted trajectories with the measured radar data, the vector corresponding to the most accurate predicted trajectory is chosen as the winner. The weights of this winner and its neighbors are updated, and the network will eventually converge to the optimal initial state. To note that, even the optimal initial state is not within those vectors initially selected from the predicted range, the SOM is able to move these vectors out of their original locations and guide them to converge to the optimal initial state. Implementation details of the SOM for trajectory prediction are given in next section.

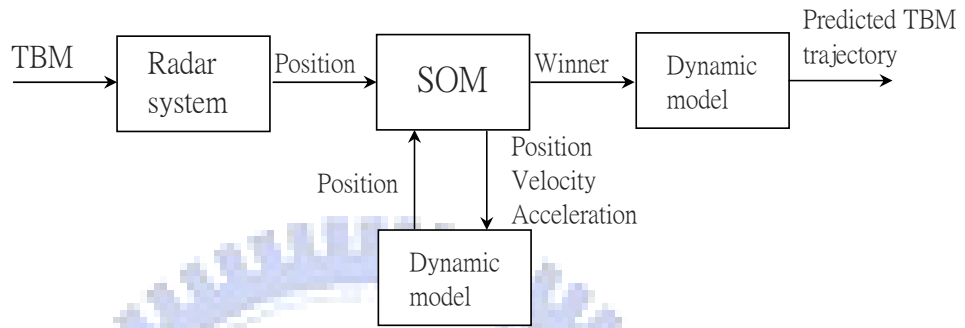


Figure 2.2 System organization of the proposed intelligent radar predictor.

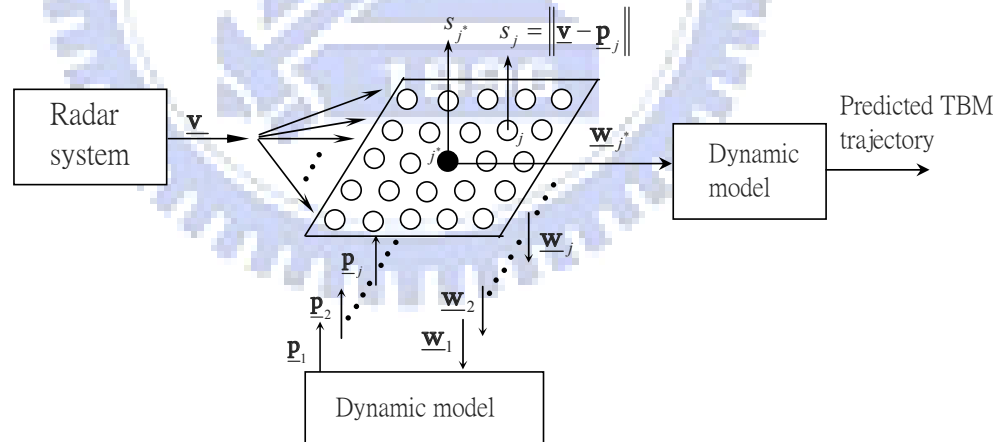


Figure 2.3 The structure and operation in the SOM.

2.2 SOM Implementation

For this TBM tracking application, the SOM needs to tackle the spatio-temporal data, instead of the spatial data it usually deals with. Therefore, a dynamic model that describes the behavior of the TBM is included in the intelligent radar predictor. By combining the SOM with the dynamic model, the SOM is able to tackle the spatio-temporal radar data. Figure 2.3 shows the structure and operation in the SOM. A 2D SOM is used for illustration in Figure 2.3, and the SOM can also be three-dimensional or other according to the applications. Each time a certain number of new measured position data $\underline{\mathbf{v}}$ are sent in from the radar system, the SOM is triggered to operate. And, it will gradually converge to an optimal prediction along with the increase of the measurement data and learning time. In Figure 2.3, for each neuron j in the SOM, it contains a vector of a possible initial state $\underline{\mathbf{w}}_j$ and generates an output s_j . By sending $\underline{\mathbf{w}}_j$ to the dynamic model, s_j is computed as the difference between $\underline{\mathbf{v}}$ and the predicted trajectory $\underline{\mathbf{p}}_j$. Of all the neurons, the neuron j^* with the smallest output s_{j^*} is chosen as the winner. When the weight of this winning neuron j^* differs from that of the previous winner \hat{j}^* (i.e., $\underline{\mathbf{w}}_{j^*} \neq \underline{\mathbf{w}}_{\hat{j}^*}$), the weights of \hat{j}^* and its neighbors will be updated in a manner that moves these weight vectors toward neuron j^* , as shown in Figure 2.4(a). When j^* is the same as \hat{j}^* (i.e., $\underline{\mathbf{w}}_{j^*} = \underline{\mathbf{w}}_{\hat{j}^*}$), the weights will then be updated so as to let the weight vectors form more and more compact clusters centering at neuron j^* , as shown in Figure 2.4(b). Under successful learning, the SOM will finally converge to a predicted optimal initial state.

Several parameters need to be determined in implementing this SOM, including the learning rate, topological neighborhood function, and number of radar data used for s_j computation. The selection of the learning rate η depends on the closeness of $\underline{\mathbf{w}}_{j^*}(k)$ and $\underline{\mathbf{w}}_{\hat{j}^*}(k)$. When they are different from each other, we intend to speed up the learning

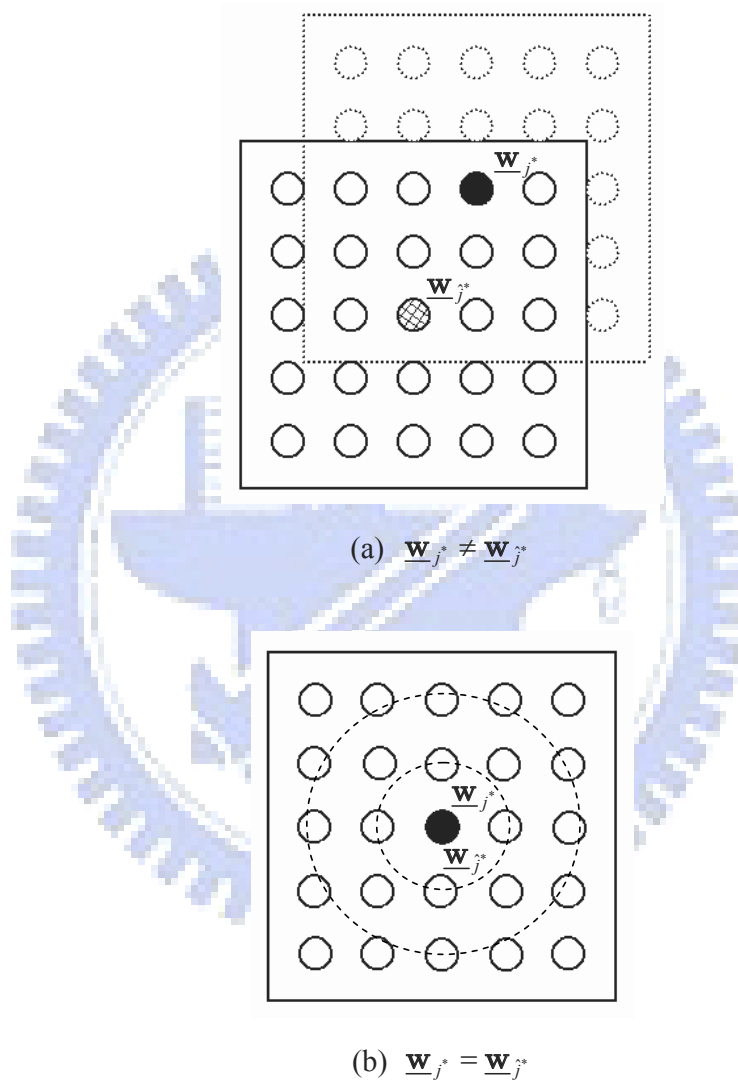


Figure 2.4 The movement of the weight vector in the two-dimensional space: (a) $\underline{\mathbf{w}}_j^* \neq \underline{\mathbf{w}}_j^*$ and (b) $\underline{\mathbf{w}}_j^* = \underline{\mathbf{w}}_j^*$.

process and choose $\eta(k)$ in the k th stage of learning to be close to 1. And when they almost coincide, we slow down the learning gradually and determine $\eta(k)$ according to Eq.(2.1):

$$\eta(k) = \begin{cases} \eta_0(1 - k/\tau), & \text{for } k \leq \tau_0 < \tau \\ \eta_1(1 - \tau_0/\tau), & \text{for } k > \tau_0 \end{cases} \quad (2.1)$$

where η_0 and η_1 are constants smaller than 1, and τ and τ_0 time constants. Other types of functions can also be used, for instance,

$$\eta(k) = \eta_1 \cdot e^{-k/\tau} + \eta_0 \quad (2.2)$$

Because the weight updating also includes the neighbors of the winning neuron, the topological neighborhood function h_{j^*} needs to be chosen. We adopt the Gaussian neighborhood function for $h_{j^*}(k)$ [17]:

$$h_{j^*}(k) = \exp\left(-\frac{d_{j,j^*}^2}{2\sigma^2}\right) \quad (2.3)$$

where d_{j,j^*} is a lateral connection distance between neural j and j^* , and σ the width. For the sake of efficiency in computing s_j , not all the accumulated measured radar data will be used to compare with the predicted trajectory. Under such selection of the learning rate and neighborhood function, they will force the minimization of the difference between the weight of the winning neuron and those corresponding to every neuron within its neighborhood in each learning cycle. The learning in the algorithm will thus converge eventually.

Based on the discussions above, we developed the SOM learning algorithm. Before the description of the algorithm, we first introduce a simplified dynamic model of the TBM. With the model, the SOM can obtain $\underline{\mathbf{p}}_j$ by sending $\underline{\mathbf{w}}_j$ into it. This dynamic model is formulated as

$$\underline{\mathbf{x}}(k+1) = \mathbf{A}(k)\underline{\mathbf{x}}(k) + \mathbf{\Gamma}(k)\underline{\boldsymbol{\xi}}(k) \quad (2.4)$$

$$\underline{\mathbf{v}}(k) = \mathbf{C}(k)\underline{\mathbf{x}}(k) + \underline{\boldsymbol{\mu}}(k) \quad (2.5)$$

where

$\underline{\mathbf{x}}(k)$: n -dimensional state vector at the k th stage

$\mathbf{A}(k)$: $n \times n$ transition matrix

$\mathbf{\Gamma}(k)$: $n \times r$ input distribution matrix

$\underline{\boldsymbol{\xi}}(k)$: r -dimensional random input vector

$\underline{\mathbf{v}}(k)$: m -dimensional output vector

$\mathbf{C}(k)$: $m \times n$ observation matrix

$\underline{\boldsymbol{\mu}}(k)$: m -dimensional random disturbance vector

with $\underline{\boldsymbol{\xi}}(k)$ and $\underline{\boldsymbol{\mu}}(k)$ assumed to be white Gaussian with the following properties:

$$E[\underline{\boldsymbol{\xi}}(k)] = 0 \quad (2.6)$$

$$E[\underline{\boldsymbol{\xi}}(j)\underline{\boldsymbol{\xi}}(k)^t] = Q\delta_{jk} \quad (2.7)$$

$$E[\underline{\boldsymbol{\mu}}(k)] = 0 \quad (2.8)$$

$$E[\underline{\boldsymbol{\mu}}(j)\underline{\boldsymbol{\mu}}(k)^t] = R\delta_{jk} \quad (2.9)$$

$$E[\underline{\boldsymbol{\xi}}(j)\underline{\boldsymbol{\mu}}(k)^t] = 0 \quad (2.10)$$

where $E[\cdot]$ stands for the expectation function, Q and R the covariance matrix of the input noise and output noise, respectively, and δ_{jk} the Kronecker delta function. In using the dynamic model, the SOM is not necessarily aware of its statistical properties. By contrast, the Kalman filter needs to know the noise distribution in the dynamic model and also a guess on the system's initial state for trajectory prediction. As the covariance matrices Q and R may be uncertain and varying in noisy, unknown environments, their estimated values are possibly imprecise, even incorrect. Thus, the Kalman filter may not be that effective under such circumstances. The reason that the SOM is more robust to the uncertainty of the dynamic model than the Kalman filter and why it does not require a guess on the initial state may be because it contains a large number of self-organizing neurons in the network. Via learning, these neurons provide many different directions to search for the optimal initial state.

In responding to the three variables, the launching position, velocity, and acceleration of the TBM, a 3D SOM is used for trajectory prediction. The SOM learning algorithm is organized as follows:

SOM Learning Algorithm: Predict an optimal initial state for an incoming TBM in a real-time manner using the measured position radar data.

Step 1: Set the stage of learning $k = 0$. Estimate the ranges of the possible initial position, velocity, and acceleration of the TBM, and randomly store the possible initial states $\underline{\mathbf{w}}_j(0)$ into the neurons, where $j = 1, \dots, N^3$, $N \times N \times N$ the total number of neurons

in the 3D space. Select neuron \hat{j}^* in the center of the neuron space as the winning neuron.

Step 2: Send $\underline{\mathbf{w}}_j(k)$ into the dynamic model, described in Eqs.(2.4)-(2.5), to compute $\underline{\mathbf{p}}_j(k)$.

Step 3: For each neuron j , compute its output s_j as the difference between the measured position data $\underline{\mathbf{v}}(k)$ and $\underline{\mathbf{p}}_j(k)$:

$$s_j(k) = \sum_{i=0}^k \left\| \underline{\mathbf{p}}_j(i) - \underline{\mathbf{v}}(i) \right\| \quad (2.11)$$

Find the winning neuron j^* with the minimum $s_{j^*}(k)$:

$$s_{j^*}(k) = \sum_{i=0}^k \left\| \underline{\mathbf{p}}_{j^*}(i) - \underline{\mathbf{v}}(i) \right\| = \min_j \sum_{i=0}^k \left\| \underline{\mathbf{p}}_j(i) - \underline{\mathbf{v}}(i) \right\| \quad (2.12)$$

Step 4: Update the weights of the previous winning neuron \hat{j}^* and its neighbors within $h_{\hat{j}^*}(k)$ using the following two rules:

$$\text{If } j^* \neq \hat{j}^*, \text{ then } \underline{\mathbf{w}}_j(k+1) = \underline{\mathbf{w}}_j(k) + \eta(k)h_{\hat{j}^*}(k)(\underline{\mathbf{w}}_{j^*}(k) - \underline{\mathbf{w}}_{\hat{j}^*}(k)) \quad (2.13)$$

$$\text{If } j^* = \hat{j}^*, \text{ then } \underline{\mathbf{w}}_j(k+1) = \underline{\mathbf{w}}_j(k) + \eta(k)h_{\hat{j}^*}(k)(\underline{\mathbf{w}}_{j^*}(k) - \underline{\mathbf{w}}_j(k)) \quad (2.14)$$

where $\eta(k)$ is the learning rate described in Eq.(2.1) and $h_{\hat{j}^*}(k)$ the neighborhood function in Eq.(2.3).

Step 5: Check whether the difference between $\underline{\mathbf{w}}_{j^*}(k)$ of the winning neuron j^* and $\underline{\mathbf{w}}_j(k)$ corresponding to every neuron j within $h_{j^*}(k)$ is smaller than a prespecified value ϵ :

$$\max_j \left\| \underline{\mathbf{w}}_j(k) - \underline{\mathbf{w}}_{j^*}(k) \right\| < \epsilon, \quad j \in h_{j^*}(k). \quad (2.15)$$

If Eq.(2.15) does not hold, let $k = k + 1$, and when k is smaller than a prespecified maximum value, go to Step 2; otherwise, the prediction process is completed and output the optimal initial state to the dynamic model to derive the TBM trajectory. Note that the value of ϵ is empirical according to the demanded resolution in learning, and we chose it very close to zero.

2.3 Simulation

To demonstrate the effectiveness of the proposed intelligent radar predictor, we performed a series of simulations based on using both the generated and real radar data. The results were compared with those using the Kalman filtering. Via coordinate transformation, the trajectory of the incoming TBM was described in a 2D ($X \times Y$) space. Because simulation results were similar for the motions in the X and Y directions, we only discussed the motion in the X direction to simplify the illustration. Thus, according to Eq.(2.4), the dynamic model for the TBM is formulated as

$$\underline{\mathbf{x}}(k+1) = \mathbf{A}(k)\underline{\mathbf{x}}(k) + \underline{\boldsymbol{\xi}}(k) \quad (2.16)$$

with

$$\underline{\mathbf{x}}(k) = \begin{bmatrix} x(k) \\ \dot{x}(k) \\ \ddot{x}(k) \end{bmatrix}, \quad \mathbf{A}(k) = \begin{bmatrix} 1 & T & T^2/2 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix}, \quad \underline{\boldsymbol{\xi}}(k) = \begin{bmatrix} 0 \\ 0 \\ \xi(k) \end{bmatrix} \quad (2.17)$$

where $x(k)$, $\dot{x}(k)$, and $\ddot{x}(k)$ stand for the position, velocity, and acceleration of the target

in the X direction, respectively, T the sampling time, and $\xi(k)$ the noise and modeling error that perturbs the target acceleration, with a zero mean and constant variance σ_a^2 .

And, according to Eq.(2.5), the measured radar position $v(k)$ is formulated as

$$v(k) = x(k) + \mu(k) \quad (2.18)$$

where $\mu(k)$ is the measurement noise with a zero mean and constant variance σ_m^2 . The ranges of the possible initial states $\underline{\mathbf{w}}_j(0)$ were predicted to be

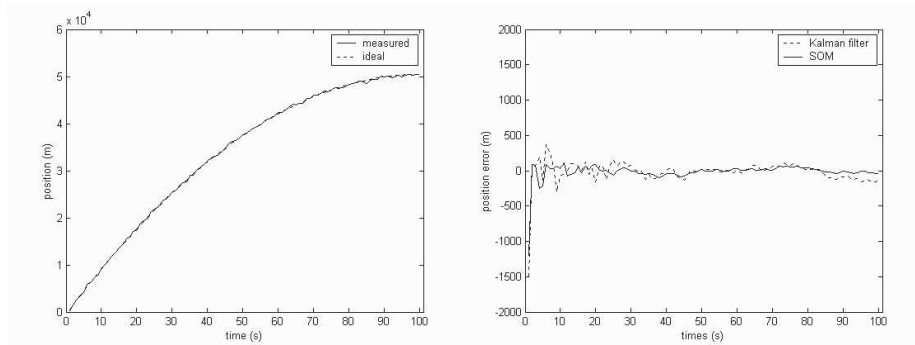
$$\begin{aligned} -1000 \text{ m} &\leq x(0) \leq 1000 \text{ m} \\ -2000 \text{ m/s} &\leq \dot{x}(0) \leq 2000 \text{ m/s} (5.88 \text{ Mach}) \\ -50 \text{ m/s}^2 &\leq \ddot{x}(0) \leq 50 \text{ m/s}^2. \end{aligned} \quad (2.19)$$

Within the ranges described in Eq.(2.19), the possible launching position, velocity, and acceleration of the TBM were selected and stored into the 125 neurons of the 3D SOM. The variances, σ_a^2 and σ_m^2 , described in Eqs.(2.17)-(2.18), are chosen to be $(0.32\text{m/s}^2)^2$ and $(200\text{m})^2$, respectively. The sampling time T was chosen to be 1s. The number of learning is set to be 20 during each stage of learning.

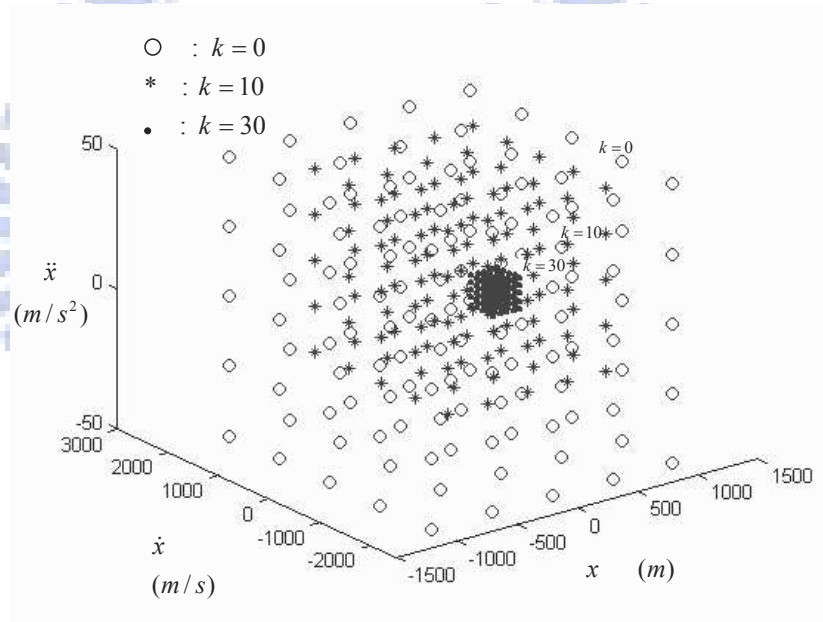
We first applied the SOM and Kalman filter for trajectory prediction under the condition that good estimates of both the initial state and noise distribution were available. The ideal initial state of the target was assumed to be $(500\text{m}, 1000\text{m/s}(2.94\text{Mach}), -10\text{m/s}^2)$, which was within the predicted ranges. And the variance of the measurement noise was set to be the same as the predicted $(200\text{m})^2$. The ideal and measured TBM trajectories were shown in Figure 2.5(a). Both the SOM and Kalman filter predicted the $(k + 1)$ th state quite well, and thus resulted in very small estimated position errors, except in the initial stage of the prediction, as shown in Figure 2.5(b). Figure 2.5(c) shows the movement of the weight vector $\underline{\mathbf{w}}_j$ during the SOM learning process. In Figure 2.5(c), the weight

vectors of the neurons in the SOM continued to move closer and closer during learning, and finally converged to a very small region, since the winning neuron was already within the predicted region from the beginning.

In the second set of simulations, we intended to investigate the performances of the SOM and Kalman filter under the following three situations: (1) good estimate of the initial state, but bad estimate of the noise distribution, (2) bad estimate of the initial state, but good estimate of the noise distribution, and (3) bad estimates of both the initial state and noise distribution. For Case 1, the ideal initial state of the target was still set to be $(500m, 1000m/s(2.94Mach), -10m/s^2)$, but the variance of the measurement noise was enlarged to be $(400m)^2$. The ideal and measured TBM trajectories were shown in Figure 2.6(a). With a bad estimate of the noise distribution, the performance of the Kalman filter degraded, but the SOM still performed well, as shown in Figure 2.6(b). In Figure 2.6(c), the neurons in the SOM exhibited similar behaviors as those shown in Figure 2.5(c). For Case 2, the ideal initial state was assumed to be $(5000m, 3000m/s(8.82Mach), -60m/s^2)$, which was outside the predicted ranges. The variance of the measurement noise was set to be $(200m)^2$. The ideal and measured TBM trajectories were shown in Figure 2.7(a). With a bad estimate of the initial state, the performances of both the SOM and Kalman filter degraded in the initial stage of the prediction, but the SOM achieved better prediction later, as shown in Figure 2.7(b). Correspondingly, the weight vectors of the neurons in the SOM moved from the original predicted region outward to the ideal initial state, and finally converged to the desired location, as shown in Figure 2.7(c). For Case 3, the ideal initial state was assumed to be $(5000m, 3000m/s(8.82Mach), -60m/s^2)$, and the variance of the measurement noise set to be $(400m)^2$. The ideal and measured TBM trajectories were shown in Figure 2.8(a). With bad estimates of both the initial state and noise distribution, the Kalman filter perform poorly, but the SOM still achieved



(a) Ideal and measured TBM trajectories (b) Estimated position error by using the SOM and Kalman filter



(c) Movement of the weight vector $\underline{\mathbf{w}}_j$ during the SOM learning process

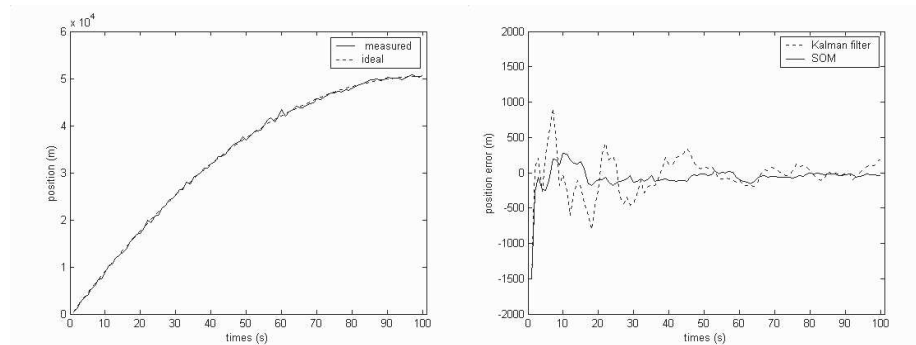
Figure 2.5 Simulation results for trajectory prediction using the SOM and the Kalman filter with good estimates of both the initial condition and noise distribution: (a) the ideal and measured TBM trajectories, (b) the estimated position error by using the SOM and Kalman filter, and (c) the movement of the weight vector $\underline{\mathbf{w}}_j$ during the SOM learning process.

satisfactory performance, as shown in Figure 2.8(b). In Figure 2.8(c), the neurons in the SOM exhibited similar behaviors as those shown in Figure 2.7(c).

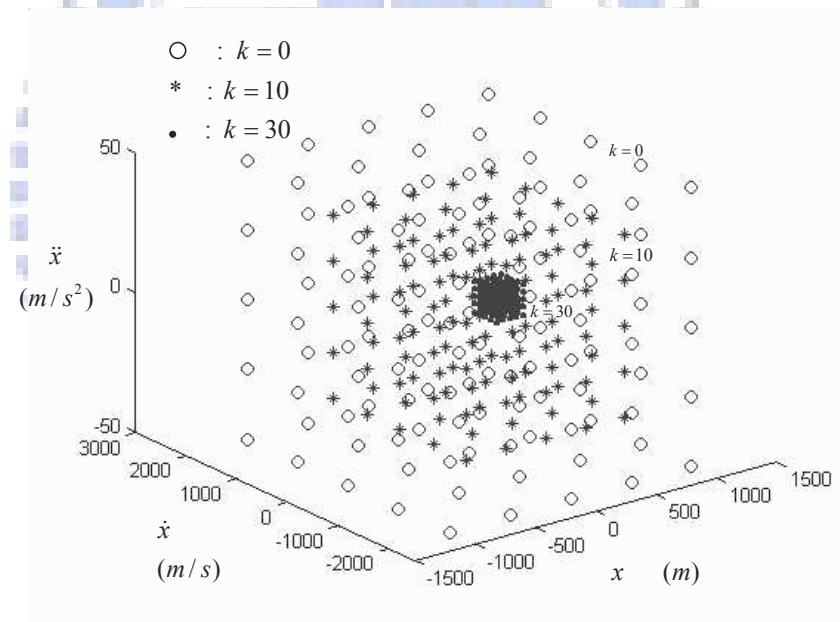
For further investigation, we performed simulations for input noises with the components in both x and y directions and also the condition of a non-zero expectation for these two components. The results show that when the expectation values were small, the intelligent radar predictor still worked quite well. We also performed simulations based on using the genetic algorithm. During the simulations, we first randomly selected the initial populations. When the optimal initial state did not fall within the selected ranges, the GA converged very slowly. We then modified the population size and crossover and mutation probabilities to speed up its convergence rate. However, it was not that straightforward to determine these parameters properly, and the process was time-consuming. From the simulation results, we conclude that the proposed intelligent predictor performed better than the GA for this trajectory tracking problem.

From the results shown in Figures 2.6-2.8, we found that bad estimates of the initial state and noise distribution much affected the performance of the Kalman filter. By contrast, their influence on the SOM was mostly at the initial stage of the prediction. After the transient, the SOM still managed to find the optimal initial state via learning. With its robustness to uncertainty and efficiency in computation, we then used the SOM to predict the TBM trajectory based on using real radar data. The radar data, provided by the military research center, had been modified due to the security consideration. The SOM used only a small number of radar data, marked by the \circ sign in Figure 2.9, to predict the TBM trajectory. In Figure 2.9, the predicted trajectory well approximated the measured one, demonstrating the ability of the SOM to deal with real radar data.

As a summary, in this chapter, we have proposed an intelligent radar predictor for

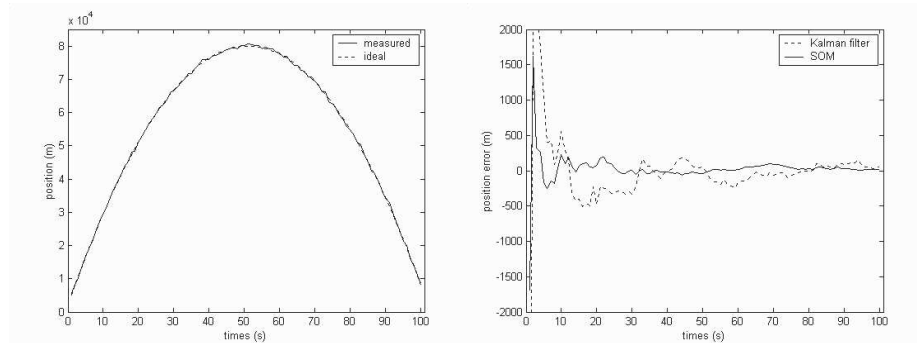


(a) Ideal and measured TBM trajectories (b) Estimated position error by using the SOM and Kalman filter

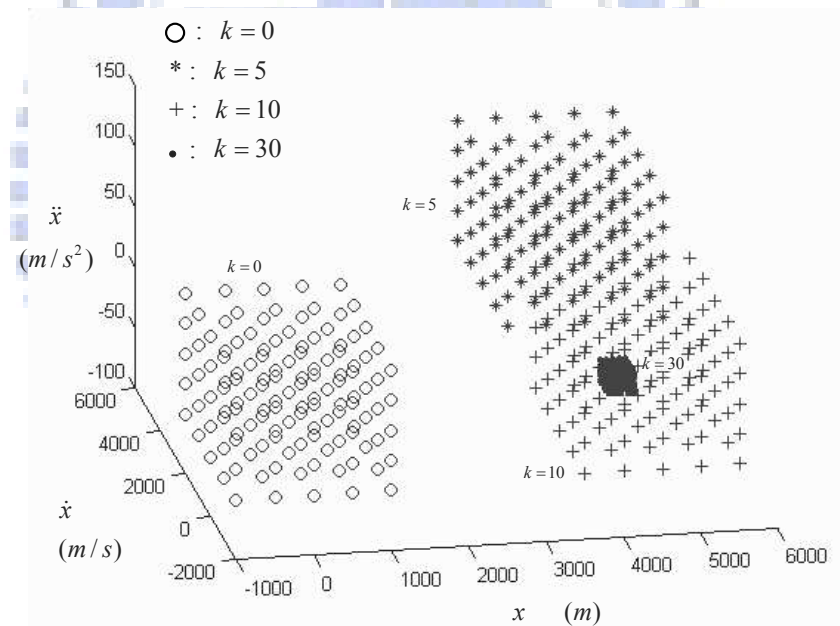


(c) Movement of the weight vector \underline{w}_j during the SOM learning process

Figure 2.6 Simulation results for trajectory prediction using the SOM and the Kalman filter with a good estimate of the initial condition but bad estimate of the noise distribution: (a) the ideal and measured TBM trajectories, (b) the estimated position error by using the SOM and Kalman filter, and (c) the movement of the weight vector \underline{w}_j during the SOM learning process.

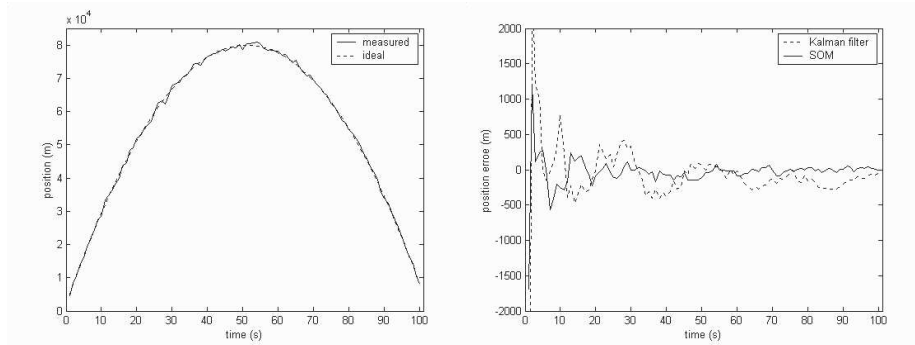


(a) Ideal and measured TBM trajectories (b) Estimated position error by using the SOM and Kalman filter

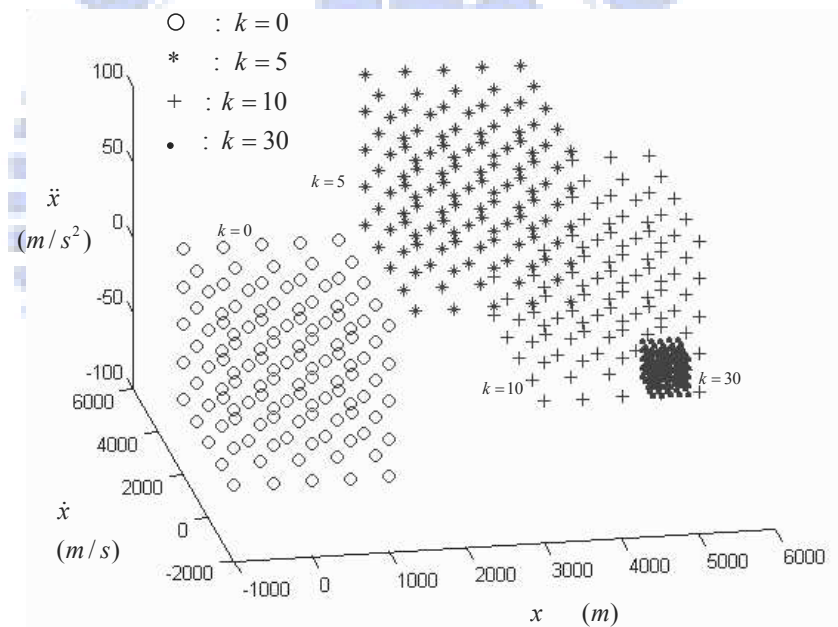


(c) Movement of the weight vector $\underline{\mathbf{w}}_j$ during the SOM learning process

Figure 2.7 Simulation results for trajectory prediction using the SOM and the Kalman filter with a bad estimate of the initial condition but good estimate of the noise distribution: (a) the ideal and measured TBM trajectories, (b) the estimated position error by using the SOM and Kalman filter, and (c) the movement of the weight vector $\underline{\mathbf{w}}_j$ during the SOM learning process.



(a) Ideal and measured TBM trajectories (b) Estimated position error by using the SOM and Kalman filter



(c) Movement of the weight vector \underline{w}_j during the SOM learning process

Figure 2.8 Simulation results for trajectory prediction using the SOM and the Kalman filter with bad estimates of both the initial condition and the noise distribution: (a) the ideal and measured TBM trajectories, (b) the estimated position error by using the SOM and Kalman filter, and (c) the movement of the weight vector \underline{w}_j during the SOM learning process.

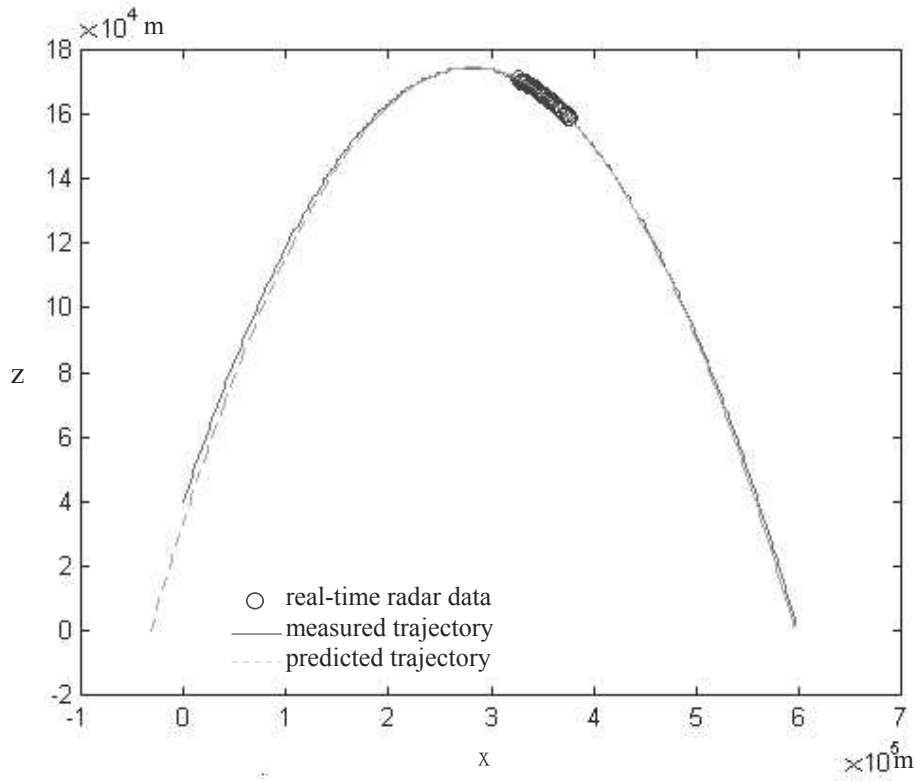
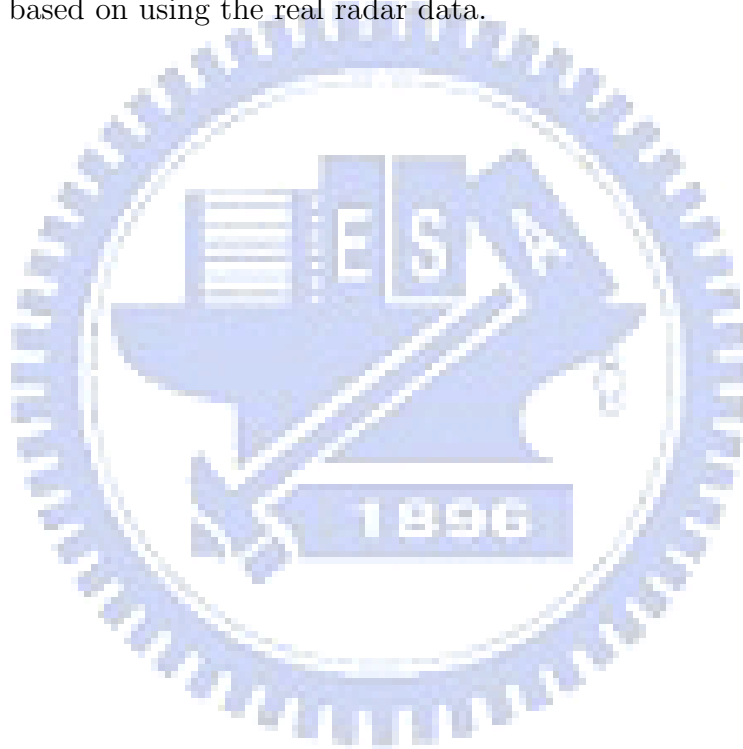


Figure 2.9 Results by applying the SOM to predict the actual TBM trajectory.

trajectory estimation. With a simplified target dynamic model, the unsupervised SOM in the predictor can achieve salient prediction in the presence of noise, even with a bad estimate of the initial state. The performance of the SOM has been compared with mainly that of the Kalman filtering. The simplified TBM dynamic model used in the current stage of the study may account for only the coarse behavior of the TBM. Nevertheless, even with only the general information provided by this simplified model, the proposed intelligent radar predictor has been able to catch up with the TBM, as demonstrated in the simulations based on using the real radar data.



Chapter 3

SOM-Based Algorithm for Optimization

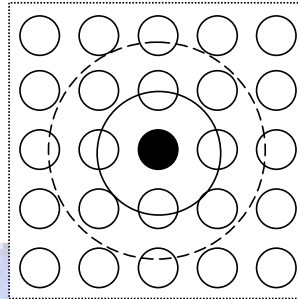
Although the SOM has been widely used in many diverse tasks, few studies are available for applying the SOM as a search mechanism. Recently, some researchers have exploited its ability in search [7, 29, 40]. Michele et al. proposed a learning algorithm for optimization based on the Kohonen SOM evolution strategy (KSOM-ES) [29]. In this KSOM-ES algorithm, the adaptive grids are used to identify and exploit search space regions that maximize the probability of generating points closer to the optima. Su et al. proposed an SOM-based optimization algorithm (SOMO) [40]. Through the self-organizing process in SOMO, solutions to a continuous optimization problem can be simultaneously explored and exploited. The point about applying the SOM as a search mechanism is that each weight vector represents a possible solution of the objective function. Through the fitness function the winner will be determined with the largest fitness and updating the weights of the winner and its neighbors, all the weights will be moved to explore and exploit the optimization space for the searching process. A major drawback is that the SOMO and

KSOM-ES converge very slowly if the optimal solution falls outside the estimated range. Because of the influence of noise the search direction is not correct probably. Meanwhile the step size is reduced continually. Eventually the optimal solution is probably outside the search range. In the dynamic optimization the search direction and step size are hard to determine effectively in noisy and unknown environment. Thus, in this chapter a new SOM weight updating rule based on a heuristic techniques is proposed to deals properly with these problems and enhance the learning efficiency; this may dynamically adjust the neighborhood function for the SOM in learning system parameters, discussed in next Section.

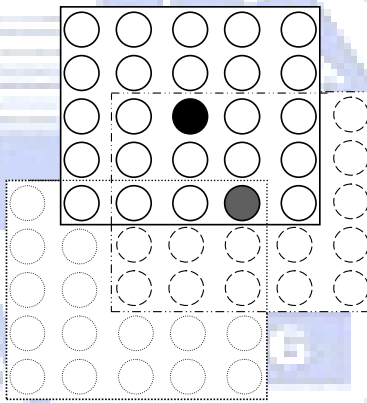
3.1 Proposed SOM-Based Search Algorithm

Figure 3.1 shows the conceptual diagram of the organized search in a 2-D SOM. Note that Figure 3.1 is slight different from Figure 2.4 in that the weight of the previous winner replaced by the average of all weights. Figure 3.1(a) shows a case where the solution is within the estimated range. In this case, the weights of the neurons are updated so as to make the weight vectors converge to a compact cluster centering at the optimal solution. Figure 3.1 (b) shows the case where the solution is outside the estimated range. The winner will be located at the corner of the SOM initially. During the next learning epoch, it is moved to the center of the SOM. The learning will continue until the solution falls within the new estimated range. The search will then follow the process shown in Figure 3.1(a) to converge to the optimal solution.

Figure 3.2 shows the proposed SOMS, which consists of mainly the evaluation and search mechanisms and the dynamic model stands for the target system. Initially, the



(a) Within the estimated range



(b) Outside of the estimated range

Figure 3.1 Conceptual diagram of the organized search in a 2-D SOM: the solution is (a) within the estimated range and (b) outside of the estimated range.

function for performance evaluation is installed in the evaluation mechanism, and possible solutions (e.g., vectors of dynamic parameters), selected from the estimated range, will be distributed among the neurons of the SOM. During each time interval of the learning process, each of all the possible solutions in the neurons is sent to the dynamic model one by one. In other words, the dynamic model will be equipped with a possible set of dynamic parameters repeatedly, when used to derive the output data corresponding to the target system. The evaluation mechanism will then compute the difference between the derived data and the incoming measured data. From the results, the search mechanism chooses the solution leading to the most accurate derived data as the winner, and updates the weights of this winner and its neighboring neurons. Note that this SOMS can also be applied to continuous optimization problems, with the dynamic model replaced by the objective function for a given optimization problem and the input by the reference data.

3.2 Proposed Weight Updating Rule

Figure 3.3 shows the structure and operation of the SOM in the SOMS. The SOM performs two operations: evaluation and search. In Figure 3.3, each neuron j in the SOM contains a vector of a possible solution set $\underline{\mathbf{w}}_j$ (the weight vector). Each time new measured data $\underline{\mathbf{v}}$ are sent into the scheme, the SOM is triggered to operate. All of the possible solution sets in the neurons will then be sent to the dynamic model to derive their corresponding data $\underline{\mathbf{p}}_j$. The SOM evaluates the difference between $\underline{\mathbf{v}}$ and each $\underline{\mathbf{p}}_j$. Of all the neurons, it chooses the neuron j^* , which corresponds to the smallest difference, as the winner. The learning process then continues, and the network will eventually converge to the optimal solution. And even when the optimal solution is not within the estimated range for some cases, the search mechanism is still expected to move the possible candidates out of their

initial locations and guide them to converge to the optimal solution.

The main purpose of the proposed SOMS is how to explore and exploit the search space and to obtain an optimal solution for the optimization problem and, furthermore, to make the variations of the weights as organized movements. To this purpose, the SOMS learns to organized and efficient search, but not random search. For effective weight updating in the SOM, the topological neighborhood function and learning rate need to be properly determined. Their determination may depend on the properties of the system parameters to learn. As mentioned above, system parameters may operate in quite different working ranges. To achieve high learning efficiency, the weight updating should be executed on an individual basis, instead of using the same neighborhood function for all the parameters. We thus propose a new SOM weight updating rule which can dynamically adjust the center and width of their respective neighborhood function for the SOM in learning each of the system parameters.

For the topological mapping, unlike in the traditional SOM applications, it is now our aim to let the weight vectors form the uniform distribution like the pre-ordered lattice in the neuron space. Generally the neuron space and weight vector space are with different dimensions, so we have to transfer them into the same one. The Gaussian type function is usually used as the neighborhood function, and it is differentiable and continuous. We also used the Gaussian type functions as the neighborhood functions in the neuron space and weight vector space. With the neighborhood functions, the magnitudes of their respective distances in lattice space and in weight vector space can be normalized to be between 0 and 1. The proposed weight updating rule is designed to first let the weight vectors approach the vicinity of the optimal solution set when it falls outside the coverage of the SOM. The weight vector cluster is then moved to the center of the SOM. The process will

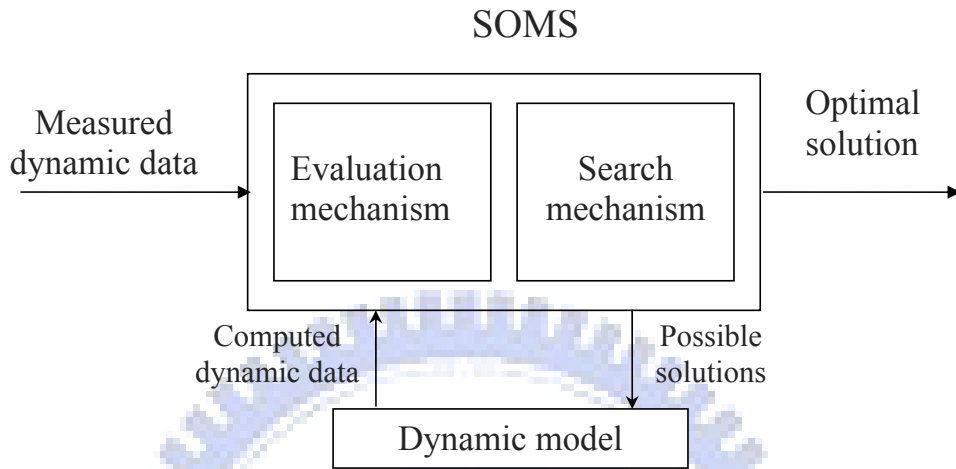


Figure 3.2 Proposed SOM-based algorithm for optimization.

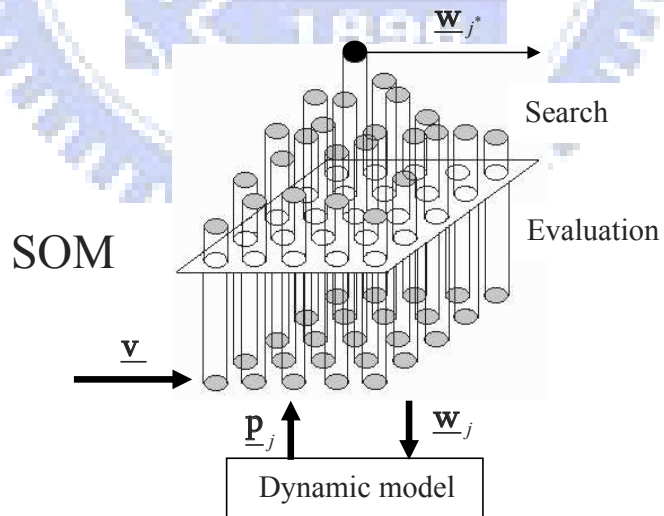


Figure 3.3 Structure and operation of the SOM in the SOMS.

continue until the solution set falls within the SOM. Later, the rule will make the weight vectors converge to a more and more compact cluster centering at the optimal solution. We then define two Gaussian neighborhood functions, D_j and $F(\underline{\mathbf{w}}_j(k))$ in the k th stage of learning as

$$D_j = \exp\left(-\frac{\|\underline{\mathbf{r}}_j - \underline{\mathbf{r}}_{j^*}\|^2}{2\sigma_d^2}\right) \quad (3.1)$$

$$F(\underline{\mathbf{w}}_j(k)) = \exp\left(\frac{-1}{q} \sum_{i=1}^q \frac{(w_{j,i}(k) - w_{j^*,i}(k))^2}{2\sigma_i^2}\right) \quad (3.2)$$

where $\underline{\mathbf{r}}_j$ and $\underline{\mathbf{r}}_{j^*}$ stand for the coordinates of neuron j to entire network and j^* , respectively, σ_d the standard deviation of the distribution for D_j , and σ_i the standard deviation of the distribution for $w_{j,i}(k)$. Note that $F(\underline{\mathbf{w}}_j(k))$ is defined by considering the effects from all q elements in $\underline{\mathbf{w}}_j(k)$. Here, D_j is used as a reference distribution for $F(\underline{\mathbf{w}}_j(k))$. In other words, We intend to map the magnitude difference of the parameter into the neurons spaces. To make $F(\underline{\mathbf{w}}_j(k))$ approach D_j , an error function $E_j(k)$ is then defined as

$$E_j(k) = \frac{1}{2}(D_j - F(\underline{\mathbf{w}}_j(k)))^2. \quad (3.3)$$

During the learning, we can find that when $w_{j^*,i}(k)$ is much different from $w_{e_i}(k)$, the average of all $w_{j,i}(k)$, the optimal solution is possibly located far outside the estimated range; contrarily, when $w_{j^*,i}(k)$ is close to $w_{e_i}(k)$, the optimal solution is possibly within the estimated range. Based on this observation, we proposed a method to speed up the learning. For illustration, we define a Gaussian distribution function $G(w_{j,i}(k))$ for each element $w_{j,i}(k)$, i th element in $\underline{\mathbf{w}}_j(k)$ in the k th stage of learning:

$$G(w_{j,i}(k)) = \exp\left(-\frac{(w_{j,i}(k) - w_{e_i}(k))^2}{2\sigma_i^2}\right) \quad (3.4)$$

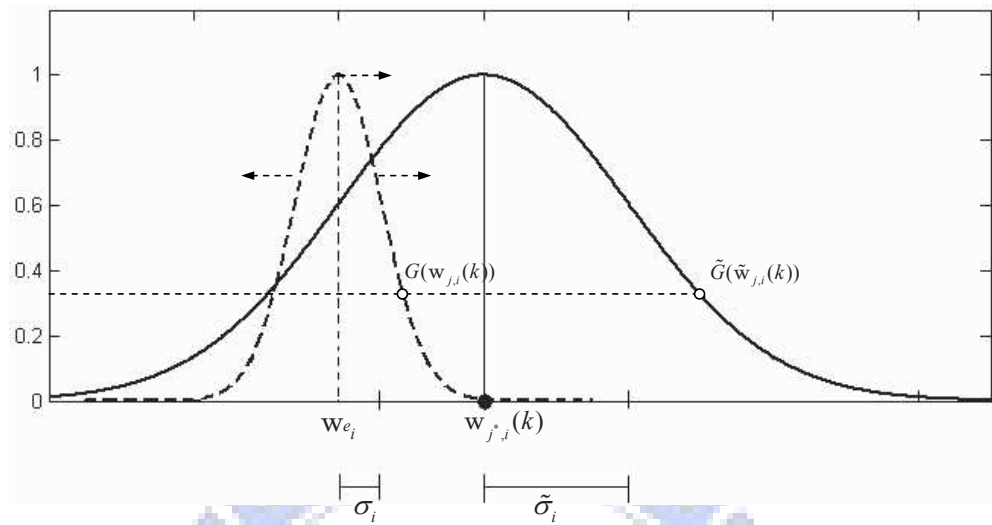
The strategy is to vary the mean and variance of $G(w_{j,i}(k))$ by moving its center to where $w_{j^*,i}(k)$ is located and enlarging (reducing) the variance σ_i^2 to be $\tilde{\sigma}_i^2 = |w_{j^*,i}(k) - w_{e_i}(k)|^2$, where $|\cdot|$ stands for the absolute value, as illustrated in Figure 3.4. The new distribution function $\tilde{G}(\tilde{w}_{j,i}(k))$ is then formulated as

$$\tilde{G}(\tilde{w}_{j,i}(k)) = \exp\left(-\frac{(\tilde{w}_{j,i}(k) - w_{j^*,i}(k))^2}{2\tilde{\sigma}_i^2}\right) = \exp\left(-\frac{(w_{j,i}(k) - w_{e_i}(k))^2}{2\sigma_i^2}\right) = G(w_{j,i}(k)) \quad (3.5)$$

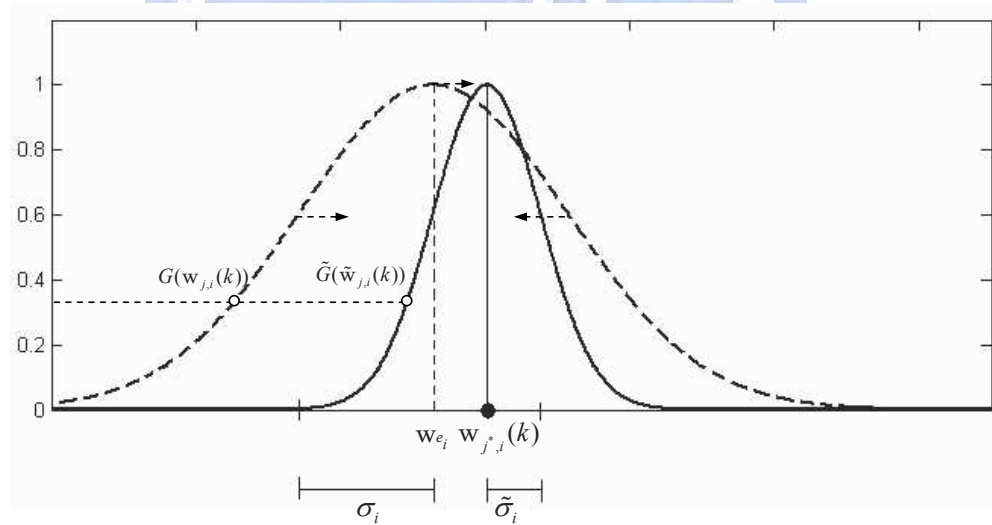
where $\tilde{w}_{j,i}(k)$ stands for the new $w_{j,i}(k)$ after the adjustment. As indicated in Figure 3.4, $\tilde{G}(\tilde{w}_{j,i}(k))$ is equal to $G(w_{j,i}(k))$ when $w_{j,i}(k)$ varies to $\tilde{w}_{j,i}(k)$. From Eq.(3.5), during each iteration of learning, $G(w_{j,i}(k))$ is dynamically centered at the location of the winning neuron j^* , with a larger (smaller) width when $w_{e_i}(k)$ is much (less) different from $w_{j^*,i}(k)$. It thus covers a more fitting neighborhood region, and leads to a higher learning efficiency. With $\tilde{G}(\tilde{w}_{j,i}(k))$, the new weight $\tilde{w}_{j,i}(k)$ is derived as

$$\tilde{w}_{j,i}(k) = \frac{|w_{j^*,i}(k) - w_{e_i}(k)|}{\sigma_i} \cdot (w_{j,i}(k) - w_{e_i}(k)) + w_{j^*,i}(k). \quad (3.6)$$

And, with a desired new weight $\tilde{w}_{j,i}(k)$, the learning should also make $\underline{\mathbf{w}}_j(k)$ approach $\tilde{\underline{\mathbf{w}}}_j(k)$, in addition to minimizing the error function $E_j(k)$ in Eq.(3.3). A new error function $\tilde{E}_j(k)$ is thus defined as



$$(a) \left(w_{e_i} - w_{j,i}^*(k) \right)^2 \geq \sigma_i^2$$



$$(b) \left(w_{e_i} - w_{j,i}^*(k) \right)^2 < \sigma_i^2$$

Figure 3.4 Center and width adjustment for the neighborhood function $G(w_{j,i}(k))$, when (a) $\left(w_{e_i} - w_{j,i}^*(k) \right)^2 \geq \sigma_i^2$ and (b) $\left(w_{e_i} - w_{j,i}^*(k) \right)^2 < \sigma_i^2$.

$$\tilde{E}_j(k) = \frac{1}{2}[(D_j - F(\mathbf{w}_j(k)))^2 + \|\mathbf{w}_j(k) - \tilde{\mathbf{w}}_j(k)\|^2]. \quad (3.7)$$

Based on Eq.(3.7) and the gradient-descent approach, the weight-updating rule is derived as

$$\begin{aligned} w_{j,i}(k+1) &= w_{j,i}(k) - \eta(k) \frac{\partial \tilde{E}_j(k)}{\partial w_{j,i}(k)} \\ &= w_{j,i}(k) - \eta(k) \left[\frac{\partial E_j(k)}{\partial F(\mathbf{w}_j(k))} \cdot \frac{\partial F(\mathbf{w}_j(k))}{\partial w_{j,i}(k)} + (w_{j,i}(k) - \tilde{w}_{j,i}(k)) \right] \\ &= w_{j,i}(k) - \eta(k) \left[\frac{(w_{j,i}(k) - w_{j^*,i}(k))}{q \cdot \sigma_i^2} \cdot F(\mathbf{w}_j(k)) \cdot (D_j - F(\mathbf{w}_j(k))) \right. \\ &\quad \left. + (w_{j,i}(k) - \tilde{w}_{j,i}(k)) \right] \end{aligned} \quad (3.8)$$

where $\eta(k)$ stands for the learning rate in the k th stage of learning, described in chapter 1. Together, the weight updating rule described in Eq.(3.8) and the learning rate in Eq.(2.2) will force the minimization of the difference between the weight vector of the winning neuron and those corresponding to every neuron in each learning cycle. The learning will eventually converge.

3.3 Applications

To demonstrate its capability, the SOMS is applied for both function optimization and dynamic trajectory prediction. Based on the SOMS, we first develop learning schemes corresponding to each of the applications. Simulations are then executed for performance evaluation. The results are especially compared with those of the genetic algorithm (GA) for their resemblance in searching. Both the SOM and GA have the merit of parallel processing. And, both of their searches are through the guidance of the evaluation function, while the SOM in our design adopts a somewhat organized search and the GA in

some sense a random approach. It implicates that the SOM may be more suitable for applications with certain knowledge, especially when the distribution of the possible solutions is not utterly random. On the contrary, for applications with no a priori knowledge available, the GA may yield better performance.

3.3.1 Function Optimization

For a function optimization problem, the goal may be to maximize (minimize) an object function $O(\cdot)$. Let $O(\underline{\mathbf{w}}_j(k))$ be the function value for the weight vector $\underline{\mathbf{w}}_j(k)$, which represents a possible solution. The learning algorithm for function optimization is organized as follows.

Algorithm for function optimization based on the SOMS: Maximize (minimize) an object function using the SOMS.

Step 1: Set the stage of learning $k = 0$. Choose a reference value P_r . Estimate the ranges of the possible parameter space and randomly store the possible parameters $\underline{\mathbf{w}}_j(0)$ into the neurons, where $j = 1, \dots, N \times N$, $N \times N$ the total number of neurons in the 2D ($N \times N$) space.

Step 2: Compute $O(\underline{\mathbf{w}}_j(k))$ for all $\underline{\mathbf{w}}_j(k)$.

Step 3: Among the neurons, find the one with the largest (smallest) value as the winning neuron j^* for the maximization (minimization) problem.

Step 4: Update the weight vectors of the winning neuron j^* and its neighbors according to the weight updating rule described in Eq.(3.8).

Step 5: Check whether the difference between $\underline{\mathbf{w}}_{j^*}(k)$ of the winning neuron j^* and $\underline{\mathbf{w}}_j(k)$ corresponding to every neuron j is smaller than a prespecified value P_r . If it is not, let $k = k + 1$, and when k is smaller than a prespecified maximum value, go to Step 2; otherwise, the learning process is completed and output the optimal value.

Two standard test functions are used to demonstrate the proposed algorithm, a 2-D Griewant function

$$f(x_1, x_2) = 1 + \frac{1}{4000} [(x_1 - 100)^2 + (x_2 - 100)^2] - \cos(x_1 - 100) \cdot \cos\left(\frac{x_2 - 100}{\sqrt{2}}\right) \quad (3.9)$$

and a 30-D Rosenbrock function

$$f(x) = \sum_{i=1}^{29} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]. \quad (3.10)$$

These two test functions have also been used in [40]. The optimization here is to minimize these two functions. Their global minimal values are known in advance: for the Griewant function, it is 0 when $(x_1, x_2) = (100, 100)$; for the Rosenbrock function, it is also 0 when all x_i are equal to 1. The SOM is chosen to be with 5×5 neurons and the learning rate as

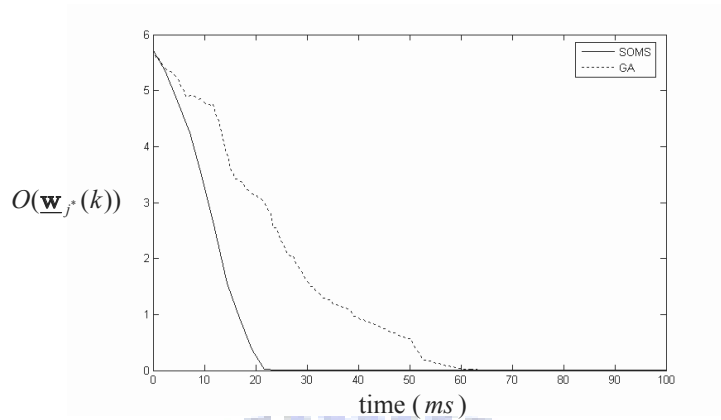
$$\eta(k) = 0.7 \cdot e^{-k/50} + 0.2 \quad (3.11)$$

For comparison, we also use the GA for function minimization, which is with a population size of 25 to match with that of the SOM, and the crossover and mutation probability of 0.6 and 0.0333, respectively.

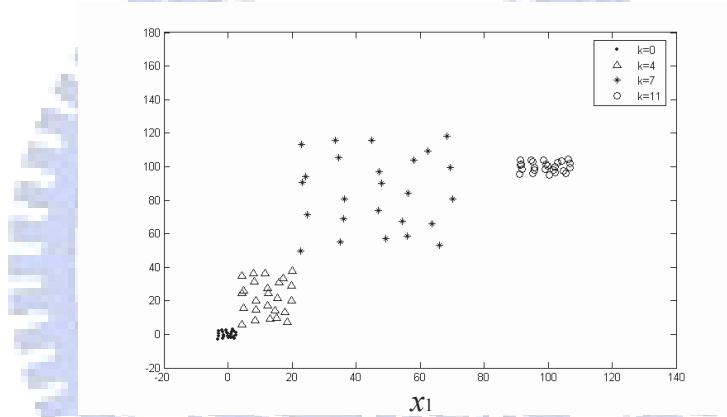
We start with the learning for the 2-D Griewant function. The initial $\underline{\mathbf{w}}_j(0)$ for the SOMS was randomly chosen within the ranges of $(-3, 3) \times (-3, 3)$, i.e., the optimal solution was outside of the estimated region. Figure 3.5 shows the simulation results. In Figure 3.5(a), both SOMS and GA found the optimal minimal value successfully, while the SOMS converged faster. Figures 3.5(b) and (c) show the weight vector movement ($k = 0 \sim 11$)

for the SOMS and GA, respectively. From the figures, we observed that the search in the SOMS was basically in grouping and more directional; by contrast, that of the GA was in a more random manner. It indicates that the SOMS was more effective for this 2-D Griewant function minimization, because the distribution of the possible solutions might not be utterly random.

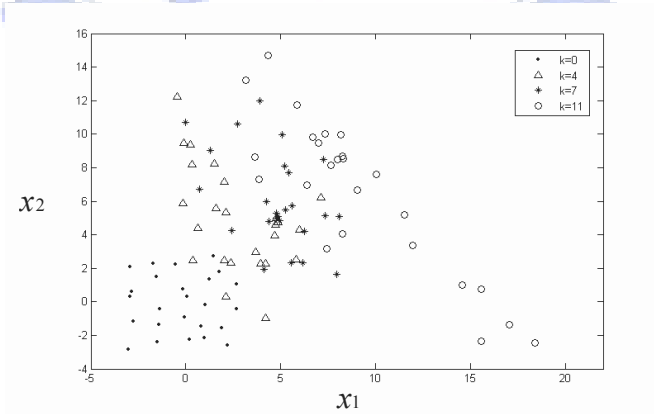
In the minimization of the 30-D Rosenbrock function, we simulated the case that the optimal solution was within the estimated region. For its complexity, the size of the SOM was enlarged to be with 25×25 neurons, and that of the GA also enlarged accordingly. The learning rate for the SOMS and the crossover and mutation probabilities for the GA were set to be the same. Each $w_{j,i}(0)$ of the initial $\mathbf{w}_j(0)$ was randomly chosen within the range of $(-5, 5)$. In addition to the SOMS and GA, the SOMO proposed in [40] was also applied for the minimization, with its parameters adjusted via a trial-and-error process to yield salient performance. Figure 3.6 shows the simulation results. In Figure 3.6, all SOMS, GA, and SOMO found the optimal minimal value successfully, while the SOMS still converged faster. It indicates that the SOMS was also effective for the 30-D Rosenbrock function minimization.



(a) Minimal function values $O(\underline{\mathbf{w}}_j^*(k))$ during the learning process



(b) Weight vector movement in the SOM



(c) Weight vector movement in the GA

Figure 3.5 Minimization of the 2-D Griewant function using the SOMS and GA with the optimal solution outside of the estimated region: (a) minimal function values $O(\underline{\mathbf{w}}_j^*(k))$ during the learning process, (b) weight vector movement in the SOM, and (c) weight vector movement in the GA.

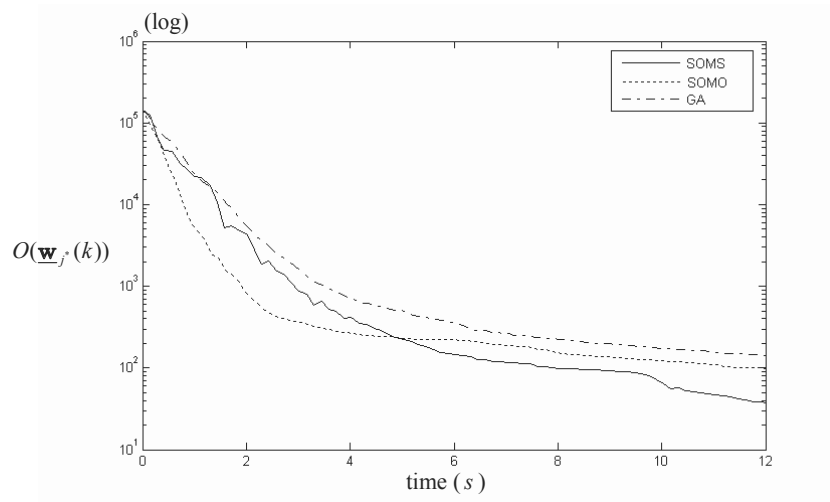
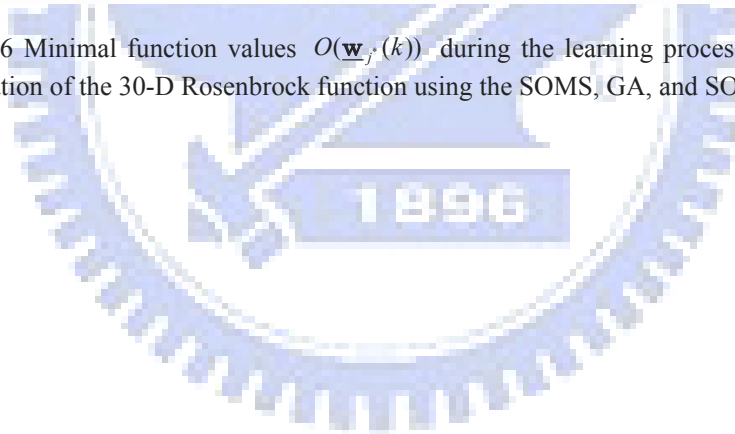


Figure 3.6 Minimal function values $O(\underline{w}_j(k))$ during the learning process for the minimization of the 30-D Rosenbrock function using the SOMS, GA, and SOMO.



3.3.2 Dynamic Trajectory Prediction

For a dynamic trajectory prediction problem, the goal may be to estimate the launching position and velocity of a moving object using the measured data. Through a learning process, the SOMS may determine a most probable initial state through repeatedly comparing the measured data with the predicted trajectories derived from the possible initial states stored in the neurons of the SOM. We consider the SOMS very suitable for this application, because the relationship between the initial state and its resultant trajectory is not utterly random. We can thus distribute the initial states into the SOM in an organized fashion, and make it as a guided search.

In this application, the nonlinear dynamic equation describing the trajectory of the moving object and the measurement equation are first formulated as

$$\underline{\mathbf{x}}(k+1) = f_k(\underline{\mathbf{x}}(k)) + \underline{\boldsymbol{\xi}}_k \quad (3.12)$$

$$\underline{\mathbf{v}}(k) = g_k(\underline{\mathbf{x}}(k)) + \underline{\boldsymbol{\zeta}}_k \quad (3.13)$$

where f_k and g_k are the vector-value function defined in R^q and R^l (q and l the dimension), respectively, and their first-order partial derivatives with respect to all the elements of $\underline{\mathbf{x}}(k)$ continuous. $\underline{\boldsymbol{\xi}}_k$ and $\underline{\boldsymbol{\zeta}}_k$ are the zero-mean Gaussian white noise sequence in R^q and R^l , respectively, with

$$E[\underline{\boldsymbol{\xi}}_k] = 0 \quad (3.14)$$

$$E[\underline{\boldsymbol{\xi}}_j \underline{\boldsymbol{\xi}}_k^t] = Q \delta_{jk} \quad (3.15)$$

$$E[\underline{\boldsymbol{\zeta}}_k] = 0 \quad (3.16)$$

$$E[\underline{\boldsymbol{\zeta}}_j \underline{\boldsymbol{\zeta}}_k^t] = U \delta_{jk} \quad (3.17)$$

$$E[\underline{\boldsymbol{\xi}}_j \underline{\boldsymbol{\zeta}}_k^t] = 0 \quad (3.18)$$

where $E[\cdot]$ stands for the expectation function, Q and U the covariance matrix of the input noise and output noise, respectively, and δ_{jk} the Dirac delta function. Q and U are expected to be uncertain and varying in noisy, unknown environments, and their estimated values possibly imprecise, even incorrect. Being unaware of the statistical properties of the dynamic model, the SOMS is utilized to find the optimal initial state via learning. The learning algorithm for dynamic trajectory prediction is organized as follows.

Algorithm for dynamic trajectory prediction based on the SOMS: Predict an optimal initial state for the trajectory of a moving object using the measured position data.

Step 1: Set the stage of learning $k = 0$. Estimate the ranges of the possible launching position and velocity of the moving object, and randomly store the possible initial states $\underline{\mathbf{w}}_j(0)$ into the neurons, where $j = 1, \dots, m \times n$, $m \times n$ the total number of neurons in the 2D ($m \times n$) space.

Step 2: Send $\underline{\mathbf{w}}_j(k)$ into the dynamic model, described in Eq.(3.12) and Eq.(3.13), to compute $\underline{\mathbf{p}}_j(k)$.

Step 3: For each neuron j , compute its output $O_j(k)$ as the Euclidean distance between the measured position data $\underline{\mathbf{v}}(k)$ and $\underline{\mathbf{p}}_j(k)$:

$$O_j(k) = \sum_{i=0}^k \left\| \underline{\mathbf{p}}_j(i) - \underline{\mathbf{v}}(i) \right\| \quad (3.19)$$

Find the winning neuron j^* with the minimum $O_{j^*}(k)$:

$$O_{j^*}(k) = \sum_{i=0}^k \left\| \mathbf{p}_{j^*}(i) - \mathbf{v}(i) \right\| = \min_j \sum_{i=0}^k \left\| \mathbf{p}_j(i) - \mathbf{v}(i) \right\| \quad (3.20)$$

Step 4: Update the weight vectors of the winning neuron j^* and its neighbors.

Step 5: Check whether the minimum $O_{j^*}(k)$ is smaller than a pre-specified value ϵ :

$$O_{j^*}(k) < \epsilon \quad (3.21)$$

If Eq.(3.21) does not hold, let $k = k + 1$ and go to Step 2; otherwise, the prediction process is completed and output the predicted optimal initial state to the dynamic model to derive the object trajectory. Note that the value of ϵ is empirical according to the demanded resolution in learning, and we chose it very close to zero. In addition, during each stage of learning, we perform a number of learning to increase the SOM learning speed. This number of learning is set to be a large number in the initial stage of the learning process, such that the SOMS may converge faster at the price of more oscillations, and decreased gradually to achieve smooth learning in the later stages of learning.

To demonstrate the effectiveness of the proposed SOMS and weight updating rule, we performed a series of simulations for dynamic trajectory prediction based on using the SOMS, the SOMS without the proposed center and width adjustment on the neighborhood function (named as SOMSO), and GA. The trajectory to predict in the simulations was designed to emulate that of a missile. Its governing equations of motion in the 3D Cartesian coordinate system are described as

$$\ddot{x} = \frac{-g_m x}{(x^2 + y^2 + z^2)^{3/2}} + 2\omega \dot{y} + \omega^2 x + \xi_x \quad (3.22)$$

$$\ddot{y} = \frac{-g_m y}{(x^2 + y^2 + z^2)^{3/2}} + 2\omega \dot{x} + \omega^2 y + \xi_y \quad (3.23)$$

$$\ddot{z} = \frac{-g_m z}{(x^2 + y^2 + z^2)^{3/2}} + \xi_z \quad (3.24)$$

where g_m and ω stand for the gravitational constant and the rotative velocity of the earth, respectively, and set to be $g_m = 3.986 \times 10^5 km^3/s^2$ and $\omega = 7.2722 \times 10^{-5} rad/s$. (ξ_x, ξ_y, ξ_z) are assumed to be continuous-time uncorrelated zero-mean Gaussian white noise processes. Referring to Eq.(3.12) and letting $\underline{\mathbf{x}} = (x, y, z, \dot{x}, \dot{y}, \dot{z})^T = (x_1, x_2, x_3, x_4, x_5, x_6)^T$, we can obtain the discretized dynamic equation as

$$\underline{\mathbf{x}}(k+1) = f(\underline{\mathbf{x}}(k)) + \underline{\boldsymbol{\xi}}_k \quad (3.25)$$

where

$$f(\underline{\mathbf{x}}(k)) = \begin{bmatrix} x_1(k) + tx_4(k) \\ x_2(k) + tx_5(k) \\ x_3(k) + tx_6(k) \\ x_4(k) - tg_m x_1(k)/(x_1(k)^2 + x_2(k)^2 + x_3(k)^2)^{3/2} + 2t\omega x_5(k) + t\omega^2 x_1(k) \\ x_5(k) - tg_m x_2(k)/(x_1(k)^2 + x_2(k)^2 + x_3(k)^2)^{3/2} + 2t\omega x_4(k) + t\omega^2 x_2(k) \\ x_6(k) - tg_m x_3(k)/(x_1(k)^2 + x_2(k)^2 + x_3(k)^2)^{3/2} \end{bmatrix} \quad (3.26)$$

and

$$\underline{\boldsymbol{\xi}}_k = [0 \ 0 \ 0 \ \xi_{x_4} \ \xi_{x_5} \ \xi_{x_6}]^T \quad (3.27)$$

with t the sampling time. $(\xi_{x_4}, \xi_{x_5}, \xi_{x_6})$ are assumed to be uncorrelated zero-mean Gaussian white noise sequences with a constant variance $\sigma_f^2 = (0.1m/s^2)^2$. And, referring to Eq.(3.13), the measurement equation is formulated as

$$\underline{\mathbf{v}}(k) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \underline{\mathbf{x}}(k) + \underline{\boldsymbol{\zeta}}_k \quad (3.28)$$

and

$$\underline{\zeta}_k = \left[\zeta_{x_1} \zeta_{x_2} \zeta_{x_3} \right]^T \quad (3.29)$$

where $(\zeta_{x_1}, \zeta_{x_2}, \zeta_{x_3})$ are the measurement noise sequences with a zero mean and constant variance $\sigma_m^2 = (15m)^2$. The ranges of the possible initial states $\underline{\mathbf{w}}_j(0)$ were estimated to be

$$\begin{aligned} 68.6 \times 10^5 m &\leq x_1(0) \leq 68.8 \times 10^5 m \\ 2.7 \times 10^5 m &\leq x_2(0) \leq 2.8 \times 10^5 m \\ 4.8 \times 10^5 m &\leq x_3(0) \leq 4.9 \times 10^5 m \\ 110 m/s &\leq x_4(0) \leq 150 m/s \\ 810 m/s &\leq x_5(0) \leq 850 m/s \\ 1360 m/s &\leq x_6(0) \leq 1380 m/s. \end{aligned} \quad (3.30)$$

Within the ranges described in Eq.(3.30), the possible launching positions and velocities of the missile were selected and stored into the 729 (27×27) neurons of the 2D SOM. And, the learning rate for the SOMS was chosen to be

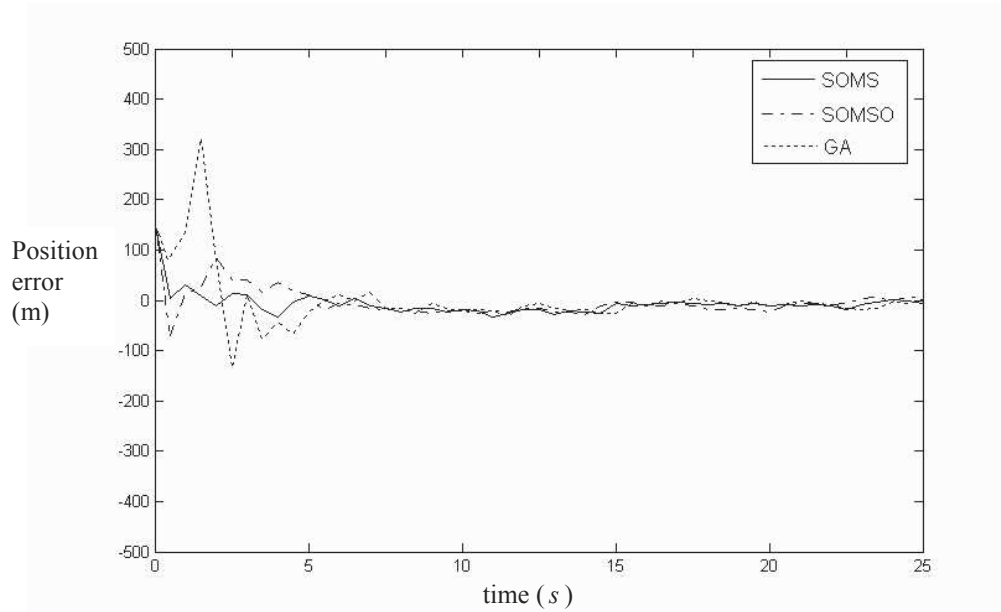
$$\eta(k) = 0.8 \cdot e^{-k/50} + 0.2 \quad (3.31)$$

The sampling time t was $0.5s$. For the GA, the population size was selected to be 729 to match with the SOM, and the crossover and mutation probability 0.6 and 0.0333, respectively. The number of learning is set to be 20 during each stage of learning.

We first applied the SOMS, SOMSO and GA for trajectory prediction with a good estimate of the initial state. The ideal initial state of the missile was assumed to be $(68.7 \times 10^5 m, 2.7 \times 10^5 m, 4.8 \times 10^5 m, 130 m/s, 820 m/s, 1370 m/s)$, which was within the estimated range. And, the variance of the measurement noise was set to be $(15m)^2$. Figure 3.7 shows the simulation results. All SOMS, SOMSO and GA predicted the initial state quite well and thus resulted in very small estimated errors, except in the initial stage of

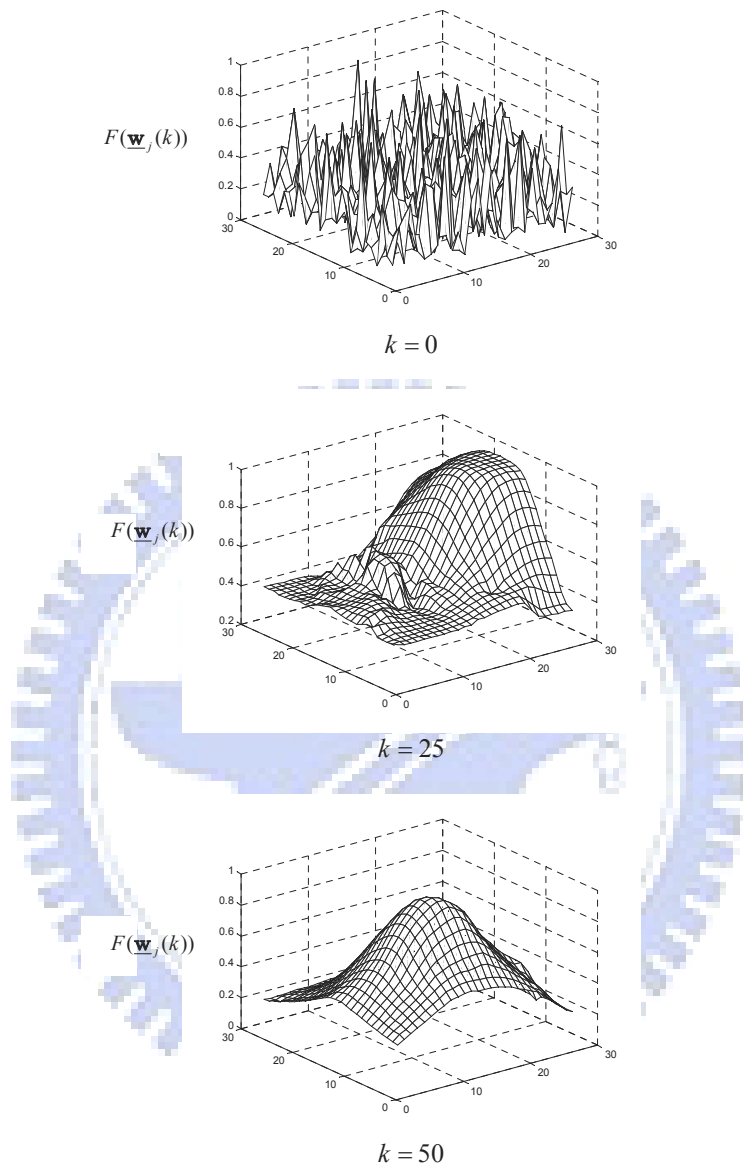
the prediction, as shown in Figure 3.7(a) (only the position error in the X-direction (x_1) is shown for illustration). Figure 3.7(b) shows how the neighborhood function $F(\underline{\mathbf{w}}_j(k))$, described in Eq.(3.2), varied during the SOM learning process. In Figure 3.7(b), from a random distribution in the beginning of the learning, $F(\underline{\mathbf{w}}_j(k))$ gradually approximated the expected Gaussian distribution along with the stage of learning.

In the second set of simulations, we investigated their performances for the condition of a bad estimate of the initial state. In this simulation, the ideal initial state was assumed to be $(64 \times 10^5 m, 4.8 \times 10^5 m, 2.4 \times 10^5 m, 215 m/s, 2130 m/s, 1030 m/s)$, which was outside the estimated range. And, the variance of the measurement noise was enlarged to be $(30m)^2$. From the simulation results shown in Fig. 3.8, the influence of the bad estimate on the SOMS and SOMSO was mostly at the initial stage of the prediction. After the transient, the SOMS and SOMSO still managed to find the optimal initial state. Meanwhile, we also observed that the SOMS converged faster than the SOMSO. As for the GA, it converged very slowly as the optimal initial state did not fall within the estimated range. We thus conclude that the SOMS performed better than the GA for this dynamic trajectory prediction application, and the proposed dynamic weight updating rule was effective. In this chapter we have proposed an SOM-based algorithm for optimization problems, which can be used for both static and dynamic functions in real time. To achieve high learning efficiency for system parameters in different working ranges, we have also proposed a new SOM weight updating rule. The applications of the proposed SOMS on both function optimization problems and dynamic trajectory predictions have clearly proven its effectiveness.



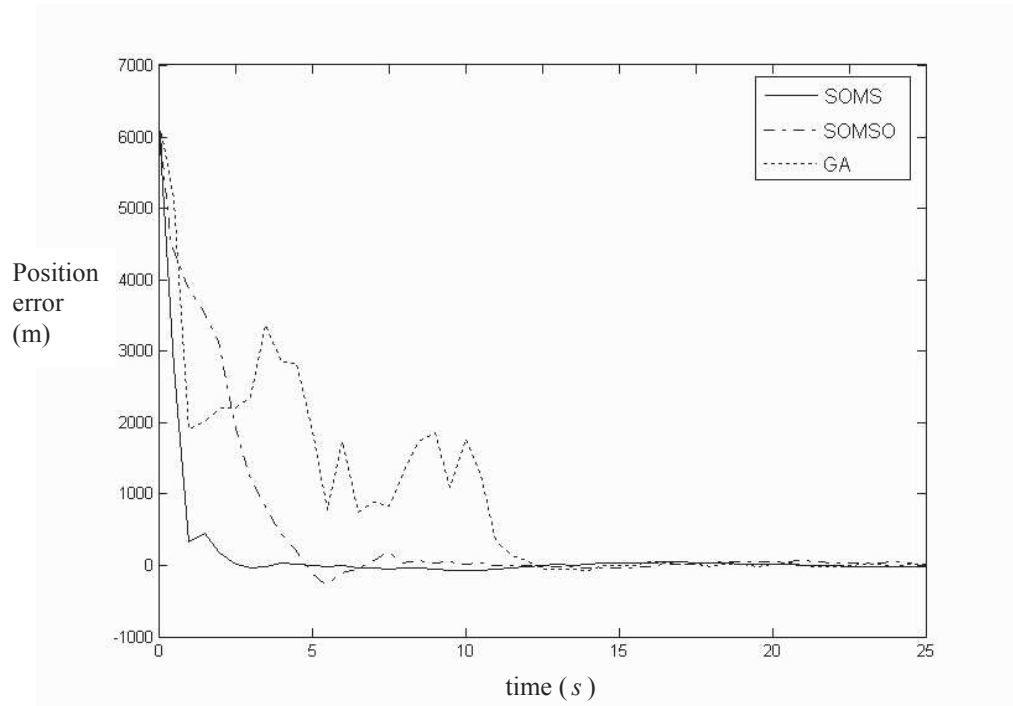
(a) Estimated position error in the X-direction

Figure 3.7 Simulation results for dynamic trajectory prediction using the SOMS, SOMSO, and GA with a good estimate of the initial state: (a) the estimated position error in the X-direction and (b) the variation of the neighborhood function $F(\mathbf{w}_j(k))$ during the SOMS learning process. (Cont.)



(b) Variation of the neighborhood function $F(\underline{\mathbf{w}}_j(k))$

Figure 3.7 Simulation results for dynamic trajectory prediction using the SOMS, SOMSO, and GA with a good estimate of the initial state: (a) the estimated position error in the X-direction and (b) the variation of the neighborhood function $F(\underline{\mathbf{w}}_j(k))$ during the SOMS learning process.



Estimated position error in the X-direction

Figure 3.8 Simulation results for dynamic trajectory prediction using the SOMS, SOMSO, and GA with a bad estimate of the initial state.

Chapter 4

Niching SOM-Based Search

Algorithm

Many global optimization techniques based on population evolution have been successfully applied for finding a global optimum [17, 12, 21], while they cannot cope with optimization with multiple optimal solutions. To tackle this problem, usually the approach employed is to repeat executing the optimization process with different initial populations. Meanwhile, it is possible that the same optimal solution was found even with different initial populations and still several solutions were remained to be found. Consequently, it may require a considerable amount of computational time for finding all the solutions. Thus, a niching method is proposed to extend the ability SOMS, discussed below.

4.1 Niching Method

In the optimization process looking for one single solution, the global search can be achieved under with similar individuals corresponding to similar fitness values. However, in multiple solutions optimization, the dissimilar individuals also correspond to the similar fitness values. If all of the populations are moved toward the one best solution, the remaining optimal solutions will be missed. How to overcome this problem is an interesting topic. It motivates us to propose the SOMS based on niching method (NSOMS), which is able to identify multiple optima in a multimodal domain.

Among previous researches, Eldridge and Gould proposed the punctuated equilibrium (PE) theory [10]. They mentioned that an appearance of new species is a branching mechanism through time. An isolated individual is developed with mutations or differences of gene pool and then some similar individuals with similar features rapidly grow into a larger and larger group. Subsequently, the isolated population evolves into a separate species. Mahfoud proposed a niching method also from such a concept to improve GA by promoting the formation of subpopulations and preserving stably around the optimal solutions. With the niche method, the separate subpopulations parallel convergence into multiple optimal solutions in the search space [28]. A restricted competition selection (RCS) method combined with the pattern search method (PSM) has been proposed and demonstrated that it performed better than two general niching methods (Sharing and Deterministic Crowding [11, 28]) for identifying multiple solutions [18]. Through the RCS restricts competition only the best individual per one niche is maintained. Therefore, the PSM assists GA due to the GA is not suitable for searching with the small population size reduced by the RCS. More niching GAs have been proposed and shown to find multiple solutions efficiently [8, 18, 35].

Based on the niche method, the deterministic competition, instead of the coercive competition, is proposed to restructure SOM by defining subpopulations (subspaces) in a multimodal domain, so as to avoid convergence of the population to a single solution. In the proposed NSOMS, each niche may represent a possible peak in a multimodal domain. A number of similar individuals populating the same niche area is defined as a subpopulation. To preserve a stable subpopulation, every subpopulation has its own living condition, that is, the neurons have their own individual weight-updating rule for each of the niches. In accordance with the proposed NSOMS algorithm, the niche location located on the winning weight site will be moved to approach the real peak location of a multimodal domain gradually. Thus, with many different niches, the NSOMS can be applied to search for multiple optimal solutions. Figure 4.1 shows the influence of niches during the learning process. If the optimization is executed without the niching method, it is possible that the certain solutions will be missed due to the compulsory competition, as shown in Figure 4.1(a). If the optimization is executed by SOMS with niching method (deterministic competition), multiple solutions will be found, as shown in Figure 4.1(b).

Figure 4.2 shows the conceptual diagram of the proposed NSOMS. Compared with the SOMS described in chapter 3, the NSOMS includes a new mechanism, the deterministic competition mechanism, in which the proposed niching method is installed. Every niche represents a possible range where the optimal solution is possibly located and possesses a subpopulation with a number of similar individuals. In every niche the solution leading to the most accurate derived data is chosen as a winner. From the results, the search mechanism updates the weights of these winners and their neighboring neurons. The learning process then continues, and the network will eventually converge to the multiple optimal solutions.

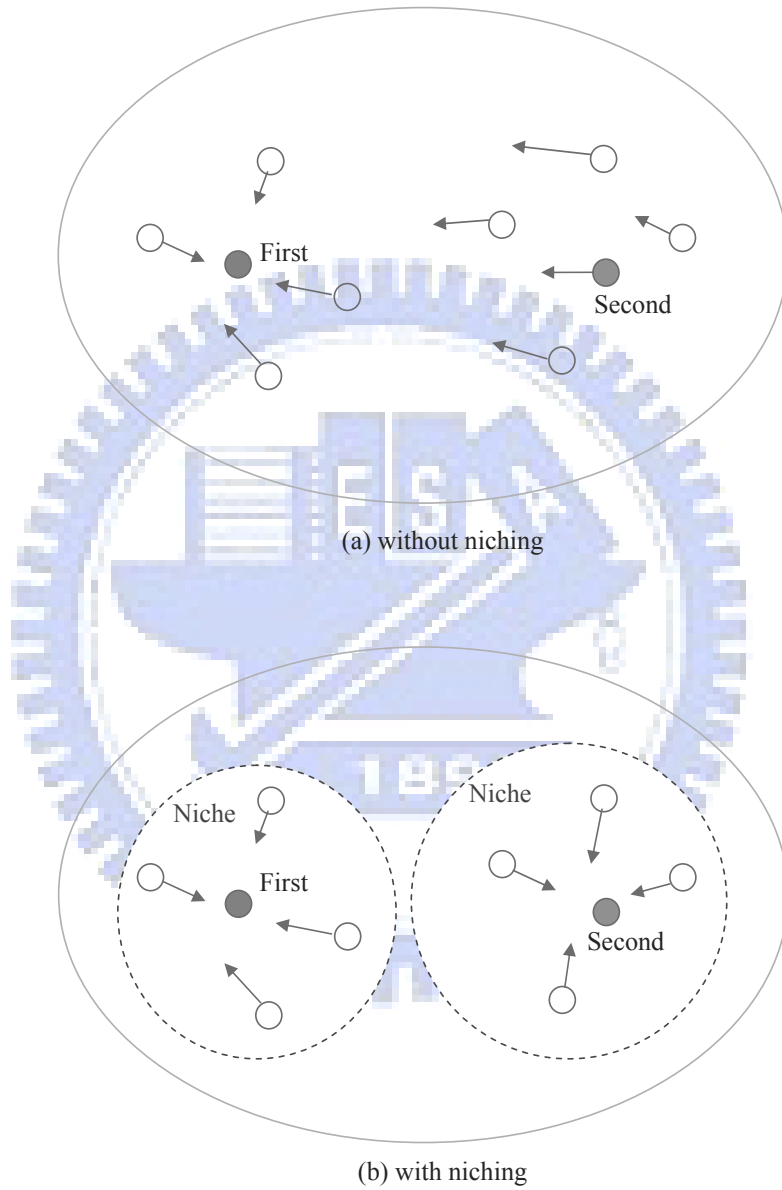


Figure 4.1 Optimization during learning process: (a) without the niching method, (b) with the niching method.

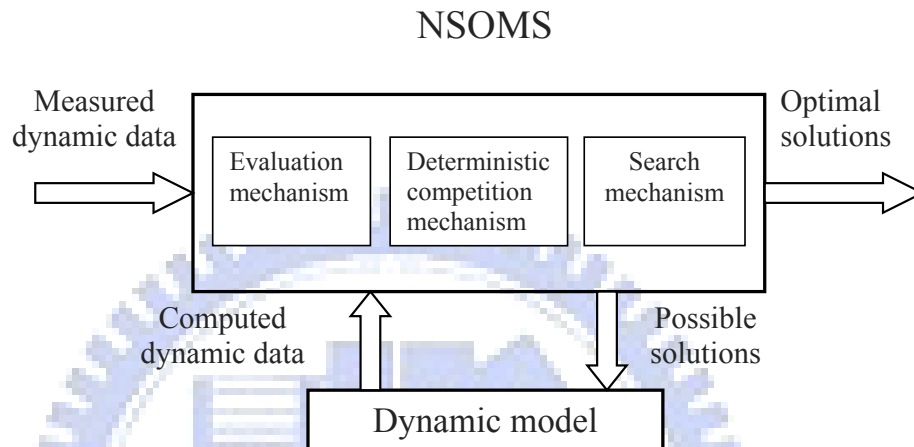


Figure 4.2 Proposed Niching SOM-based search algorithm.

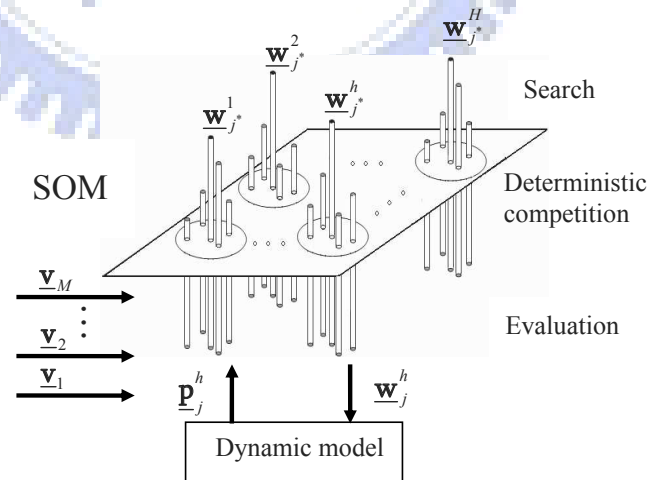


Figure 4.3 Structure and operation of the SOM in the NSOMS.

4.2 Proposed Niching SOMS Weight Updating Rule

Figure 4.3 shows the structure and operation of the SOM in the NSOMS. The SOM performs three operations: evaluation, deterministic competition, and search. In Figure 4.3, initially, we divide the whole SOM network into H subnetworks (niches), each niche comprises N neurons, and each neuron j in the h th niche ($j \in \Lambda^h$) contains a vector of a possible solution set $\underline{\mathbf{w}}_j^h$ (the weight vector). So the total number of neurons equals $H \times N$ in the whole network. The initial center location of the h th niche is set on the $\underline{\mathbf{w}}_e^h$ the average of all $\underline{\mathbf{w}}_j^h$. Take the missile interception application as an example and let the number of incoming missiles be M . Each time new measured data $[\underline{\mathbf{v}}_1, \underline{\mathbf{v}}_2, \dots, \underline{\mathbf{v}}_M]$ are sent into the scheme, the SOM is triggered to operate. All of the possible solution sets in the neurons will then be sent to the dynamic model to derive their corresponding data $\underline{\mathbf{p}}_j^h$. The SOM evaluates the product of all terms of $\|\underline{\mathbf{v}}_m - \underline{\mathbf{p}}_j^h\|$ for $m = 1, \dots, M$. Of all the neurons for the h th niche, it chooses the neuron j^* , which corresponds to the smallest value, as the winner. The learning process then continues, and each niche will eventually converge to the nearest optimal solution.

The search strategy of the population-based optimization algorithm is to find the best individual and move other individuals to approach to the optimal solution. However, one drawback of the SOM-based optimization algorithms is that the network size increases exponentially along with the dimension (r) of the search space. The network needs least 2^r neurons to ensure that each dimension can be considered during the search. To overcome this difficulty, an additional random term, such as random noise and random search methods, is added to raise the optimization efficiency. It might then use few neurons and randomly explore to each coordinate direction during the search. In [40], a small amount of random noise and also a narrowing down method are included in the weight updating rules

to improve its performance. In [29], Michele et al. also derive an alternative optimization algorithm based on neural gas networks (NG-ES) to overcome the bad scaling problem of the KSOM-ES by introducing a mechanism for generating trial points randomly. Wu and Chow proposed a self-organizing and self-evolving agents (SOSENs) neural network that combines multiple simulated annealing algorithms (SAs) and SOM algorithm [44]. Each neuron of SOSENs has its own updating rules (self-evolving) with an SA, and learns from other neurons by the SOM algorithm (self-organizing) after some time. However, when the distance between the best current solution and the real optimal solution is very large, the search process of these methods do not achieve good performance with only small random changes.

From the discussions above, the SOMS weight updating rule previously proposed may not be suitable for optimization in a multimodal domain. Thus, we made several modifications so as to reduce the number of neurons to raise the optimization efficiency. We similarly define a Gaussian distribution function $G(w_{j,i}^h(k))$ as distribution function for each element $w_{j,i}^h(k)$, the i th element in $\underline{\mathbf{w}}_j^h(k)$ in the k th stage of learning:

$$G(w_{j,i}^h(k)) = \exp\left(-\frac{(w_{j,i}^h(k) - w_{e_i}^h(k))^2}{2\sigma_i^h(k)^2}\right) \quad (4.1)$$

where $w_{e_i}^h(k)$ stands for the i th element in $\underline{\mathbf{w}}_e^h(k)$ average of all $\underline{\mathbf{w}}_j^h(k)$, and $\sigma_i^h(k)$ is the standard deviation of the distribution for $w_{j,i}^h(k)$. From the same concept to speed up the learning, described in chapter 3, the strategy is to vary the mean and standard deviation of $G(w_{j,i}^h(k))$ by moving its center toward $w_{j^*,i}^h(k)$ and enlarging (reducing) the standard deviation $\sigma_i^h(k)$ according to the double distance between $w_{j^*,i}^h(k)$ and $w_{e_i}^h(k)$. The new distribution function $\tilde{G}(\tilde{w}_{j,i}^h(k))$ is then formulated as

$$\tilde{G}(\tilde{w}_{j,i}^h(k)) = \exp\left(-\frac{(\tilde{w}_{j,i}^h(k) - \tilde{w}_{e_i}^h(k))^2}{2\tilde{\sigma}_i^h(k)^2}\right) \quad (4.2)$$

where $\tilde{w}_{j,i}^h(k)$ stands for the new $w_{j,i}^h(k)$, $\tilde{w}_{e_i}^h(k)$ the new $w_{e_i}^h(k)$, and $\tilde{\sigma}_i^h(k)$ the new $\sigma_i^h(k)$ after the adjustment. Based on the same strategy in the SOMS, during each iteration of learning, $G(w_{j,i}^h(k))$ is dynamically centered at the location of the winning neuron j^* , with a larger (smaller) width when $w_{e_i}^h(k)$ is much (less) different from $w_{j^*,i}^h(k)$. It thus covers a more fitting neighborhood region, and leads to a higher learning efficiency.

In order to reduce the network size greatly for dealing with the optimization of a multimodal domain, we make the adjustment in weight updating rule of the SOMS. From the mapping property of the SOM, we understand that the SOM cannot obtain a good feature maps with a small network size. Therefore, let the weight vectors form the uniform distribution like the pre-ordered lattice in the neuron space becomes not so meaningful. So far a more fitting search range is already well-defined with the determined mean and standard deviation of the Gaussian distribution function. Thus, we propose a deterministic neighborhood to design the NSOMS weight updating rules. Based on the proposed concept, the new $\tilde{w}_{e_i}^h(k)$ and $\tilde{\sigma}_i^h(k)$ are then formulated as

$$\tilde{w}_{e_i}^h(k) = w_{e_i}^h(k) + \eta_w(k) \cdot (w_{j^*,i}^h(k) - w_{e_i}^h(k)) \quad (4.3)$$

$$\tilde{\sigma}_i^h(k) = \sigma_i^h(k) + \eta_w(k) \cdot (2|w_{j^*,i}^h(k) - w_{e_i}^h(k)| - \sigma_i^h(k)) + \varepsilon \quad (4.4)$$

where $\eta_w(k)$, ($0 < \eta_w(k) \leq 1$) stands for the learning rate in the k th stage of learning and ε the a small value added to avoid that $\sigma_i^h(k)$ rapidly converges to zero. We can set a large value of $\eta_w(k)$ to speed up convergence. However, if $\eta_w(k)$ set to be 1, it

may probably converges to the local optimum. Thus, the premature convergence can be avoided through introduction of the additional adaptation term. From Eqs.(4.3) and (4.4), we can regenerate the new weight $\tilde{w}_{j,i}^h(k)$ from a Gaussian distribution with mean $\tilde{w}_{e_i}^h(k)$ and standard deviation $\tilde{\sigma}_i^h(k)$. With the new weight $\tilde{w}_{j,i}^h(k)$, the weight-updating rule is derived as

$$\begin{cases} w_{j,i}^h(k+1) = \tilde{w}_{j,i}^h(k), & j \in \Lambda^h \text{ and } j \neq j^* \\ w_{j,i}^h(k+1) = w_{j,i}^h(k), & j \in \Lambda^h \text{ and } j = j^* \end{cases} \quad (4.5)$$

Under this learning process, the network will gradually converge to a very small region with $\sigma_i^h(k)$ continuing to decrease.

Sometimes more than two niches eventually converge to the same location of the optimal solution, or several optimal solutions have not been found yet. To overcome this difficulty, a technique for automatically determining the number of niches is introduced into the NSOMS to find as many solutions as possible. First, we use Eqs.(4.3) and (4.4) to detect a searched optimal solution when the standard deviation $\tilde{\sigma}_i^h(k)$ for every element is less than preset value and to determine an effective optimal solution with duplicate optimal solutions excluded when the mean values $\tilde{\mathbf{w}}_e^i(k)$ and $\tilde{\mathbf{w}}_e^j(k)$ are very close. If more than two niches are similar, only one is reserved and the others eliminated from the competition. If the number of niches is equal to the number of effective optimal solutions, a new niche is generated randomly. In other words, we intend to make the niche set size $H(k)$ vary depending on the effective optimal solutions set size $Es(k)$. We thus define a specific relation between the niche set size $H(k)$ and the effective optimal solutions set size $Es(k)$.

$$H(k) = n_1 \cdot Es(k) + n_0 \quad (4.6)$$

where n_0 and n_1 are positive integers which can be either constants or variables decaying along with time. Of course, other types of functions can also be used. During the searching process, in order to prevent that the new niches regenerated converge on the locations of searched optimal solutions repeatedly, the initial center location of the regenerate niche should be far away from the initial center locations of all previous niches as much as possible. Hence, we define an evaluation criterion as

$$D_{W_{i^*}} = \min_i \|\tilde{\mathbf{w}}_R^h - \mathbf{w}_C^i\| \geq \lambda(k) \quad (4.7)$$

where $\tilde{\mathbf{w}}_R^h$ stands for the initial center location of the regenerated niche, \mathbf{w}_C^i the i th location included in the set \mathfrak{R}^C of the initial center locations of all previous niches, and $\lambda(k)$ the distance evaluation parameter. If the minimum distance $D_{W_{i^*}}$ less than $\lambda(k)$, this new niche will be regenerated randomly again. In the initial stage of the learning, $\lambda(k)$ can be set larger to prevent that the similar niches converge to the same optimal solution repeatedly. Later, $\lambda(k)$ may be decreased gradually to let some optimal solutions that are possibly very close can be found. Thus, a function for $\lambda(k)$ that satisfies the demand can be formulated as

$$\lambda(k) = \frac{\lambda(0)}{2} \cdot e^{-k/\tau} \quad (4.8)$$

where the initial value $\lambda(0)$ is the average of all $\|\mathbf{w}_i(0) - \mathbf{w}_e(0)\|$ and τ time constant. Of course, other types of functions can also be used. Under this design, the searching efficacy

of the NSOMS will become faster and more efficiently.

4.3 Visualization Of the Distribution Of Optimal Solutions

Different optimal solutions can now be found by the NSOMS. The next significant task is how to select the most useful solutions from the set of the optimal solutions. Visualization and clustering of high-dimensional data are well-known successes in SOM. We employ the basic principle of the double self-organizing map (DSOM), which updates the weight vectors together with the two-dimensional position vector of the neuron, to achieve the visualization of distribution of optimal solutions. In other words, the positions of the optimal solutions within the parameter space are mapped onto a two-dimensional (2-D) space. Through this map, it allows us to classify the optimal solutions into clusters easily, yielding useful information for solution selection.

The NSOMS is different from the DSOM in that the position updating rules of the neurons in the DSOM cannot be applied directly to optimization due to the system parameters operate in quite different ranges. We thus proposed a new adaptive mapping model to visualize the distribution of the optimal solutions. First, we know that the neuron space and weight vector spaces are with different dimensions, so we have to transform them into the same dimension. We define two Gaussian type functions as the neighborhood functions in the neuron space and the weight vector space, D_i and $F(\mathbf{w}_i(k))$ in the k th stage of learning as

$$D_i = \exp\left(-\frac{\|\mathbf{r}_i - \mathbf{r}_{j^*}^h\|^2}{2\sigma_d}\right) \quad (4.9)$$

$$F(\mathbf{w}_i(k)) = \exp\left(\frac{-\|\mathbf{w}_i(k) - \mathbf{w}_{j^*}^h(k)\|^2}{2\sigma_c}\right) \quad (4.10)$$

where \mathbf{r}_i and $\mathbf{r}_{j^*}^h$ stand for the coordinates of neuron i to entire network and j^* to h th niche, respectively, $\mathbf{w}_i(k)$ the weight vector of neuron i to entire network, σ_d the average of all $\|\mathbf{r}_i - \mathbf{r}_{j^*}^h\|^2$, and σ_c the average of all $\|\mathbf{w}_i(k) - \mathbf{w}_{j^*}^h(k)\|^2$. The Gaussian type function is frequently used as the neighborhood function, and it is differentiable and continuous. With the neighborhood functions, the magnitudes of their distances in the neuron space and weight vector space, respectively, can be normalized to be between 0 and 1. The new learning model in the SOM is designed to let nearby neurons of a feature map correspond to nearby weights. From this mapping, a cost function $E_i(k)$ is then defined as

$$E_i(k) = \frac{1}{2}(D_i - F(\mathbf{w}_i(k)))^2. \quad (4.11)$$

Based on the gradient-descent approach, the position updating rule of the neurons is derived as

$$\begin{aligned} \mathbf{r}_i(k+1) &= \mathbf{r}_i(k) - \eta_p(k) \frac{\partial E_i(k)}{\partial \mathbf{r}_i(k)} \\ &\doteq \mathbf{r}_i(k) + \eta_p(k) (D_i \cdot (D_i - F(\mathbf{w}_i(k))) \cdot (\mathbf{r}_i(k) - \mathbf{r}_{j^*}^h(k))) \end{aligned} \quad (4.12)$$

where $\eta_p(k)$ stands for the learning rate in the k th stage of learning.

It is evident that only two learning parameters $\eta_w(k)$ and $\eta_p(k)$ need to be determined. It is then straightforward to determine the learning parameters for the diverse optimization problems. Through the weight and position updating rules of the neurons

synchronously, the NSOMS can be applied to optimization, in particular, identification and visualization of multiple optimal solutions on the 2D neuron space.

4.4 Applications

To demonstrate its capability, the NSOMS is applied to both function optimization and dynamic trajectory prediction. A PC with 3GHz and MATLAB software were used for all the simulations. Based on the NSOMS, we first develop learning schemes corresponding to each of the applications. Simulations are then executed for performance evaluation. The results are especially compared with the RCS-PSM [18] for their similar searching abilities. To compare their performances, the maximum ratio of the searched peak values to the real peak values (MPR), which denotes the sum of the fitness of the searched optima divided by that of the fitness of the actual optima and the effective number of maintained peaks (NMP), are selected as the performance criteria [18]. A searched peak is detected with the best fitness value at least 80% of the real peak value.

4.4.1 Function Optimization of a Multimodal Domain

For a function optimization problem, the goal may be to maximize (minimize) an object function $O(\cdot)$. Let $O(\underline{\mathbf{w}}_j^h(k))$ be the function value for the weight vector $\underline{\mathbf{w}}_j^h(k)$, which represents a possible solution. During the learning process, P_r is a reference value of the performance evaluation for optimization. Note that the value of P_r is empirical according to the demanded resolution in learning, and we chose it very close to zero. The learning algorithm for function optimization is organized as follows.

Algorithm for function optimization based on the NSOMS: Maximize (minimize) an object function using the NSOMS.

Step 1: Set the stage of learning $k = 0$. Choose H number of niches, N number of neurons within each niche, and reference value P_r . Estimate the ranges of the possible parameter space and randomly store the possible parameters $\underline{\mathbf{w}}_j^h(0)$ into the neurons, where $j = 1, \dots, N$, $h = 1, \dots, H$.

Step 2: Compute $O(\underline{\mathbf{w}}_j^h(k))$ for all $\underline{\mathbf{w}}_j^h(k)$.

Step 3: Among the neurons to every niche, find the one with the largest (smallest) value as the winning neuron j^* for the maximization (minimization) problem.

Step 4: Update the weight vectors of the winning neuron j^* and its neighbors to every niche, and update the positions of neurons to entire network according to the position and weight updating rules described in Eqs.(4.3)-(4.5) and Eq.(4.12).

Step 5: If $\sum_{i=1}^q \tilde{\sigma}_i^h(k) < P_r$ to every niche, $\underline{\mathbf{w}}_{j^*}^h(k)$ is determined to be as an effective optimal solution with duplicate optimal solutions excluded. From Eqs.(4.6) and (4.7), the new $\underline{\mathbf{w}}_j^h(k)$ will be randomly regenerated and then added to the set \mathfrak{R}^c .

Step 6: Check whether the number of iterations is smaller than a pre-specified maximum number of iterations. If it is not, let $k = k + 1$, and go to Step 2; otherwise, the learning process is completed and output all optimal values. The final network mapping is ready for visualizing the distribution and structure of the optimal solutions.

Four multimodal functions are used to demonstrate the proposed algorithm. These functions are defined as follows:

$$F1(x) = \sin^6(5\pi x), \quad \text{where } 0 \leq x \leq 1 \quad (4.13)$$

$$F2(x) = \exp \left[-2(\ln 2) \left(\frac{x - 0.08}{0.854} \right)^2 \right] \cdot \sin^6(5\pi[x^{0.75} - 0.05]) \quad (4.14)$$

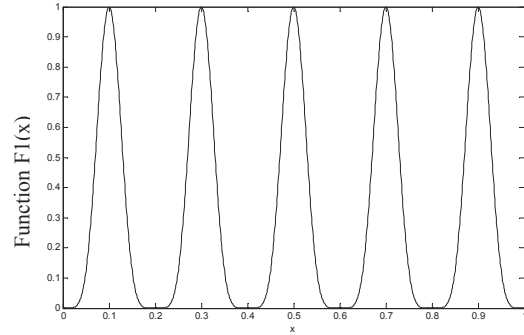
$$F3(x, y) = 500 - \frac{1}{0.002 + \sum_{i=0}^{24} \frac{1}{1+i+(x-a(i))^6+(y-b(i))^6}} \quad (4.15)$$

$$F4(\underline{x}) = \sum_{i=1}^4 \frac{10}{1 + |\underline{x} - \underline{y}_i|^2} \quad (4.16)$$

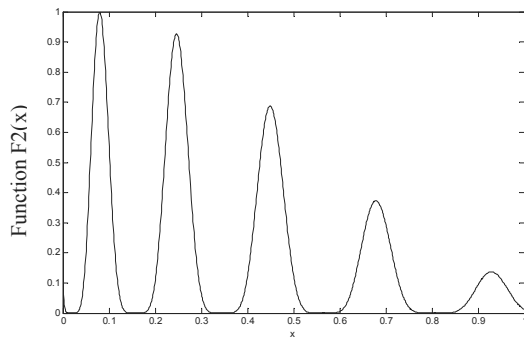
These four test functions have also been used in [15, 18]. The optimization here is to maximize these four functions. The first function, F1, has five peaks with the same height for $x = 0.1, 0.3, 0.5, 0.7,$ and 0.9 . The F2 function has five unequally spaced peaks with different heights. In F3, $a(i) = 16[(i \bmod 5) - 2]$ and $b(i) = 16([i/5] - 2)$. F3 is a two-dimensional function with 25 peaks of different heights in the interval $[476.191, 499.002]$ and the highest peak is located at $(32,32)$. The 2D test functions F1-F3 are shown in Figure 4.4. F4 is a four-dimensional function with 4 peaks of the same height, where \underline{y}_i is a constant vector obtained by permuting the components of the vector $(8,2,2,2)$.

The initial number of the neurons was set to be 1×10 ($H = 1, N = 10$) and the parameters $n_1, n_0, \eta_w(k)$ and $\eta_p(k)$ were set to be constants, 2, 1, 0.4, and 0.4, respectively for all function optimization. For comparison, we also use the RCS-PSM for function minimization, which is with an initial population size of 10 to match with that of the SOM, and the crossover and mutation probability of 0.6 and 0.0333 for the GA operation, respectively. To automatically determining the number of niches, we use the same condition described in Eq.(4.6) to RCS-PSM for simulations.

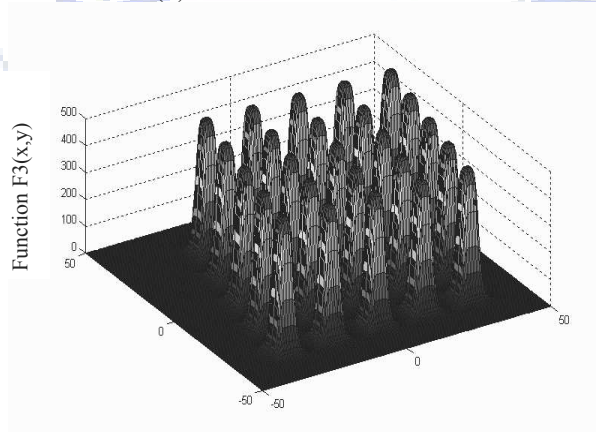
For each test functions, each algorithm is repeated 30 times and carried out after 30 iterations each time. The comparison results of average are listed in Table 4.1. As shown in Table 4.1, NSOMS and RCS-PSM perform quite well in terms of the number and quality



(a) uniform sine function



(b) nonuniform sine function



(c) Shekel's Foxholes function

Figure 4.4 Three multimodal functions. (a) F1: uniform sine function, (b) F2: nonuniform sine function, and (c) F3: Shekel's Foxholes function.

of peaks obtained. But, NSOMS works better than RCS-PSM for the computational time. Figure 4.5 shows the variations of NMP and MPR according to the increase of time for F3 test function. We observed that the NSOMS converged faster than the RCS-PSM. Figure 4.6 shows the variation of the network structure on the SOM using the NSOMS for F4 test function. In Figure 4.6(a), we only show the variation of the best four niches. It shows clearly that the positions of neurons on the SOM reveal the distribution and structure of the optimal solutions. Figure 4.6(b) shows the neighboring relationship of the neurons of the best four niches, and we can observe that the neighborhood functions of neurons and weights, described in Eqs.(4.9) and (4.10), varied during the NSOMS learning process, and eventually they are very close to each other. From these results, the NSOMS demonstrates the identification and visualization of the optimal solutions of high dimension.

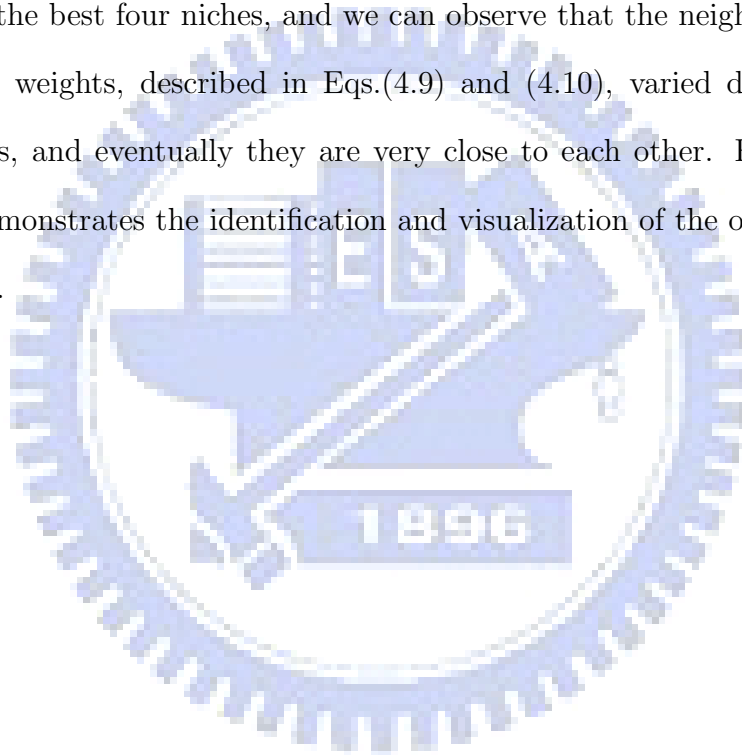
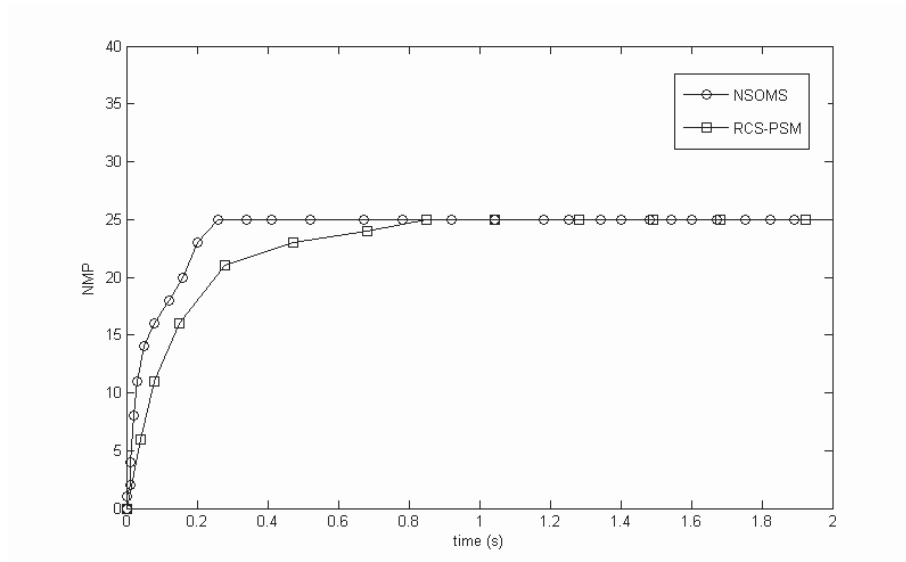
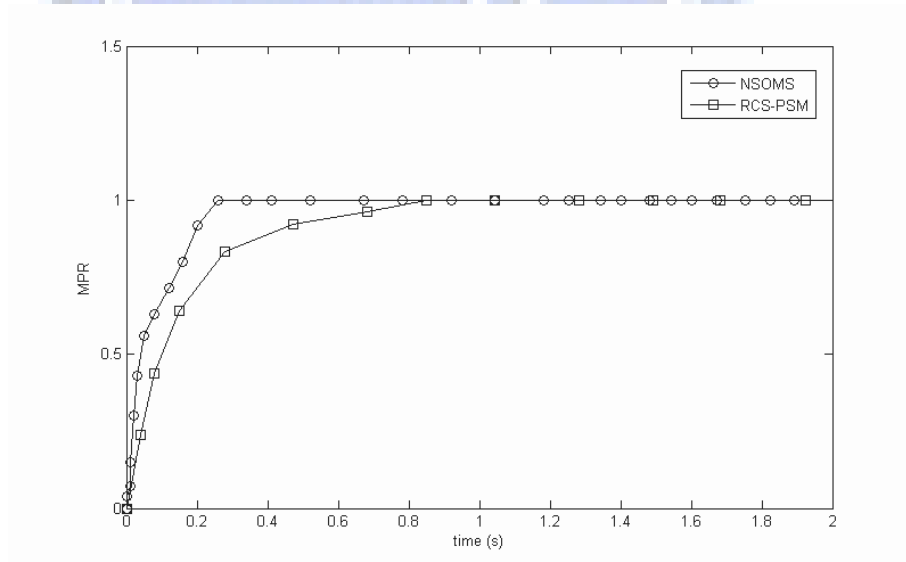


Table 4.1: Comparison results for NSOMS and RCS-PSM on the 4 test functions.

Function	Method	NMP	MPR	Time (s)
F1	NSOMS	5	1	0.1213
	RCS-PSM	5	1	0.2154
F2	NSOMS	5	1	0.1436
	RCS-PSM	5	1	0.2417
F3	NSOMS	25	1	2.0301
	RCS-PSM	25	1	5.5171
F4	NSOMS	4	1	0.1337
	RCS-PSM	4	1	0.7265

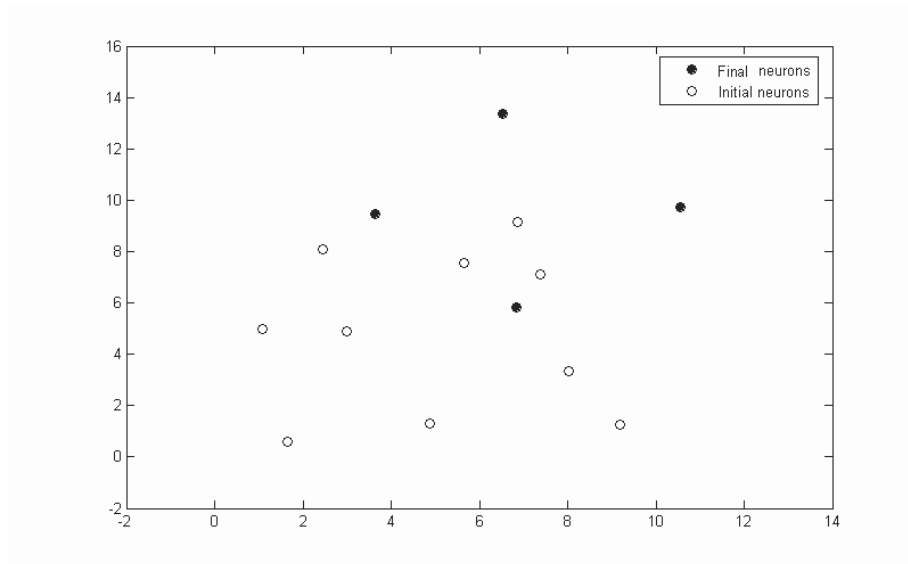


(a) Number of maintained peaks

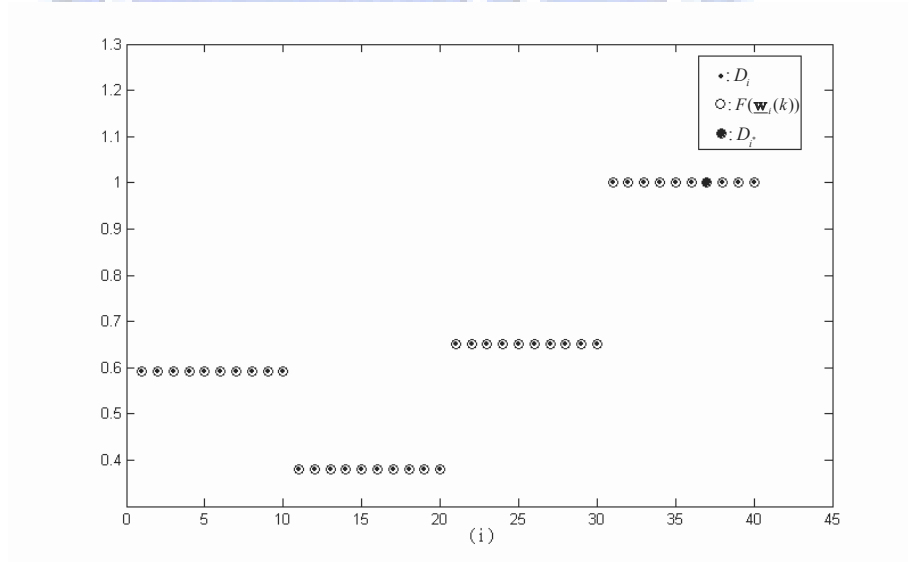


(b) Maximum peak ratio

Figure 4.5 Convergence comparisons for F3 function: (a) the variation of the number of maintained peaks and (b) the variation of the maximum peak ratio during the NSOMS learning process.



(a) Projection result in 2D neuron space



(b) Final neighborhood function values

Figure 4.6 The results obtained by the NSOMS for F4 function: (a) projection result in the 2D neuron space, and (b) final neighborhood function values.

4.4.2 Multiple Dynamic Trajectories Prediction

For a dynamic trajectory prediction problem, the goal may be to estimate the initial position and velocity of a moving object using the measured data. For the trajectory prediction of multiple targets, here we assume that the target detection has been carried out in advance, and we focus on the estimation of the initial states of multiple moving objects. Through a learning process, the NSOMS may determine a most probable initial state of each target through repeatedly comparing the measured data with the predicted trajectories derived from the possible initial states stored in the neurons of the SOM.

In this application, the nonlinear dynamic equation describing the trajectory of the moving object and the measurement equation are as those described in Chapter 3. The learning algorithm for multiple trajectories prediction is organized as follows.

Algorithm for multiple trajectory prediction based on the SOMS: Predict an optimal initial state for the trajectory of every moving target using the measured position data.

Step 1: Set the stage of learning $k = 0$. Choose H number of niches, N number of neurons within each niche, and reference value P_r . Estimate the ranges of the possible position and velocity of the moving object, and randomly store the possible initial states $\underline{\mathbf{w}}_j^h(0)$ into the neurons, where $j = 1, \dots, N$, $h = 1, \dots, H$.

Step 2: Send $\underline{\mathbf{w}}_j^h(k)$ into the dynamic model, described in Eqs.(3.25) and (3.28), to compute $\underline{\mathbf{p}}_j^h(k)$.

Step 3: For each neuron j of every niche, compute its output $O_j^h(k)$:

$$O_j^h(k) = \prod_{m=1}^M \left[\sum_{i=0}^k \left\| \mathbf{v}_m(i) - \underline{\mathbf{p}}_j^h(i) \right\| \right] \quad (4.17)$$

where M is the number of the objects detected.

Find the winning neuron j^* with the minimum $O_{j^*}^h(k)$:

$$O_{j^*}^h(k) = \min_j O_j^h(k) \quad (4.18)$$

Step 4: Update the weight vectors of the winning neuron j^* and its neighbors to every niche, and update the positions of neurons of the entire network.

Step 5: If $\sum_{i=1}^q \tilde{\sigma}_i^h(k) < P_r$ for every niche, $\underline{\mathbf{w}}_{j^*}^h(k)$ is determined to be as an effective optimal solution with the duplicate optimal solutions excluded. The prediction process outputs the predicted optimal initial states to the dynamic model to derive the object trajectories. From Eqs.(4.6) and (4.7), the new $\underline{\mathbf{w}}_j^h(k)$ will be randomly regenerated and then added into the set \mathfrak{R}^c .

Step 6: Check whether the number of iterations is smaller than a pre-specified maximum number of iterations. If it is not, let $k = k + 1$, and go to Step 2; otherwise, the prediction process is completed and output optimal states of all objects. The final network mapping provides the visualization of the distribution of the optimal states. In addition, during each stage of learning, we perform a number of learning to increase the SOM learning speed. This number of learning is set to be a large number in the initial stage of the learning process, such that the NSOMS may converge faster at the price of more oscillations, and decreased gradually to achieve smooth learning in later stages of learning.

To demonstrate the effectiveness of the proposed NSOMS and weight updating rule, we performed a series of simulations for dynamic trajectory prediction based on using the proposed NSOMS and two NSOMS without the proposed dynamic weight updating rule (named as SOMSO-1 and SOMSO-2 by using the SOMO's and SOSENS's weight updating rules, respectively, in place of that of the NSOMS). The trajectory to predict in the simulations was designed to emulate that of a missile. Its governing equations of motion in the 3D Cartesian coordinate system are as those described in Chapter 3. The ranges of the possible initial states $\underline{\mathbf{w}}_j(0)$ were estimated to be

$$\begin{aligned}
 1.14 \times 10^6 m &\leq x_1(0) \leq 2.14 \times 10^6 m \\
 25.75 \times 10^6 m &\leq x_2(0) \leq 26.75 \times 10^6 m \\
 15.9 \times 10^6 m &\leq x_3(0) \leq 16.9 \times 10^6 m \\
 1785 m/s &\leq x_4(0) \leq 2285 m/s \\
 -180 m/s &\leq x_5(0) \leq 820 m/s \\
 -2000 m/s &\leq x_6(0) \leq 1000 m/s.
 \end{aligned} \tag{4.19}$$

Within the ranges described in Eq.(4.19), the possible initial positions and velocities of the missile were selected and stored into the 1125 (5×225) neurons of the 2D SOM. We consider three targets to be detected in the following simulations. The parameters of NSOMS were set to be $n_1 = 2$, $n_0 = 1$, respectively. The additional adaptation term ε , described in Eq.(4.4), was set to be 0.1. For comparison, we set the same learning rate as those in the NSOMS described in Sect. 4.4,1, and several parameters of the SOMSO-1 and SOMSO-2 were adjusted via a trial-and-error process to yield salient performance. The number of learning is set to be 20 during each stage of learning.

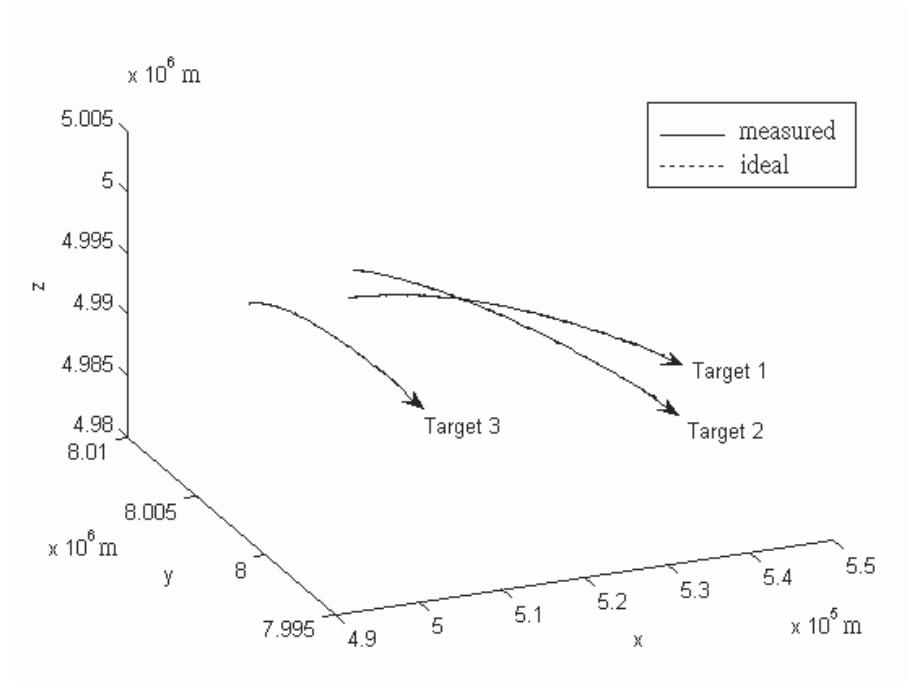
We first applied the SOMS, SOMSO-1, and SOMSO-2 for trajectory prediction with a good estimate of the initial state. Three ideal initial states of the missiles were assumed to be within the estimated range. The variance of the measurement noise was set to be

$(15m)^2$. Figure 4.7 shows the simulation results. The ideal and measured trajectories are shown in Figure 4.7(a). These three methods predicted the initial state quite well and thus resulted in very small estimated errors, except in the initial stage of the prediction, as shown in Figure 4.7(b)-(d) (the estimated initial state error is shown for illustration). We observed that the NSOMS converged faster than the other methods did. Figure 4.8 shows the neighboring relationship of neurons of the best three niches using the NSOMS. In Figure 4.8(a), from a random distribution of neurons in the beginning of the learning, the mapping structure gradually form three clusters along with the stage of learning. Figure 4.8(b) shows how the neighborhood function D_i and $F(\mathbf{w}_i(k))$ varied during the SOM learning process, and eventually they are very close to each other.

In the second set of simulations, we investigated their performances for the condition of a bad estimate of the initial state. The ranges of the possible initial states $\mathbf{w}_j(0)$ were estimated to be

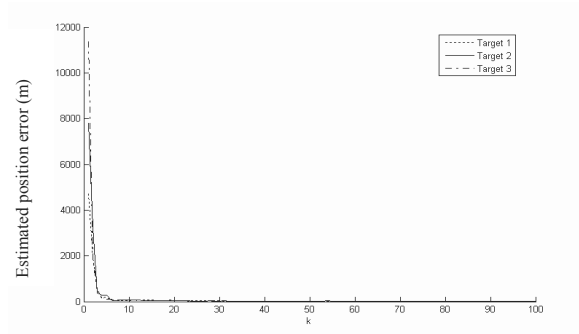
$$\begin{aligned}
 1.94 \times 10^6 m &\leq x_1(0) \leq 2.94 \times 10^6 m \\
 27.75 \times 10^6 m &\leq x_2(0) \leq 28.75 \times 10^6 m \\
 16.9 \times 10^6 m &\leq x_3(0) \leq 17.9 \times 10^6 m \\
 2285 m/s &\leq x_4(0) \leq 2785 m/s \\
 500 m/s &\leq x_5(0) \leq 1500 m/s \\
 0 m/s &\leq x_6(0) \leq 3000 m/s.
 \end{aligned} \tag{4.20}$$

In this simulation, three ideal initial states were assumed to be outside of the estimated range. The variance of the measurement noise was enlarged to be $(50m)^2$. The setting of all parameters was set to be the same as them in first simulation. Figure 4.9(a) shows the ideal and measured trajectories and Figure 4.9(b)-(d) the estimated initial error. From the results, the influence of the bad estimate on these methods was mostly at the initial stage of the prediction. After the transient, the NSOMS still managed to find the optimal

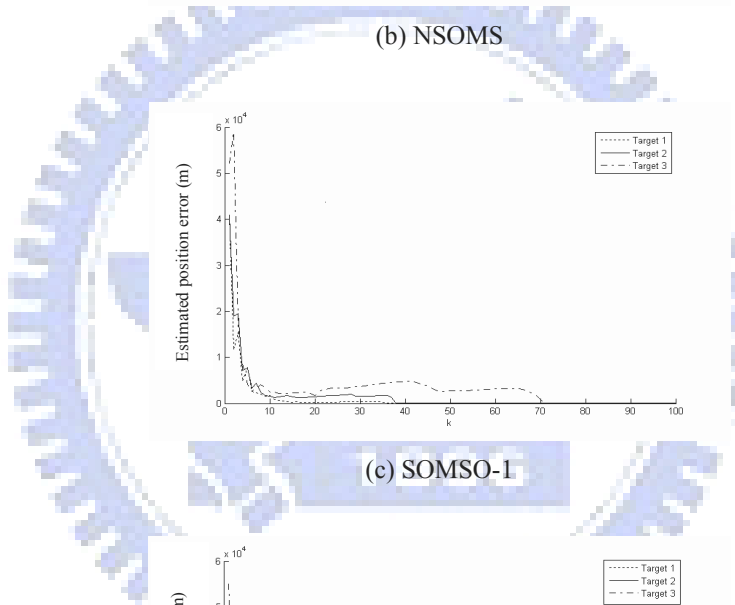


(a) Ideal and measured TBM trajectories

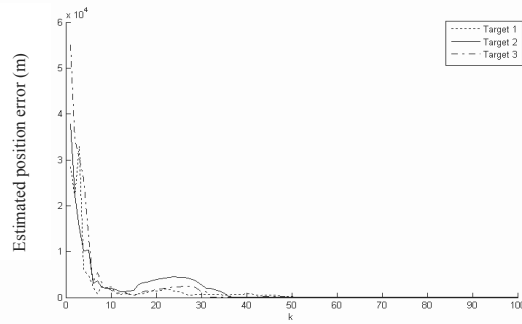
Figure 4.7 Simulation results for the multiple trajectories prediction using the NSOM, SOMSO-1, and SOMSO-2 with a good estimate of the initial state: (a) the ideal and measured TBM trajectories, (b)-(d) the estimated initial state error by using the NSOM, SOMSO-1, and SOMSO-2. (Cont.)



(b) NSOMS

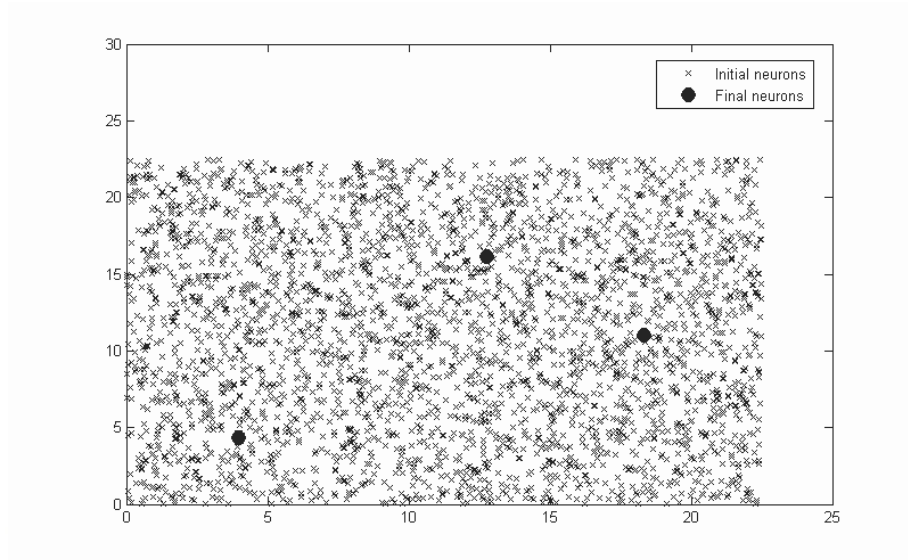


(c) SOMSO-1

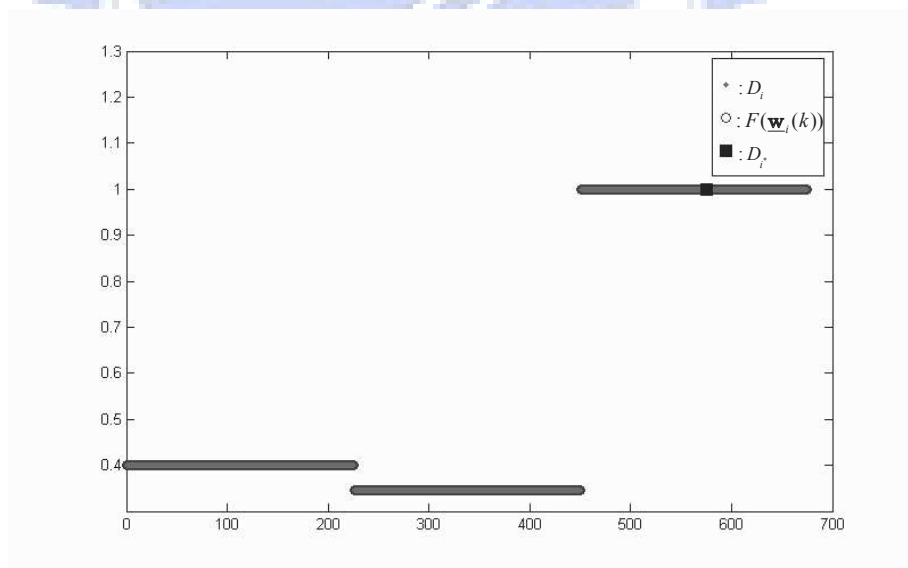


(d) SOMSO-2

Figure 4.7 Simulation results for the multiple trajectories prediction using the NSOM, SOMSO-1, and SOMSO-2 with a good estimate of the initial state: (a) the ideal and measured TBM trajectories, (b)-(d) the estimated initial state error by using the NSOM, SOMSO-1, and SOMSO-2.



(a) Projection result in 2D neuron space

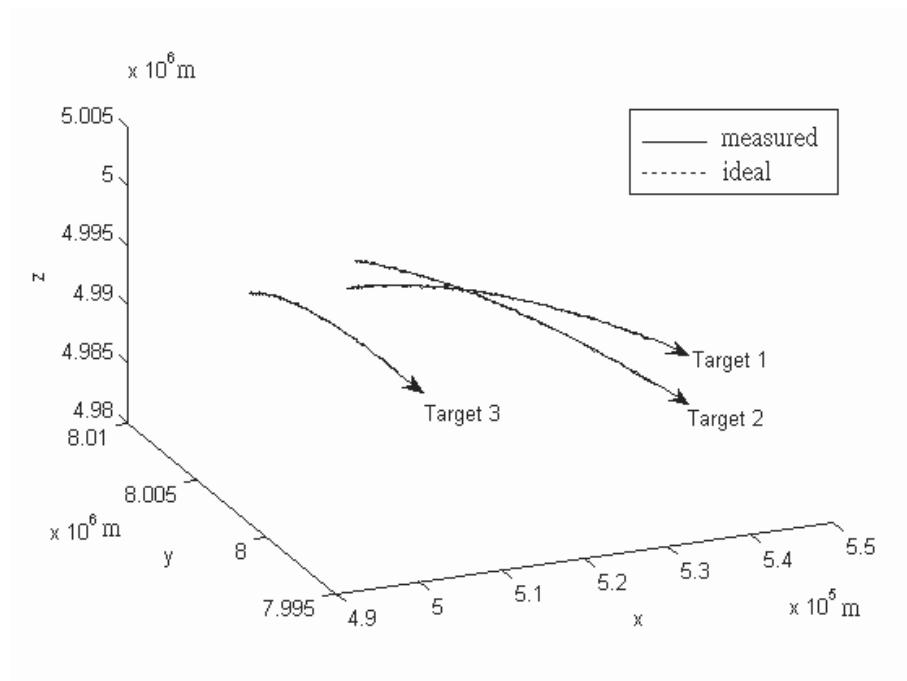


(b) Final neighborhood function values

Figure 4.8 Final results obtained by the NSOMS for the multiple trajectories prediction: (a) projection result in 2D neuron space and (b) final neighborhood function values.

initial states of all targets. Meanwhile, we also observed that the NSOMS converged very faster than the other methods did. As for the SOMSO-1 and SOMSO-2, they converged very slowly as the optimal initial state did not fall within the estimated range. In Figure 4.10(a), from a random distribution of neurons in the beginning of the learning, the mapping structure gradually form three clusters along with the stage of learning. Figure 4.10(b) shows how the neighborhood function D_i and $F(\mathbf{w}_i(k))$ varied during the SOM learning process, and eventually they are very close to each other.

To further demonstrate the NSOMS search ability, we used four different network sizes and learning parameters to run these optimization algorithms for the dynamic trajectory prediction. We only consider one target for comparison. The ideal initial state of the missile was assumed to be within the estimated range, described in Eq.(4.19). Figure 4.11 shows how the population size affects for these algorithms. We observed that the NSOMS performs better than the SOMSO-1 and SOMSO-2. As Figure 4.11 shows, the SOMSO-1 and SOMSO-2 did not converge to the optimal state when the network size was very small. Figure 4.12 show the performance with the different learning parameters for the influence of the these algorithms under a fixed network size (1×225). As Figure 4.12 illustrates, we observed that the large learning parameter may speed up the learning of the NSOMS. Although a small learning parameter made the NSOMS converge slight slowly, it still converged faster than the SOMSO-1 and SOMSO-2 did. Table 4.2 shows that the different network (population) size and learning parameter affect the learning result for the NSOMS, SOMSO-1 and SOMSO-2 in detail. We calculated the RMS (Root-Mean-Square) value of the error between the ideal and predicted trajectories at $k = 100$ to evaluate their performance. The comparison results of average of repeated 30 times are listed in Table 4.2. We observed that the NSOMS performed better than the other methods did. Figure 4.13 shows the performance of 5 runs with the same initial weights



(a) Ideal and measured TBM trajectories

Figure 4.9 Simulation results for multiple trajectories prediction using the NSOM, SOMSO-1, and SOMSO-2 with a bad estimate of the initial state: (a) the ideal and measured TBM trajectories, (b)-(d) the estimated initial state error by using the NSOM, SOMSO-1, and SOMSO-2. (Cont.)

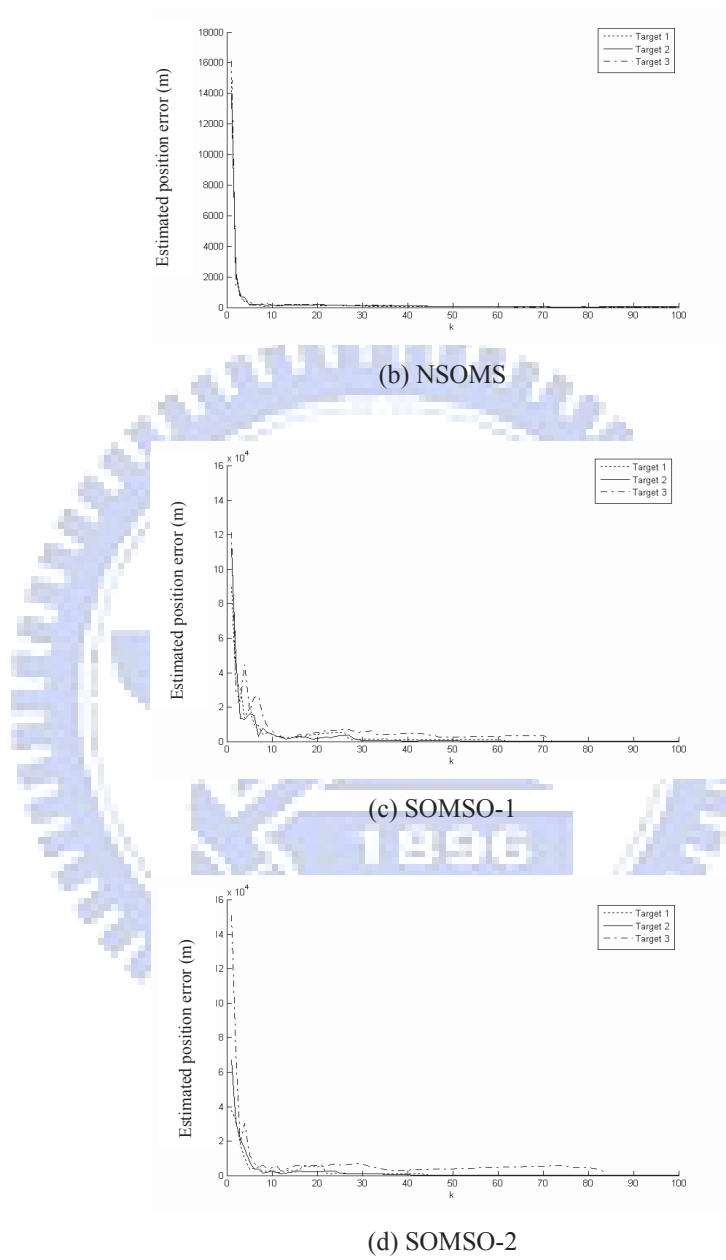
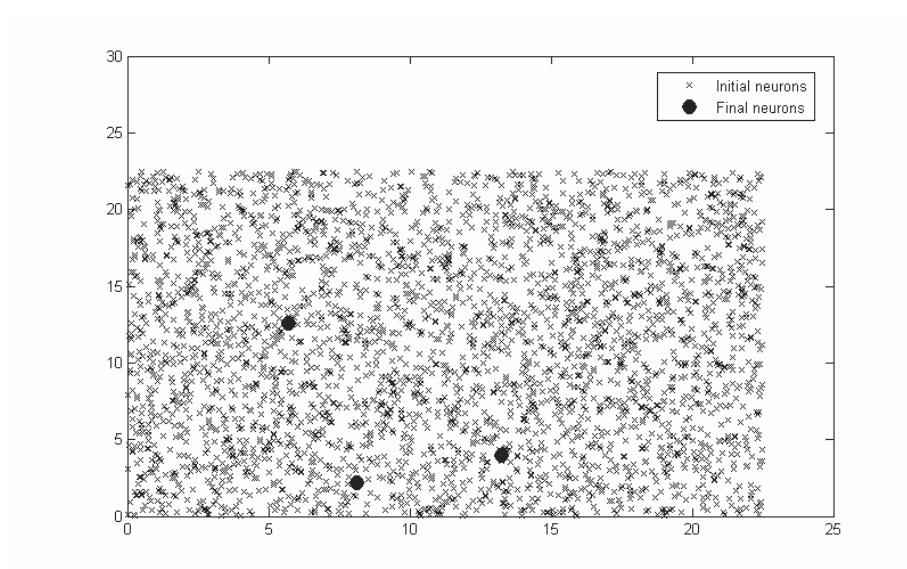
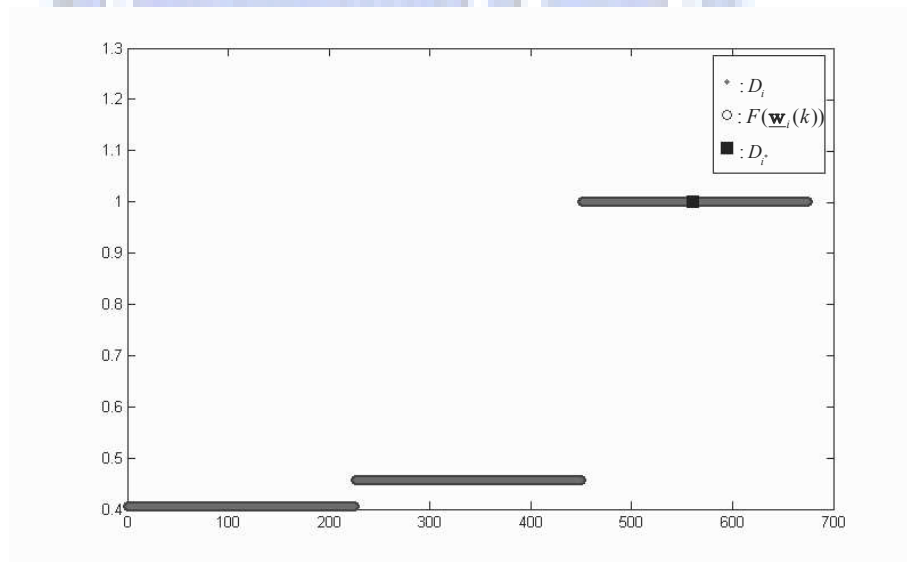


Figure 4.9 Simulation results for multiple trajectories prediction using the NSOM, SOMSO-1, and SOMSO-2 with a bad estimate of the initial state: (a) the ideal and measured TBM trajectories, (b)-(d) the estimated initial state error by using the NSOM, SOMSO-1, and SOMSO-2.

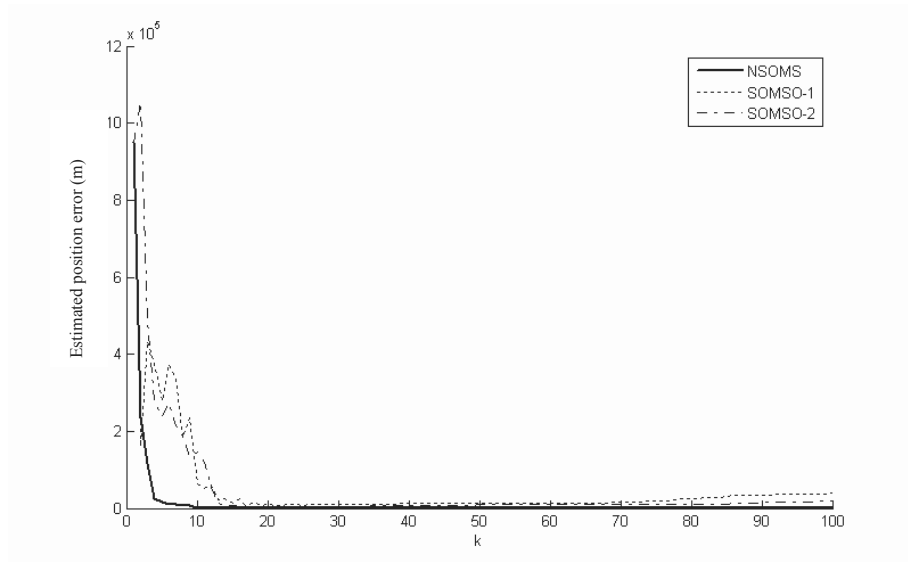


(a) Projection result in 2D neuron space

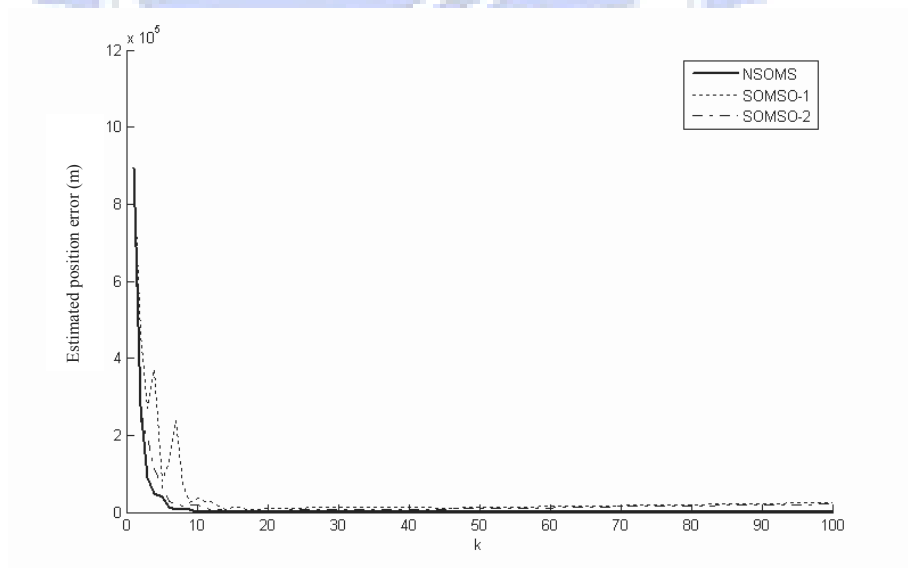


(b) Final neighborhood function values

Figure 4.10 Final results obtained by the NSOMS for the multiple trajectories prediction : (a) projection result in 2D neuron space and (b) final neighborhood function values.

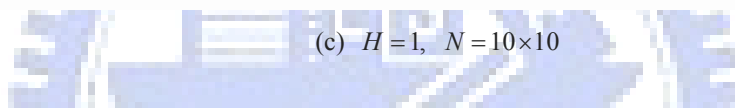
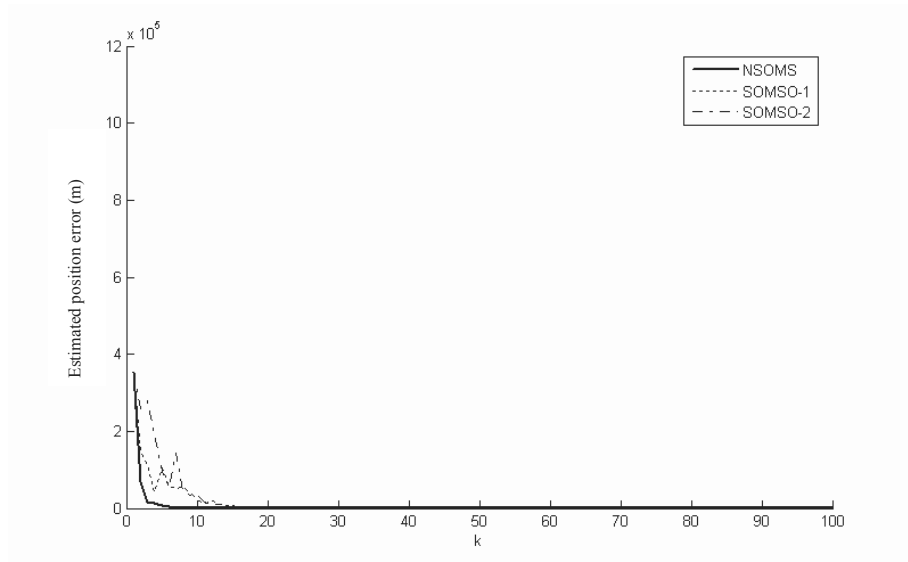


(a) $H = 1, N = 3 \times 3$

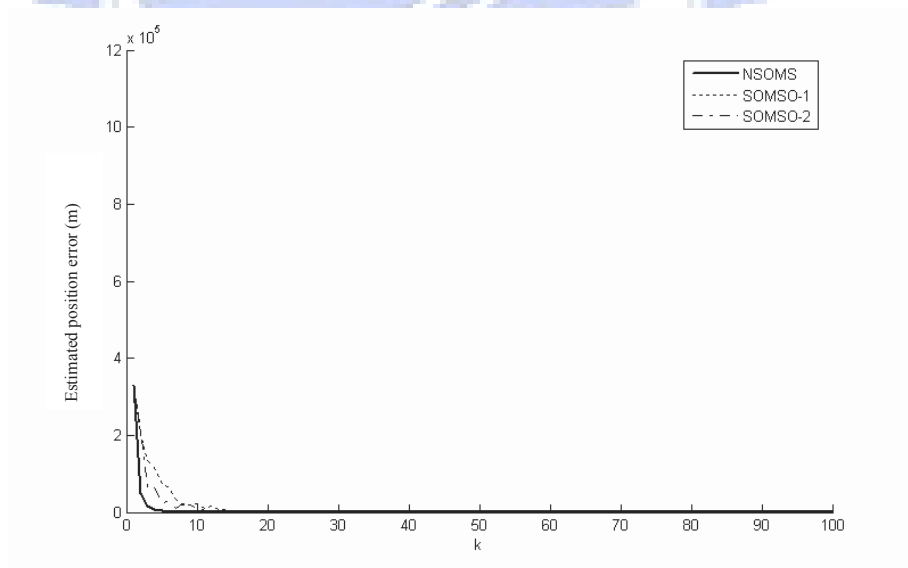


(b) $H = 1, N = 5 \times 5$

Figure 4.11 Performance for different network sizes using the NSOMS, SOMSO-1, and SOMSO-2. (Cont.)

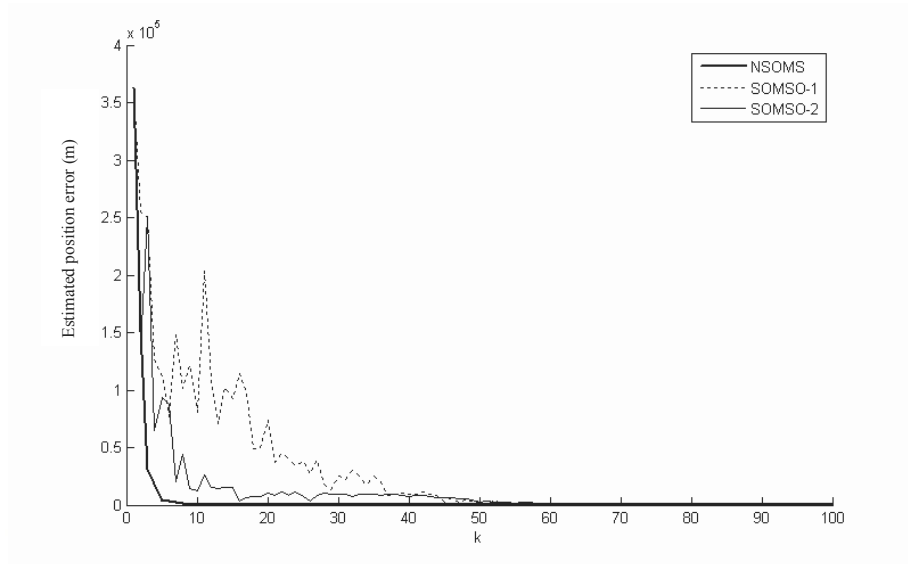


(c) $H = 1, N = 10 \times 10$

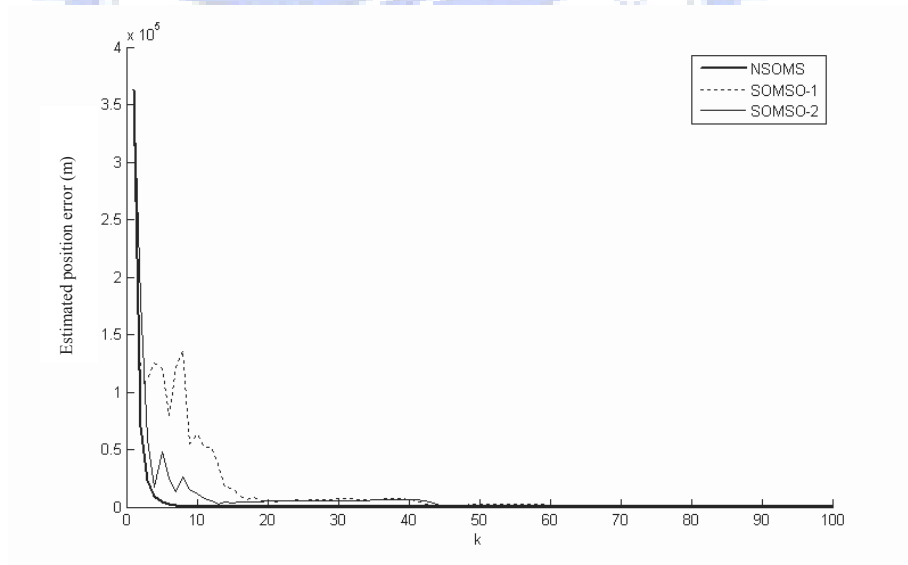


(d) $H = 1, N = 20 \times 20$

Figure 4.11 Performance for different network sizes using the NSOMS, SOMSO-1, and SOMSO-2.

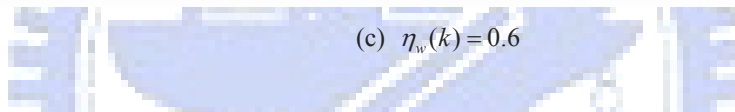
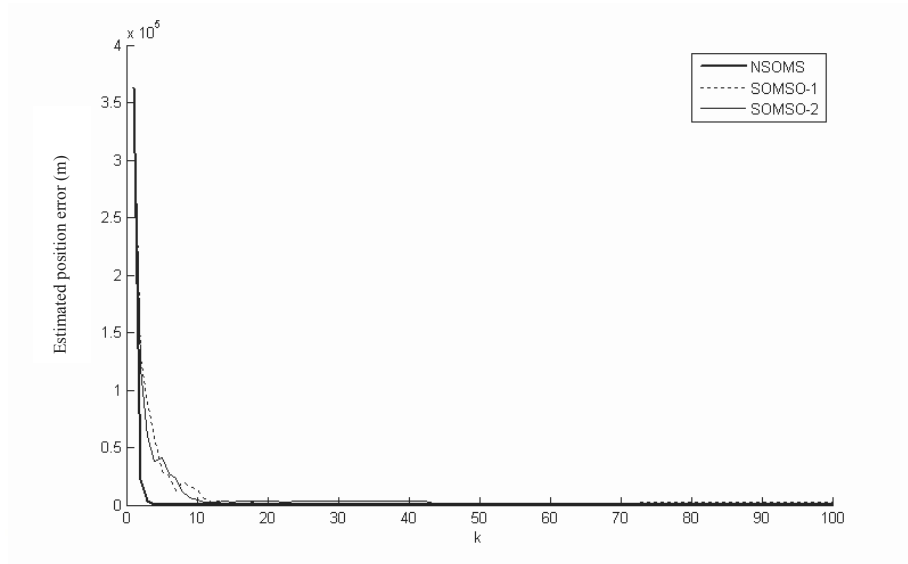


(a) $\eta_w(k) = 0.2$

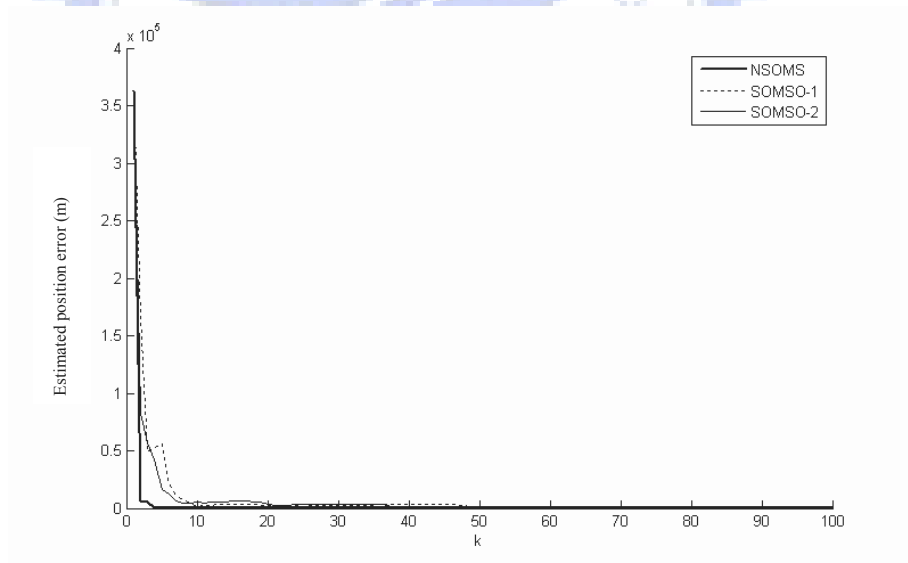


(b) $\eta_w(k) = 0.4$

Figure 4.12 Performance for different parameters using the NSOMS, SOMSO-1, and SOMSO-2. (Cont.)



(c) $\eta_w(k) = 0.6$



(d) $\eta_w(k) = 0.8$

Figure 4.12 Performance for different parameters using the NSOMS, SOMSO-1, and SOMSO-2.

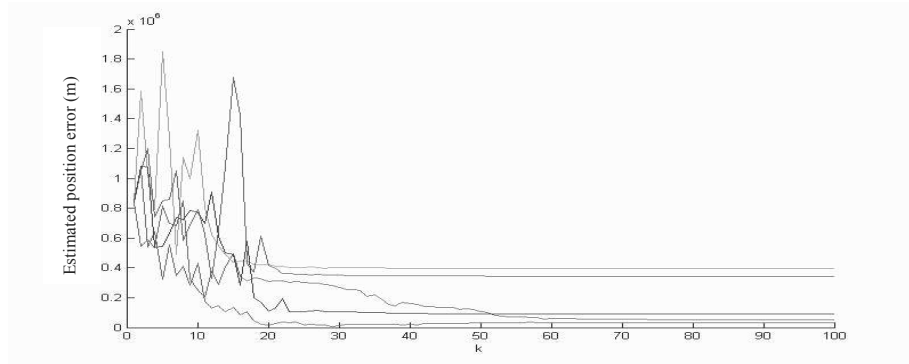
Table 4.2: Comparison results for NSOMS, SOMSO-1, and SOMSO-2 on the dynamic trajectory prediction.

Network size $H = 1$	Learning rate $\eta_w(k)$	Mean and Standard Deviation of RMS Values (m)		
		NSOMS	SOMSO-1	SOMSO-2
$N = 2 \times 2$	0.2	30.991 ± 6.008	$6.161 \times 10^6 \pm 1.697 \times 10^6$	$9.423 \times 10^3 \pm 4.346 \times 10^3$
	0.5	30.088 ± 1.385	$6.811 \times 10^4 \pm 5.189 \times 10^4$	$6.067 \times 10^3 \pm 2.913 \times 10^3$
	0.8	26.325 ± 1.047	$6.152 \times 10^4 \pm 5.016 \times 10^4$	$6.034 \times 10^3 \pm 2.162 \times 10^3$
$N = 5 \times 5$	0.2	25.861 ± 1.324	$1.445 \times 10^3 \pm 1.476 \times 10^3$	$2.571 \times 10^2 \pm 3.421 \times 10^2$
	0.5	25.641 ± 0.906	$1.645 \times 10^2 \pm 2.201 \times 10^2$	$1.502 \times 10^2 \pm 1.239 \times 10^2$
	0.8	25.985 ± 0.937	$1.391 \times 10^2 \pm 1.248 \times 10^2$	$1.165 \times 10^2 \pm 1.124 \times 10^2$
$N = 10 \times 10$	0.2	25.981 ± 1.233	$1.056 \times 10^2 \pm 1.016 \times 10^2$	44.012 ± 35.543
	0.5	25.711 ± 0.786	25.838 ± 0.752	25.834 ± 0.761
	0.8	25.775 ± 1.075	25.834 ± 1.258	25.845 ± 1.281

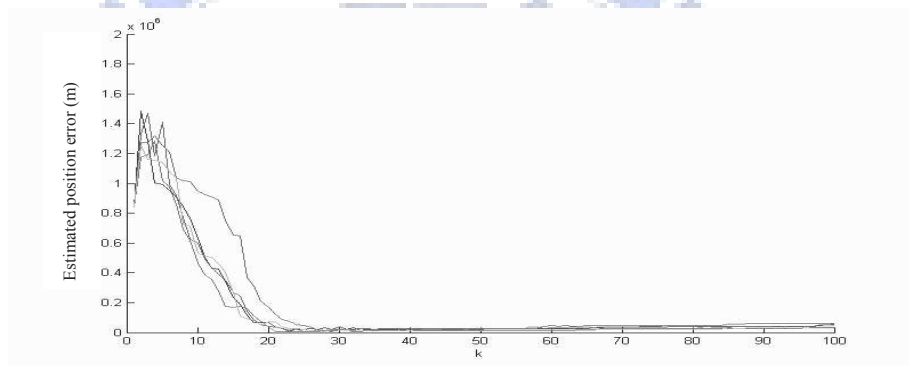
for the influence of three algorithms under a quite small network size ($H = 1, N = 2 \times 2$). Figure 4.14 shows the performance of 5 runs with the different initial weights. From the results shown in Table 4.2 and Figures 4.11-4.14, the NSOMS was more robust and converged faster than the other two algorithms did.

We also performed simulations based on using the RCS-PSM. We modified the internal parameters of the PSM method to enhance its search abilities. However, it was not that straightforward to determine its parameters properly, and the process was time-consuming. The SOMSO-1, SOMSO-2, and RCS-PSM might not be that effective under such circumstances that the ranges of the possible initial states may be uncertain and varying in noisy, unknown environments. We thus conclude that the NSOMS performed better than the SOMSO-1, SOMSO-2, and RCS-PSM for this dynamic trajectory prediction application, and the proposed dynamic weight updating rule was effective.

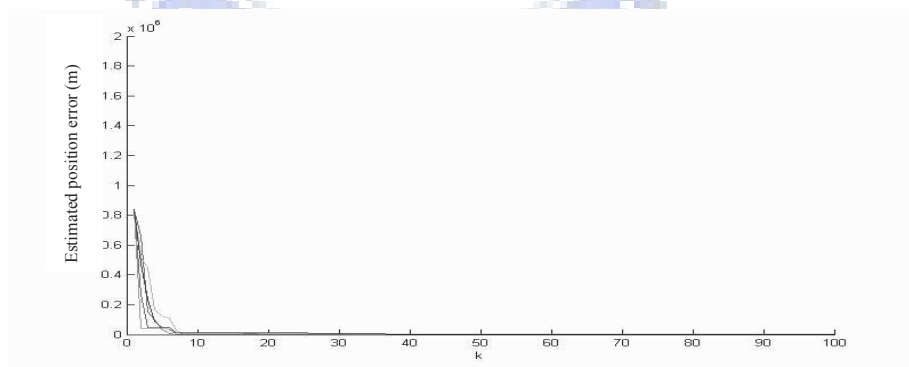
As a summary, in this chapter, a niching SOM-based search algorithm has been proposed for identification and visualization of multiple optimal solutions. Through reducing the network size greatly for search in the high-dimensional space, we have also proposed a niche weight updating rule to raise the learning efficiency. The final network structure allows us to easily classify the optimal solutions into clusters, thus yielding useful information for solution selection.



(a) SOMSO-1

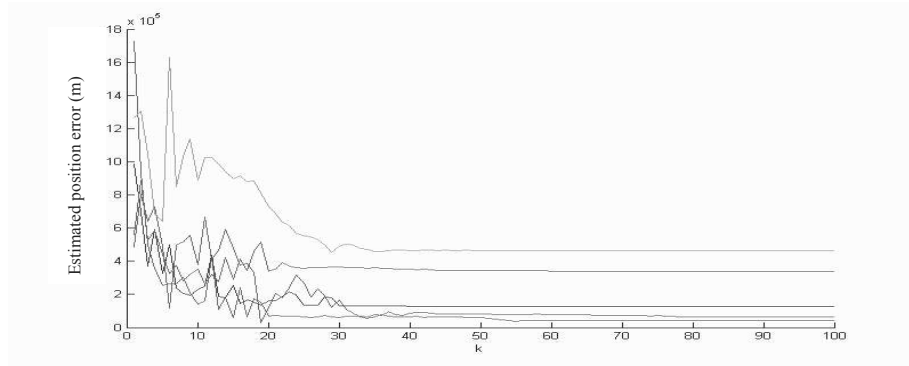


(b) SOMSO-2

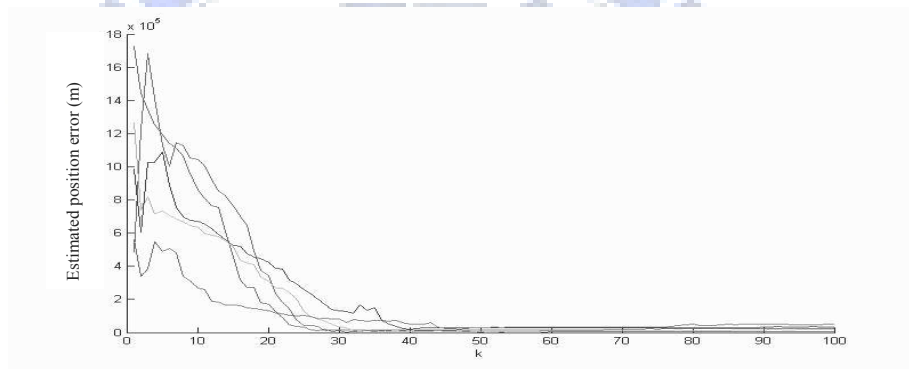


(c) NSOMS

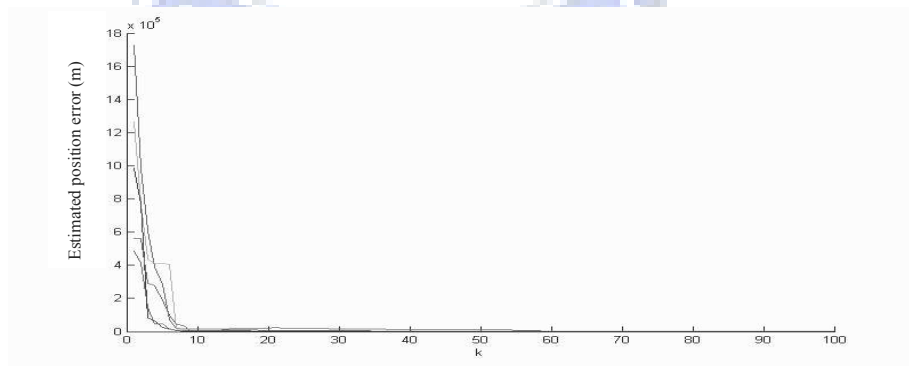
Figure 4.13 Performance of 5 runs with the same initial weights using the NSOMS, SOMSO-1, and SOMSO-2.



(a) SOMSO-1



(b) SOMSO-2



(c) NSOMS

Figure 4.14 Performance of 5 runs with the different initial weights using the NSOMS, SOMSO-1, and SOMSO-2.

Chapter 5

Conclusion

In this dissertation, we have proposed an SOM-based search algorithm (SOMS), which can be used for both static and dynamic functions in real time. To achieve high learning efficiency for system parameters in different working ranges, we have also proposed a new SOM weight updating rule. An intelligent radar predictor for trajectory estimation is first developed for application. With a simplified target dynamic model, the unsupervised SOM in the predictor can achieve salient prediction in noisy, unknown environments. The SOM is more robust to the uncertainty of the dynamic model than Kalman filter and GA. Furthermore, the SOM's search abilities have been adequately exploited in a multimodal domain. A new niche method (deterministic competition) to extend the ability of the SOM-based search algorithm has been proposed for identification of multiple optimal solutions. In order to reduce the network size, another new SOM weight updating rule is proposed to enhance the learning efficiency. With the dynamic weight updating, the NSOMS converges faster than other algorithms such as SOMO, KSOM-ES, SOSENs and RCS-PSM. Moreover, a new adaptive mapping model is proposed to visualize the distribution and structure of the optimal solutions into the 2D neuron space. In our proposed

NSOMS, only two learning parameters need to be determined in the weight and position updating rules. The applications of the proposed NSOMS on both function optimization in a multimodal domain and dynamic trajectory predictions involving multiple targets have clearly demonstrated its effectiveness.

5.1 Future Research

In this dissertation, by combining the SOM with the dynamic model, the SOM is able to tackle the spatiotemporal data. The SOM has been applied to search optimal parameters for dynamic systems. To further exploit its search ability, in one of the future works, we will apply the NSOMS for system identification and control problems. Because the current searching process includes the weight updating rules and the parameters of learning, it is not easy to appropriate the learning rate, number of neurons, and termination criteria. Although these parameters can be selected through a trial-and-error process, the time response of the learning affects the performance of the dynamic systems for system identification and control problems. Thus, we will also discuss the convergence issue in details. As the SOM also possesses an appealing feature in responding to distinct properties exhibited by the input data through forming several corresponding clusters, another worthwhile future work will be to extend the proposed NSOMS for a wide application such as image processing, speaker recognition, machine learning, and others.

Bibliography

- [1] A. P. Azcarraga, T. N. Yap, Jr., J. Tan, and T. S. Chua, "Evaluating Keyword Selection Methods for WEBSOM Text Archives," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 16(3), pp. 380-383, 2004.
- [2] G. A. Barreto and A. F. R. Araujo, "Identification and Control of Dynamical Systems Using the Self-Organizing Map," *IEEE Trans. on Neural Networks*, Vol. 15(5), pp. 1244-1259, 2004.
- [3] A. G. Barto, "Reinforcement Learning and Adaptive Critic Methods," *Handbook of Intelligent Control*, White and Sofge, eds., Van Nostrand-Reinhold, New York, pp. 469-491, 1992.
- [4] G. A. Carpenter and S. Grossberg, "The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network," *IEEE Computer*, Vol. 21(3), pp. 77-88, 1988.
- [5] C.B. Chang and J. A. Tabaczynski, "Application of State Estimation to Target Tracking," *IEEE Trans. on Automatic Control*, Vol. 29(2), pp. 98-109, 1984.
- [6] L. Chin, "Application of Neural Networks in Target Tracking Data Fusion," *IEEE Trans. on Aerospace and Electronic Systems*, Vol. 30(1), pp. 281-287, 1994.
- [7] Y. Y. Chen and K. Y. Young, "An Intelligent Radar Predictor for Hight-speed Moving-target Tracking," *International J. Fuzzy Systems*, Vol. 6(2), pp. 90-99, 2004.

- [8] A. D. Cioppa, C. D. Stefano, and A. Marcelli, "On the Role of Population Size and Niche Radius in Fitness Sharing," *IEEE Trans. on Evolutionary Computation*, Vol. 8(6), pp. 580-592, 2004.
- [9] M. Efe and D. P. Atherton, "Maneuvering Target Tracking with an Adaptive Kalman Filter," *IEEE Conference on Decision and Control*, pp. 737-742, 1998.
- [10] N. Eldredge and S. J. Gould, "Punctuated equilibria: an alternative to phyletic gradualism", *Models in Paleobiology*, pp. 82-115, 1972.
- [11] D. E. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," *Proceedings of 2nd ICGA*, pp. 41-49, 1987.
- [12] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, New York, 1989.
- [13] M. Hagenbuchner and A. C. Tsoi, "A Supervised Self-Organizing Map for Structures," *IEEE Conference on Neural Networks*, pp. 1923-1928, 2004.
- [14] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan, New York, 1994.
- [15] H. Igarashi, "Visualization of Optimal Solutions Using Self-Organizing Maps in Computational Electromagnetism," *IEEE Trans. on Magnetism*, Vol. 41(5), pp. 1816-1819, 2005.
- [16] H.-D. Jin, K.-S. Leung, M.-L. Wong, and Z.-B. Xu, "An Efficient Self-Organization Map Designed by Genetic Algorithms for the Traveling Salesman Problem," *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 33(6), pp. 877-888, 2003.

- [17] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," *IEEE Int. Conference on Neural Networks*, pp. 1942-1948, 1995.
- [18] J. K. Kim, D. H. Cho, H. K. jung, and C. G. Lee, "Niching Genetic Algorithm Adopting Restricted Competition Selection Combined with Pattern Search Method", *IEEE Trans. on Magnetics*, vol. 38(2), pp. 1001-1004, 2002.
- [19] K. J. Kim and S. B. Cho, "Fusion of Structure Adaptive Self-Organizing Maps Using Fuzzy Integral," *IEEE Conference on Neural Networks*, pp. 28-33, 2003.
- [20] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, Vol. 220, pp. 671-680, 1983.
- [21] T. Kirubarajan, H. Wang, Y. Bar-Shalom, and K. R. Pattipati, "Efficient Multisensor Fusion Using Multidimensional Data Association," *IEEE Trans. on Aerospace and Electronic Systems*, Vol. 37(2), pp. 386-400, 2001.
- [22] T. Kohonen, *Self-Organizing Map*, Springer, Berlin, Germany, 1997.
- [23] D.-C. Liaw, Y.-W Liang, and C.-C. Cheng, "Nonlinear Control for Missile Terminal Guidance," *ASME J. Dynamic Systems, Measurement, and Control*, Vol. 122(4), pp. 663-668, 2000.
- [24] J. Laaksonen, M. Koskela, and E. Oja, "PicSOM — Self-Organizing Image Retrieval with MPEG-7 Content Descriptors," *IEEE Trans. on Neural Networks*, Vol. 13(4), pp. 841-853, 2002.
- [25] R. Storn and K. Price, "Differential Evolution - A simple and efficient global optimization over continuous spaces," *Journal of Global Optimization*, Vol 11. pp 341-359, 1997.

- [26] T. M. Martinetz, H. J. Ritter, and K. J. Schulten, "Three-Dimensional Neural Net for Learning Visuomotor Coordination of a Robot Arm," *IEEE Trans. on Neural Networks*, Vol. 1(1), pp. 131-136, 1990.
- [27] R. L. Moose, H. F. Vanlandingham, and D. H. McCabe, "Modeling and Estimation for Tracking Maneuvering Targets," *IEEE Trans. on Aerospace and Electronic Systems*, Vol. 15(3), pp. 448-456, 1979.
- [28] S. W. Mahfoud, "Niching Methods for Genetic Algorithms," PhD thesis, University of Illinois at Urbana Champaign, 1995.
- [29] M. Milano, P. Koumoutsakos, and J. Schmidhuber, "Self-Organizing Nets for Optimization," *IEEE Trans. on Neural Networks*, Vol. 15(3), pp. 758-765, 2004.
- [30] K. Obermayer and T. J. Sejnowski, ed., *Self-Organizing Map Formation: Foundation of Neural Computation*, MIT Press, Cambridge, 2001.
- [31] J. C. Principe, L. Wang, and M. A. Motter, "Local Dynamic Modeling with Self-Organizing Maps and Applications to Nonlinear System Identification and Control," *Proceedings of the IEEE*, Vol. 86(11), pp. 2240-2258, 1998.
- [32] K. V. Ramachandra, "A Kalman Tracking Filter for Estimating Position, Velocity and Acceleration from Noisy Measurements of a 3-D Radar," *Electro Technology*, Vol. 33, pp. 66-76, 1989.
- [33] J. M. Roberts, D. J. Mills, D. Charnley, and C. J. Harris, "Improved Kalman Filter Initialisation Using Neurofuzzy Estimation," *International Conference on Artificial Neural Networks*, pp. 329-334, 1995.

- [34] B. Sareni, L. Krahenbuhl, and A. Nicolas, "Niching genetic algorithm for optimization in electromagnetics I. Fundamentals," *IEEE Trans. on Magnetics*, Vol. 34(5), pp. 2984-2991, 1998.
- [35] B. Sareni and L. Krahenbuhl, "Fitness Sharing and Niching Methods Revisited," *IEEE Trans. on Evolutionary Computation*, Vol. 2(3), pp. 97-106, 1998.
- [36] D. Sbarbaro and D. Bassi, "A Nonlinear Controller Based on Self-Organizing Maps," *IEEE Int. Conference on Systems, Man and Cybernetics*, pp. 1774-1777, 1995.
- [37] H. Shah-Hosseini and R. Safabakhsh, "TASOM: a New Adaptive Self-Organization Map," *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 33(2), pp. 271-282, 2003.
- [38] M. C. Su and H. T. Chang, "Fast Self-Organizing Feature Map Algorithm," *IEEE Trans. on Neural Networks*, Vol. 11(3), pp. 721-733, 2000.
- [39] M. C. Su and H. T. Chang, "A New Model of Self-Organizing Neural Networks and its Application in Data Projection," *IEEE Trans. on Neural Networks*, Vol. 12(1), pp. 153-158, 2001.
- [40] M. C. Su, Y. X. Zhao, and J. Lee, "SOM-Based Optimization," *IEEE Int. Conference on Neural Networks*, pp. 781-786, 2004.
- [41] J.A. Vasconcelos, R. R. Saldanha, L. Krahenbuhl, A. Nicolas, "Genetic Algorithm Coupled with a Deterministic Method for Optimization In Electromagnetics," *IEEE Trans. on Magnetics*, Vol.33(2), pp. 1860-1863, 1997.
- [42] J. A. Walter and K. I. Schulten, "Implementation of Self-Organizing Neural Networks for Visuo-Motor Control of an Industrial Robot," *IEEE Trans. on Neural Networks*, Vol. 4(1), pp. 86-96, 1993.

- [43] S. Wu and T. W. S. Chow, "PRSOM: a New Visualization Method by Hybridizing Multidimensional Scaling and Self-Organizing Map," *IEEE Trans. on Neural Networks*, Vol. 16(6), pp. 1362-1380, 2005.
- [44] S. Wu and T. W. S. Chow, "Self-Organizing and Self-Evolving Neurons: A New Neural Network for Optimization," *IEEE Trans. on Neural Networks*, Vol. 18(2), pp. 385-396, 2007.
- [45] P. Xu, C. H. Chang, and A. Paplinski, "Self-Organizing Topological Tree for On-line Vector Quantization and Data clustering," *IEEE Trans. on Systems, Man and Cybernetics, Part B: Cybernetics*, Vol. 35(3), pp. 515-526, 2005.
- [46] C.-D. Yang, F.-B. Hsiao, and F.-B. Yeh, "Generalized Guidance Law for Homing Missles," *IEEE Trans. on Aerospace and Electronic Systems*, Vol. 25(2), pp. 197-212, 1989.
- [47] H. Yin, "ViSOM - a Novel Method for Multivariate Data Projection and Structure Visualization," *IEEE Trans. on Neural Networks*, Vol. 13(1), pp. 237-243, 2002.

Vita

姓名：陳一元 (Yi-Yuan Chen)

研究方向：類神經網路，模糊系統，機器學習演算法，機器人控制.

出生地：台灣南投縣.

戶籍地址：桃園縣楊梅鎮三民北路 163 巷 2 弄 1 號.

電子郵件：yiyuan.ece90g@nctu.edu.tw

學經歷：

1. 81 年 9 月～84 年 6 月 國立彰化師大附屬高級工業學校
2. 84 年 9 月～88 年 6 月 私立淡江大學電機工程學系
3. 88 年 9 月～90 年 6 月 國立交通大學電機與控制工程研究所
4. 90 年 9 月～97 年 7 月 國立交通大學電機與控制工程學系博士學位
5. 97 年 3 月～ now 工業技術研究院 辨識中心 工程師

發表著作：

期刊論文

- [1] Y.Y. Chen and K.Y. Young, "An SOM-Based Algorithm for Optimization with Dynamic Weight Updating," International Journal of Neural Systems, Vol. 17(3), pp. 171-181, 2007.
- [2] Y.Y. Chen and K.Y. Young, "An Intelligent Radar Predictor for High-Speed Moving-Target Tracking," International Journal of Fuzzy Systems, Vol. 6(2), pp. 90-99, 2004.

國際會議論文

- [1] Y.Y. Chen and K.Y. Young, "An SOM-Based Search Algorithm for Dynamic Systems," 9th Joint Conference on Information Sciences, pp. 1212-1215, 2006.
- [2] Y.Y. Chen and K.Y. Young, "Applying SOM as a Search Mechanism for Dynamic System," pp. 4111-4116, IEEE Conference on Decision and Control, 2005.
- [3] Y.Y. Chen and K.Y. Young, "An Intelligent Radar Predictor for High-Speed Moving-Target Tracking," IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering Proceedings, pp. 1638-1641, 2002.

國內會議論文

- [1] Y.Y. Chen and K.Y. Young, "智慧型雷達預估器於追蹤高速運動目標之研究," Ninth National Conference on Fuzzy Theory and its Applications, pp. 598-603, 2001.
- [2] Y.Y. Chen and K.Y. Young, "智慧型雷達預估器於高速多目標之追蹤," Tenth National Conference on Fuzzy Theory and its Applications, pp. 5-10, 2002.