Lyapunov Analysis of Neural Network Stability in an Adaptive Flight Control System *

Sampath Yerramalla[1], Edgar Fuller[2], Martin Mladenovski[1], and Bojan Cukic[1]

[1] Lane Department of Computer Science and Electrical Engineering
West Virginia University
Morgantown WV 26506
{sampath, cukic, martinm}@csee.wvu.edu
[2] Department of Mathematics
West Virginia University
Morgantown WV 26506
ef@math.wvu.edu

Abstract. The paper presents the role of self-stabilization analysis in the design, verification and validation of the dynamics of an *Adaptive Flight Control System* (AFCS). Since the traditional self-stabilization approaches lack the flexibility to deal with the continuous adaptation of the neural network within the AFCS, the paper emphasizes an alternate self-stability analysis approach, namely *Lyapunov's Second Method*. A Lyapunov function for the neural network is constructed and used in presenting a formal mathematical proof that verifies the following claim: While learning from a *fixed* input manifold, the neural network is self-stabilizing in a *Globally Asymptotically Stable* manner. When dealing with *variable* data manifolds, we propose the need for a real-time stability monitor that can detect unstable state deviations. The test results based on the data collected from an F-15 flight simulator provide substantial heuristic evidence to support the idea of using a Lyapunov function to prove the self-stabilization properties of the neural network adaptation.

1 Introduction

1.1 Background

Adaptive control is the latest trend in the application of Neural Networks (NN) in realtime automation, one of the world's leading markets for computer control systems. The concept of Neural Adaptive Flight Control is perhaps the most challenging of them all as constructing it with guaranteed stability that ensures peak performance of the aircraft requires a thorough understanding of the objective functions [19]. Qualitatively speaking, an adaptive flight control system that has the ability to sense its environment, process information, reduce uncertainty,

* This work was supported in part by NASA through cooperative agreement NCC 2-979. The opinions, findings, conclusions and recommendations expressed herein are those of the authors and do not reflect the views of the sponsors.

plan, generate and execute control actions is considered an Intelligent Flight Control System (IFCS) [16, 17]. The goal of IFCS is to develop and flight evaluate flight control concepts that incorporate emerging algorithms and techniques to provide an extremely robust system capable of handling multiple accident and off-nominal flight scenarios [12]. Figure 1 shows the architectural overview of an IFCS consisting of an Online Learning Neural Network (OLNN) that accounts for dramatic changes of the aircraft exceeding robustness limits [22].

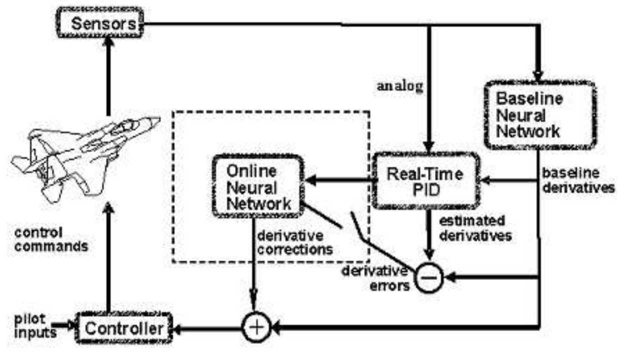

Fig. 1. An Intelligent Flight Control System Architecture

A Baseline or Pre-trained Neural Net (PNN) is used for replacing linear maps from standard designs (like reference adaptive control) in order to produce a robust system that can handle nonlinearities. PNN is non-adaptive, meaning that once trained, it does not change during the flight. It is essentially a table-lookup scheme based on wind-tunnel experiments for the stable aircraft conditions. The adaptive nature of the IFCS is induced by the realtime Parameter Identification (PID), that uses an equation error technique employing Fourier Transform Regression algorithm. During off-nominal flight conditions (combat damage or major system failure), notable discrepancies between the outputs of the PNN and the real-time PID are generated that need to be accounted by the OLNN. This difference in output of the PNN and PID is termed as Stability and Control Derivative (SCD) error. The role of the OLNN is to store these SCD errors and provide a better estimate of the stored values for new flight conditions. The SCD outputs from the BLNN along with the estimated SCD errors form the OLNN ensure that the controller is able to compute control gains needed to make the aircraft stable and, consequently, safe.

Adaptive systems are hard to model as they are often accompanied with difficulties: many degrees of freedom, distributed sensors, high noise levels, and uncertainty. Nevertheless, greater the ability of the system to deal with these difficulties, the more intelligent is the system [17]. The central control goal of the IFCS is to calculate the present state of the system and determine a strategy to drive the system to a desired final state. If the system contains a provably self-stabilizing on-line learning neural network, it ensures that while IFCS achieves its central goal, states of the system do not deviate towards instability, thereby avoiding a potential disaster. However, verification of self-stabilization properties is usually a complicated task [13]. Our research goal is all the more challenging since we need to verify self-stabilization properties of a neural network which is able to adapt during the flight, potentially having direct consequences for the overall system safety.

1.2 Limitations of Traditional Self-stabilization Proof Techniques

The original idea of self-stabilizing systems was introduced by Dijkstra in 1974 [8]. Traditional methodologies for the analysis of self-stabilization require a detailed system understanding, needed in the definition of an *invariance* condition. The role of an invariant function in self-stabilization has been described by Arora in [1]: *if the system is initiated appropriately, the invariant is always satisfied. If the system is placed in an arbitrary state to continue execution, it eventually reaches a state from where the invariant is always satisfied.*

The self-stabilization proof methodology applicable to distributed algorithms exhibits interesting analogies with feedback system stabilization in control engineering [23]. These analogies are based on the existence of invariants, but the proof mechanisms differ significantly. Therefore, very early in our research we had to ask ourselves whether it would be possible to apply any of the existing methodologies to prove the self-stabilization properties of a continuously evolving, topologically complex neural network embedded in an adaptive flight control environment.

A major practical limitation of application of traditional methodologies for proving self-stabilization properties of neural networks appears to be the *scalability* [2]. Standard approaches are based on the definition of the invariance. But in order to define the invariance, a detailed system description must be available. However, the goal of using an adaptive component in a flight control system is to cope with unanticipated and/or failure conditions. If these conditions were to be systematically described, the excessive size of the data (and the ensuing system designs) would make computing an invariance a complicated task. The second limitation of traditional approaches is their inability to handle *uncertainties*, regularly encountered in adaptive systems. But, even if we assume that uncertainty and scalability are of no concern, the challenge still remains: *Do traditional stabilization proof techniques work for adaptive systems?*

It should not come as a surprise that we discovered the need to define a different notion of self-stabilization and embarked on developing alternative verification techniques to reason about the self-stabilization properties of the specific type of neural network used in the intelligent flight control system.

1.3 Overview

The rest of the paper is organized as follows. Section 2 introduces the Lyapunov approach to self-stability analysis of dynamic systems. Section 3 describes the specific type of neural networks, so called Dynamic Cell Structures (DCS), used in the context of the adaptive flight control system. DCS neural networks are the object of our stability analysis. Section 4 presents our main result, formal stability proof for the DCS neural network. This proof deals with the flight conditions that exclude unexpected environmental conditions (failures). Therefore, Section 5 introduces the notion of on-line stability monitoring, which can provide early warnings by detecting the states leading to unstable system conditions. These concepts are further described in a case study involving the flight data collected in an F-15 flight simulator in Section 6. We conclude by a brief summary in Section 7.

2 Self-Stabilization Using Lyapunov Functions

Often, the mathematical theory of stability analysis is (mis)understood as the process of finding a solution for the differential equation(s) that govern the system dynamics. Stability analysis is the theory of validating the existence (or non-existence) of stable states. Theoretically, there is no guarantee for the existence of a solution to an arbitrary set of nonlinear differential equations, let alone the complicated task of solving them [7].

In the context of the analysis of the stability of adaptation in the IFCS, the problem is to find an effective tool applicable to the online learning neural network. It is often seen that adaptive systems that are stable under some definitions of stability tend to become unstable under other definitions [9]. This difficulty in imposing strong stability restrictions for nonlinear systems was realized as early as a century ago by a Russian mathematician A. M. Lyapunov. Details on Lyapunov's stability analysis technique for nonlinear discrete systems can be found in [4,9,27]. The fact that Lyapunov's *direct method* or Lyapunov's *second method* can be easily and systematically applied to validate the existence (or non-existence) of stable states in an adaptive system, intrigued us in using Lyapunov's concept of self-stabilization in our analysis as a means of answering the question posed earlier.

In the discrete sense, Lyapunov stability can be defined as follows:

Definition 1. *Lyapunov Stability*
If there exists a Lyapunov function, $V : \mathbb{R}^O \to \mathbb{R}$, defined in a region of state space near a solution of a dynamical system such that

1. $V(0) = 0$
2. $V(x) > 0 : \forall x \in O, x \neq 0$
3. $V(x(t_{i+1})) - V(x(t_i)) = \Delta V(x) \leq 0 : \forall x \in O$

then the solution of the system is said to stable in the sense of Lyapunov.

$x = 0$ represents a solution of the dynamical systems and $\mathbb{R}^O$, O represent the output space and a region surrounding this solution of the system respectively. Though this concept was intended for mathematics of control theory, Lyapunov stabilization in a general sense can be simplified as follows. A system is said to be stable near a given solution if one can construct a Lyapunov function (scalar function) that identifies the regions of the state space over which such functions decrease along some smooth trajectories near the solution.

Definition 2. ***Asymptotic Stability (AS)***
If in addition to conditions 1 and 2 of Definition 1, the system has a negative-definite Lyapunov function

$$\Delta V(x) < 0 : \forall x \in O \tag{1}$$

then the system is Asymptotically Stable.

Asymptotic stability adds the property that in a region surrounding a solution of the dynamical system trajectories are approaching this given solution asymptotically.

Definition 3. ***Global Asymptotic Stability (GAS)***
If in addition to conditions 1 and 2 of Definition 1, the Lyapunov function is constructed such that,

$$\lim_{t \to \infty} V(x) = 0 \tag{2}$$

over the ***entire*** *state space then the system is said to be Globally Asymptotically Stable.*

A notable difference between AS and GAS is the fact that GAS implies any trajectory beginning at *any* initial point will converge asymptotically to the given solution, as opposed to AS where only those trajectories begining in the neighborhood of the solution approach the solution asymptotically. The types of stability defined above have increasing property strength, i.e.

$$\text{Global Asymptotic Stability} \Longrightarrow \text{Asymptotic Stability} \Longrightarrow \text{Lyapunov Stability.}$$

The reverse implication does not necessarily hold as indicated by the Venn diagram of Figure 2. In simple terms, the system is stable if all solutions of the state that start *nearby* end up *nearby*. A good distance measure of *nearby* must be defined by a Lyapunov function (V) over the states of the system. By constructing V, we can guarantee that all trajectories of the system converge to a stable state. The function V should be constructed keeping in mind that it needs be scalar ($V \in \mathbb{R}$) and should be non-increasing over the trajectories of the state space. This is required in order to ensure that all *limit points* of any trajectory are stationary. Thus a strict Lyapunov function should force every trajectory to asymptotically approach an equilibrium state. Even for non-strict

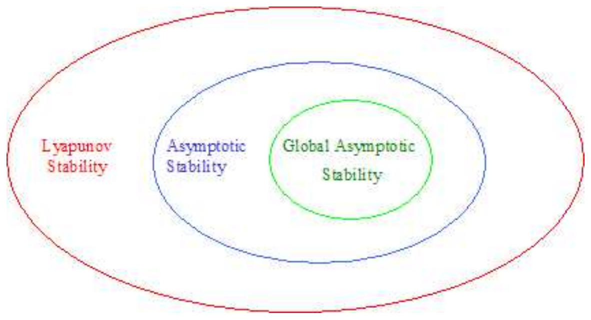

Fig. 2. Relative strengths of Stability

Lyapunov functions it is possible to guarantee convergence by LaSalle's invariance principle. In mechanical systems a Lyapunov function is considered as an energy minimization term, in economy and finance evaluations it is considered as a cost-minimization term, and for computational purposes it can be considered as an error-minimization term.

3 DCS Neural Network

In 1994 Jorg Bruske and Gerald Sommer of the University of Kiel, Germany, introduced the concept of DCS as a family of topology representing self-organizing NN [3, 5]. This Topology Preserving Feature Map (TPFM) generation scheme was motivated by Growing Neural GAS algorithm developed by Fritzke and the former work on Topology Representing Networks (TRN) by Martinetz [15].

DCS uses Kohonen-like adaptation to shift the weighted centers of the local neighborhood structure, closer to the feature manifold. When applied in conjunction with Competitive Hebb Rule (CHR) to update lateral connections of the neighbors, this produces a network representation that preserves the features of the input manifold. These two essential building blocks (rules) of the DCS algorithm, play a key role in the generation of a TPFM. Before we proceed for an in-depth analysis of the DCS algorithm we need to study and formulate these competitive rules that govern the DCS dynamical system.

3.1 Competitive Hebb Rule (CHR)

DCS NN rests upon a Radial Basis Function (RBF) and an additional layer of lateral connections between the neural units [3]. These lateral connection strengths are symmetric and bounded in nature, $c_{ij} = c_{ji} \in [0, 1]$. The goal of CHR is to update the lateral connections by mapping neighborhoods in the input manifold to neighborhoods of the network. Thereby, avoiding any restrictions of the topology of the network [5]. For each input element of the feature manifold, CHR operates by setting the connection strength between the two neural units that are closer to the input than any other neuron pair, to a highest possible connection strength of 1. These two neural units are referred to as the Best Matching Unit (*bmu*) and Second Best Unit (*sbu*). CHR then proceeds to decay the strength of all other existing lateral connections emanating from the *bmu* using a forgetting constant, α. If any of these existing connections drop below a predefined threshold θ, they are set to zero. The set of all neural units that are connected to the *bmu* is defined as the neighborhood of the *bmu*, and represented by *nbr*. All other connections of the network remain unaltered. In this way CHR induces a Delaunay triangulation into the network by preserving the neighborhood structure of the feature manifold.

$$c_{ij}(t+1) = \begin{cases} 1 & (i = bmu) \wedge (j = sbu) \\ 0 & (i = bmu) \wedge (j \in nbr - sbu) \wedge (c_{ij} < \theta) \\ \alpha c_{ij}(t) & (i = bmu) \wedge (j \in nbr - sbu) \wedge (c_{ij} \geq \theta) \\ c_{ij}(t) & i, j \neq bmu \end{cases} \tag{3}$$

It was shown in [15] that algorithms utilizing CHR to update lateral connections between neural units generate a TPFM.

3.2 Kohonen-Like Rule (KLR)

Unlike a typical feed-forward NN, the weight center $\boldsymbol{w_i}$ associated with a neural unit i of the DCS network represents the location of the neural unit in the output space. It is crucial to realize that these weighted centers be updated in a manner that preserves the geometry of the input manifold. This can be achieved by adjusting the weighted center of the *bmu* and its surrounding neighborhood structure *nbr* closer to the input element. For each element of the feature manifold, $\boldsymbol{u} \in I$, DCS adapts the corresponding *bmu* and its neighborhood set *nbr* in a Kohonen-like manner [18]. Over any training cycle, let $\boldsymbol{\Delta w_i = w_i(t+1) - w_i(t)}$ represent the adjustment of the weight center of the neural unit, then the Kohonen-like rule followed in DCS can be represented as follows

$$\Delta w_i = \begin{cases} \varepsilon_{bmu}(\boldsymbol{u} - \boldsymbol{w_i(t)}) & i = bmu \\ \varepsilon_{nbr}(\boldsymbol{u} - \boldsymbol{w_i(t)}) & i \in nbr \\ 0 & (i \neq bmu) \wedge (i \notin nbr) \end{cases} \tag{4}$$

where $\varepsilon_{bmu}, \varepsilon_{nbr} \in [0,1]$ are predefined constants known as the learning rates that define the momentum of the update process. For every input element, applying CHR before any other adjustment ensures that *sbu* is a member of *nbr* set for all further adjustments within the inner loop of the DCS algorithm.

3.3 Growing the Network

Unlike traditional Self-Organizing Maps (SOM), DCS has the ability to grow or shrink the map by increasing or decreasing the number of neurons of the network. A local error measure associated with the network, namely *Resource*, is used to determine if the network experienced a large enough cumulative error, meaning there is a requirement for an additional neuron in the network. In most cases Euclidean distance between the best matching unit (*bmu*) and the training input stimulus serves as a measure of the resource. After a cycle of adaptation (epoch), if needed an additional neuron is introduced into the network at the region between the highest and second highest resource neurons of the network.

3.4 DCS algorithm

Knowing the operational aspects of the individual building blocks, we now analyze the DCS training algorithm. As shown in Figure 3, the DCS algorithm is allowed to train on the input stimulus until the network has reached a specific stopping criteria. For each training input stimulus, the network is searched for

```
while stopping criteria is not satisfied
{
    for each training input stimulus
    {
            find bmu, sbu
            update connections using CHR
            adapt weights using KLR
            update resource error
    }
    compute cumulative network resource error
    if (cumulative network resource error) > (Predefined Error)
    {
            grow the network
            decrement all resource values
    }
}
```

Fig. 3. DCS Algorithm

the two closest neurons, best matching unit (*bmu*) and second best unit (*sbu*). The lateral connection structure surrounding the *bmu* is updated using Hebb rule. Kohonen adaptation of the weights of the *bmu* and its neighbors (*nbr*) is performed.

The resource value of the *bmu* is updated correspondingly, marking the end of a training cycle (epoch). The cumulative resource error of the network is computed to determine the need for inserting an additional neuron into the network. Decreasing the resource values of all the neurons by a decay constant prevents the resource values from growing out of bounds.

We want to determine if the DCS algorithm will *reliably* learn a fixed input manifold $I \subset \mathbb{R}^I$, on successive applications. The question is then how much can the evolving state of the DCS algorithm, denoted by x_t, deviate from a previously learned stable state, denoted by x_0? Stability analysis in similar terms was performed by Kaynak et. al. for a backpropagation neural network that has a relatively simple topological structure in comparison with a DCS network [26].

A network basically consists of nodes (weights) and vertices (connections). Thereby, it can be completely represented using a weight center matrix, $W \subset \mathbb{R}^O$, and a connection strength matrix, $C \subset \mathbb{R}^O$. Considering the map generated by the DCS network as a graph, $G(W, C)$, we provide the following definition.

Definition 4. ***DCS Network's Mapping***
DCS network's mapping $G(W, C)$ for a given feature manifold, $I \subset \mathbb{R}^I$, is an N^{th} order neural network representation of I in the output space, $O \subset \mathbb{R}^O$, generated by assigning N neural units in t_n steps of the DCS algorithm.

This definition characterizes the DCS process as a mapping which we can now proceed to analyze in the context of discrete dynamical systems.

4 Self-stabilization of DCS network

4.1 State Space of the DCS Network

Let the DCS network be learning from a set of training examples of the input space $\boldsymbol{u(t)} \in \mathbb{R}^I$. At some point of time during or after the learning, if a test input stimulus, $\boldsymbol{u^{\star}(t)} \in \mathbb{R}^I$ is presented, the network generates an estimated output, $\hat{\boldsymbol{y}} = G(W(t), C(t), \boldsymbol{u}^{\star}(t))$. The dynamics of such learning can be represented as:

$$\begin{aligned} \dot{\boldsymbol{x(t)}} &= f(\boldsymbol{x(t)}, \boldsymbol{u(t)}, t) : X \times \mathbb{R}^I \times \mathbb{R} \\ \hat{\boldsymbol{y}} &= G(W(t), C(t), \boldsymbol{u^{\star}(t)}) : \mathbb{R}^O \times \mathbb{R}^O \times \mathbb{R}^I \end{aligned} \tag{5}$$

where $\boldsymbol{x(t)}$ represents the DCS networks learning state, X represents the learning profile. For the sake of simplicity we consider only discrete time variations. Specifically, whenever t is such that $t_i \leq t < t_{i+1}$ we will have $f(\boldsymbol{x(t)}, \boldsymbol{u(t)}, t) = f(\boldsymbol{x(t)}, \boldsymbol{u(t)}, t_i)$. This implies that the DCS network learning depends only on x and u. The dynamics from (5) can be re-written as:

$$\begin{aligned} \boldsymbol{\Delta x} &= \boldsymbol{x(t_{i+1})} - \boldsymbol{x(t_i)} = f(\boldsymbol{x}, \boldsymbol{u}) \\ y(\boldsymbol{u}^{\star}) &= G(W, C, \boldsymbol{u}^{\star}) \end{aligned} \tag{6}$$

4.2 Mathematical Verification of Self-stabilization

Theorem 1. *During DCS network's representation of a fixed input manifold, the evolving state of the system due to neural unit's position adjustment, $x_W \in W$, is self-stabilizing in a globally asymptotically stable manner.*

Proof. Since the DCS network's learning is a time-varying process whose state changes according to the difference relations (6), we first need to construct a Lypaunov function that is non-increasing over the state trajectories.

To set up a Lyapunov function, we measure how accurately a current state of the algorithm models the input manifold in terms of the amount of geometry being preserved by the network. We formulate a function that measures the effectiveness of the placement of neural units by computing the amount by which a neural unit deviates from an element $\boldsymbol{u} \in I$ of the input manifold for which it is the best matching unit. We then average this over the total number of neurons, N, in that phase of DCS approximation to get

$$V = \frac{1}{N} \sum_{\boldsymbol{u} \in I} \|\boldsymbol{u} - \boldsymbol{w}_{\boldsymbol{bmu(u,t)}}\| \tag{7}$$

Throughout this proof, V, commonly referred as the network's quantization error serves as the network's Lyapunov function.

First of all we need to show that the above presented Lyapunov function (V) is valid. It is obvious that $V \geq 0$ since $\|u - w_{bmu(u,t)}\| \geq 0$. Also $V(0) = 0$, since there are no neurons in the DCS network during zero state. To show that $\frac{\Delta V}{\Delta t} < 0$, first note that since the time step $\Delta t > 0$, the numerator ΔV will determine the sign in question. Over any learning cycle, DCS network adjusts the weights of the *bmu* and its neighbors, *nbr*, according to the Kohonen-like rule. Let $|\boldsymbol{u} - \boldsymbol{w}_{\boldsymbol{bmu(u,t)}}\|$ be represented by $d(\boldsymbol{u}, t)$, then we see that over a single training cycle

$$\begin{aligned} \Delta V &= \frac{1}{N+1} \sum_{\boldsymbol{u} \in I} d(\boldsymbol{u}, t+1) - \frac{1}{N} \sum_{\boldsymbol{u} \in I} d(\boldsymbol{u}, t) \\ &= \frac{N \sum_{\boldsymbol{u} \in I} d(\boldsymbol{u}, t+1) - (N+1) \sum_{\boldsymbol{u} \in I} d(\boldsymbol{u}, t)}{N(N+1)} \\ &= \frac{N \sum_{\boldsymbol{u} \in I} (d(\boldsymbol{u}, t+1) - d(\boldsymbol{u}, t)) - \sum_{\boldsymbol{u} \in I} d(\boldsymbol{u}, t)}{N(N+1)} \end{aligned} \tag{8}$$

For any $\boldsymbol{u} \in I$, we need to show that either the corresponding portion of the numerator is negative or that some other portion compensates when it is positive. There are three ways in which a neural unit's weighted center may get adjusted. It may get updated as

1. as the *bmu* for $\boldsymbol{u}$,
2. as the *bmu* of some other $\boldsymbol{u'} \in I$,
3. or as one of the neighbors of the *bmu* of some other $\boldsymbol{u'} \in I$.

In the first case, to show that equation (8) evaluates to less than 0, it is sufficient to show that

$$d(\boldsymbol{u}, t+1) - d(\boldsymbol{u}, t) < 0 \tag{9}$$

Computation using the weight-adaptation rule followed in DCS network gives

$$
\begin{aligned}
&d(\boldsymbol{u}, t+1) = \|\boldsymbol{u} - w_{bmu(\boldsymbol{u},t+1)}\| \\
&= \|\boldsymbol{u} - (w_{bmu(\boldsymbol{u},t)} + \varepsilon_{bmu}(\boldsymbol{u} - w_{bmu(\boldsymbol{u},t)}))\| \\
&= \|(\boldsymbol{u} - w_{bmu(t)}) - \varepsilon_{bmu}(\boldsymbol{u} - w_{bmu(t)})\| \\
&= \|\boldsymbol{u} - w_{bmu(t)} - \varepsilon_{bmu}\boldsymbol{u} + \varepsilon_{bmu} w_{bmu(t)})\| \\
&= (1 - \varepsilon_{bmu})\|\boldsymbol{u} - w_{bmu(t)}\| \\
&= (1 - \varepsilon_{bmu}) d(\boldsymbol{u}, t).
\end{aligned}
$$

Since $\varepsilon_{bmu} \in (0, 1)$,

$$d(\boldsymbol{u}, t+1) < d(\boldsymbol{u}, t),$$

which implies (9).

In the second case, the best matching unit for $\boldsymbol{u} \in I$, $bmu(\boldsymbol{u})$, may get updated as the best matching unit for some other $\boldsymbol{u'} \in I$, $bmu(\boldsymbol{u'})$. The update as $bmu(\boldsymbol{u'})$ can either precede or follow the update as $bmu(\boldsymbol{u})$ and the effect on V depends primarily on which input stimulus is closest to $bmu(\boldsymbol{u}) = bmu(\boldsymbol{u'})$. When $\boldsymbol{u}$ is farthest from $bmu(\boldsymbol{u})$, the triangle inequality for the euclidean metric implies that $d(t) > d(t+1)$ regardless of the order of update. On the other hand, if $\boldsymbol{u'}$ is farther from $bmu(\boldsymbol{u})$, $d(t)$ may be smaller than $d(t+1)$ but any increase in the distance from $\boldsymbol{u}$ to its bmu is smaller than the decrease in the distance from u' to the same neuron since $\boldsymbol{u}$ is closer to $bmu(\boldsymbol{u})$. Again, this follows from the triangle inequality for the euclidean metric. The net effect in all cases is a decrease in V.

The third case will have $\Delta V < 0$ since $\varepsilon_{bmu} >> \varepsilon_{sbu}$ in general. In case the two values are comparable, the result follows from the same argument as in the second case above. As a result, the function V is a Lyapunov function for the state of position adjustments of the DCS network and furthermore since $\Delta V < 0$ and $V \to 0$ as $t \to \infty$, the network is asymptotically stable in the sense of Lyapunov. Finally, the decrease in V is independent of the initial placement of neurons in the DCS algorithm which implies that this stability is global.

The theorem verifies the fact that if we can construct a Lyapunov function as an error-minimizing term with initial boundary conditions ($V(0) = 0$), Lyapunov's stability theory can then guarantee the neural network's neighborhood structure, in terms of the best matching unit (bmu) to get closer to the training example in a globally asymptotically stable manner and thus preserving the features of the input manifold, which is the central goal of on-line learning neural network of the adaptive flight control system [24].

5 Online Stability Monitoring

It is of prime importance to understand if the neural network is convergent, meaning that trajectories converge to a stationary state even before we use them in real applications. Since we provided the required mathematical foundation to ensure the system to be stable, we now need to assure the robustness of system. In other words, if the online neural network encounters unusual data patterns that force the state of the system to deviate *away* from its current pattern, it always converges back to a stable equilibria within a *finite* amount of time. We may not always be able to assure robustness of the online network due to its implementation in an adaptive system, where the data patterns have no prior probability distribution. However, as the last resort of system assurance, we should at least be able to detect deviations of state that could lead to unstable behavior. This is is the objective of the Online Stability Monitor, shown in Figure 4.

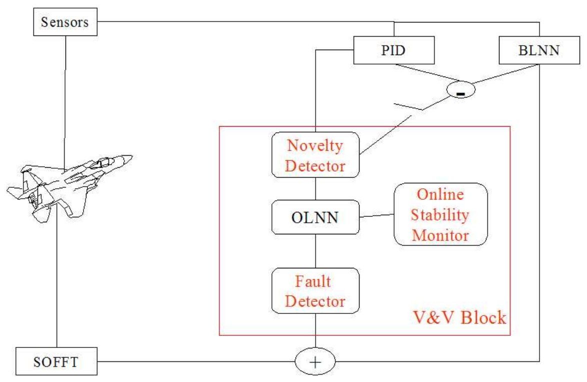

Fig. 4. V&V Mehtodology for OLNN

An interesting question guiding the design of online monitoring feature is: *How much an evolving state of the system needs to deviate in order for the stability monitor to label it as unstable?*

Since, the objective functions for online adaptive systems evolve over time it is hard to establish a complete system description a priori. Hence, we cannot precisely specify what an unstable state (for adaptive neural network) is. The hope is that the on-line monitor will be able to *detect* it when it sees it. An inadequate Lyapunov function may, of course, cause the excess of false-positive warnings, but this risk cannot be avoided. Online Stability Monitor complements analytical stability analysis by being being able to detect system states that deviate *away* from stable equilibria in real-time.

6 Case Study

In order to better understand the role of Lyapunov functions in self-stabilization of the given system a case study has been performed. The learning data for the DCS neural network was collected in an *F-15* flight simulator. For the sake of simplicity, the simulation data depicts nominal conditions of approximately 4 seconds of flight. During the simulation, the derivative outputs of the PID met the convergence criteria.

The plot in Figure 5 shows a portion of the DCS network learning behavior for the so called DCS_C_z subnetwork, one of the five DCS neural networks used in the real implementation of the IFCS. The input to the DCS_C_z network consisted of 186 data frames, each in a seven-dimensional space. Each input data frame consisted of 4 sensor readings and 3 stability and control derivative errors from PID and PNN.

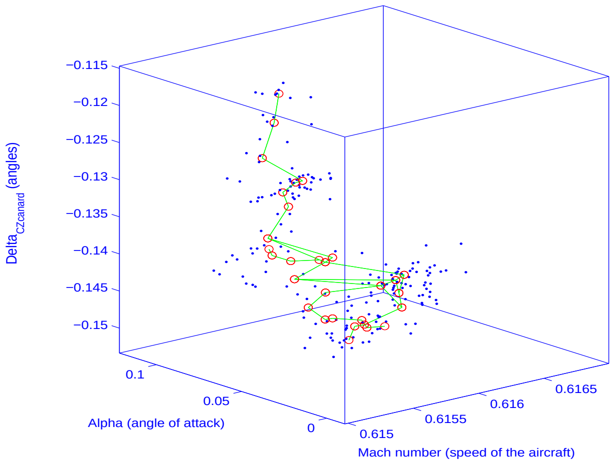

Fig. 5. Network Representation of Flight Control Variables

In this figure, two *independent* input variables (from sensors) to the $DCS - C_z$ network are plotted against a third *dependent* input variable (from PID-PNN) 'fed' into the $DCS - C_z$ network. These independent variables represent Mach numbers, the speed of the aircraft as compared to the local speed of sound, and alpha, aircraft's angle of attack. The dependent variable is one of the aircrafts stability and control derivative errors, Cz_{canard}. Figure 5 depicts the DCS network approximation of the data configuration plotted in a three dimensional subspace of the total seven dimensional data space.

Since adaptive systems are associated with uncertainty, degrees of freedom and high noise-level in real flight conditions, we may not always be able to check to see if each dimension of the input data is effectively abstracted and represented by the network. The data sets being modelled here represent very short data sequences for one out of five neural networks in the intelligent flight control system. The use of the constructed Lyapunov function (V), as shown in Figure 6, reduces the need for checking effective learning by each dimension as $V \in \mathbb{R}$. Rather than looking onto several dozen graphs, the adequacy (stability) of learning can be assessed from the analysis of a single graph, the graph representing

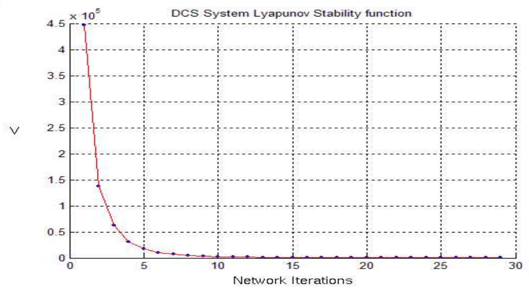

Fig. 6. Network's Lyapunov Function.

the Lyapunov function. Figure 6 depicts the convergence of V to a stable state within 10 epochs of neural network learning. Consequently, we can pinpoint the precise time at which the network reaches a stable state using the rate of decay of the Lyapunov function.

7 Summary and Discussion

In this paper, we discussed practical limitations to the idea of applying traditional self-stabilization approaches to adaptive systems. As an alternate approach we emphasize the role of a Lyapunov function in detecting unstable state deviations. A Lyapunov function for a DCS neural network has been constructed and used in a formal proof that established the conditions under which on-line learning for this type of network is self-stabilizing.

Further, we propose the idea of online stability monitoring for adaptive flight control systems. The goal of on-line monitoring is to provide a real-time safety warning methodology. A simulation study was conducted and it provided further evidence of self-stabilizing properties of the DCS neural network learning.

References

1. A. Arora, M. Demirbas and S. Kulkarni. *Graybox Stabilization.* International Conference on Dependable Systems and Networks (DSN'2001), Goteborg, Sweden, July 2001.
2. A. Arora. *Stabilization.* Encyclopedia of Distributed Computing, edited by Partha Dasgupta and Joseph E. Urban, Kluwer Academic Publishers, 2000.
3. Ingo Ahrns, Jorg Bruske, Gerald Sommer. *On-line Learning with Dynamic Cell Structures.* Proceedings of the International Conference on Artificial Neural Networks, Vol. 2, pp. 141-146, 1995.
4. William L. Brogan. *Modern Control Theory.* II Edition, Prentice-Hall Inc., 1985.
5. Jorg Bruske, and Gerald Sommer. *Dynamic Cell Structures*, NIPS, Vol. 7, Pages 497-504, 1995.
6. Jorg Bruske, Gerald Sommer. *Dynamic Cell Structure Learns Perfectly Topology Preserving Map.* Neural Computations, Vol. 7, No. 4, pp. 845-865, 1995.

7. Marc W. Mc. Conley, Brent D. Appleby, Munther A. Dahleh, Eric Feron. *Computational Complexity of Lyapunov Stability Analysis Problems for a Class of Nonlinear Systmes.* Society of industrial and applied mathematics journal of control and optimization, Vol. 36, No. 6, pp. 2176-2193, November 1998.
8. Edsger W. Dijkstra. *Self-stabilizing systems in spite of Distributed Control.* Communications of the Assocoation for Computing Machinery, 17(11), 643-644, 1974.
9. Bernard Friedland. *Advanced Control System.* Prentice-Hall Inc., 1996.
10. Bernd Fritzke. *A Growing Neural Gas Network Learns Topologies.* Advances in Neural Information Processing Systems, Vol. 7, pp. 625-632, MIT press, 1995.
11. Tom M. Heskes, Bert Kappen. *Online Learning Processes in Artificial Neural Networks.* Mathematical Foundations of Neural Networks, pp. 199-233, Amsterdam, 1993.
12. Charles C. Jorgensen. *Feedback Linearized Aircraft Control Using Dynamic Cell Structures.* World Automation Congress, ISSCI 050.1-050.6, Alaska, 1991.
13. J. L. W. Kessels. *An Exercise in Proving Self-satbilization with a variant function.* Information Processing Letters (IPL), Vol. 29, No.1, pp. 39-42, September 1998.
14. Teuvo Kohonen. *The Self-Organizing Map.* Proceedings of the IEEE, Vol. 78, No. 9, pp. 1464-1480, September 1990.
15. Thomas Martinetz, Klaus Schulten. *Topology Representing Networks.* Neural Networks, Vol. 7, No. 3, pp. 507-522, 1994.
16. Kumpati S. Narendra, Kannan Parthasarathy. *Identification and Control of Dynamic Systems Using Neural Networks.* IEEE Transactions on Neural Networks, Vol. 1, No. 1, pp. 4-27, March 1990.
17. Kumpati S. Narendra. *Intelligent Control.* American Control Conference, San Diego, CA, May 1990.
18. Jurgen Rahmel. *On The Role of Topology for Neural Network Interpretation*, Proc. of European Conference on Artificial Intelligence, 1996.
19. Orna Raz. *Validation of Online Artificial Neural Networks (ANNs)-An Informal Classification of Related Approaches.* Technical Report for NASA Ames Research Center, Moffet Field, CA, 2000.
20. N. Rouche, P. Habets, M. Laloy. *Stability Theory by Liapunov's Direct Method.* Springer-Verlag, New York Inc. publishers, 1997.
21. Marco Schneider. *Self-Stabilization.* Assocoation for Computing Machinery (ACM) sureveys, Vol. 25, No. 1, March 1993.
22. The Boeing Company. *Intelligent Flight Control: Advanced Concept Program.* Project report, 1999.
23. Oliver Theel. *An Exercise in Proving Self-Stabilization through Lyapunov Functions.* 21st International Conference on Distributed Computing Systems, ICDS'01, April 2001.
24. Sampath Yerramalla, Bojan Cukic, Edgar Fuller. *Lyapunov Stability Analysis of Quantization Error for DCS Neural Networks.* Accepted for publication at International Joint Conference on Neural Networks (IJCNN'03), Oregon, July 2003.
25. Wen Yu, Xiaoou Li. *Some Stability Properties of Dynamic Neural Networks.* IEEE Transactions on Circuits and Systems, Part-1, Vol. 48, No. 2, pp. 256-259, February 2001.
26. Xinghuo Yu, M. Onder Efe, Okyay Kaynak. *A Backpropagation Learning Framework for Feedforward Neural Networks*, IEEE Transactions on Neural Networks , No. 0-7803-6685-9/01, 2001.
27. V. I. Zubov. *Methods of A. M. Lyapunov and Their Applications.* U.S. Atomic Energy Commission, 1957.