Multistep-Ahead Neural-Network Predictors for Network Traffic Reduction in Distributed Interactive Applications

AARON McCOY, TOMAS WARD, and SEAMUS McLOONE
National University of Ireland Maynooth
and
DECLAN DELANEY
Eni

Predictive contract mechanisms such as dead reckoning are widely employed to support scalable remote entity modeling in distributed interactive applications (DIAs). By employing a form of controlled inconsistency, a reduction in network traffic is achieved. However, by relying on the distribution of instantaneous derivative information, dead reckoning trades remote extrapolation accuracy for low computational complexity and ease-of-implementation. In this article, we present a novel extension of dead reckoning, termed neuro-reckoning, that seeks to replace the use of instantaneous velocity information with predictive velocity information in order to improve the accuracy of entity position extrapolation at remote hosts. Under our proposed neuro-reckoning approach, each controlling host employs a bank of neural network predictors trained to estimate future changes in entity velocity up to and including some maximum prediction horizon. The effect of each estimated change in velocity on the current entity position is simulated to produce an estimate for the likely position of the entity over some short time-span. Upon detecting an error threshold violation, the controlling host transmits a predictive velocity vector that extrapolates through the estimated position, as opposed to transmitting the instantaneous velocity vector. Such an approach succeeds in reducing the spatial error associated with remote extrapolation of entity state. Consequently, a further reduction in network traffic can be achieved. Simulation results conducted using several human users in a highly interactive DIA indicate significant potential for improved scalability when compared to the use of IEEE DIS standard dead reckoning. Our proposed neuro-reckoning framework exhibits low computational resource overhead for real-time use and can be seamlessly integrated into many existing dead reckoning mechanisms.

This research is based upon works supported by Enterprise Ireland under Grant SC/2002/129/, and by the Science Foundation of Ireland and Enterprise Ireland under Grant IRCSET/ SC/04/CS0289.
Authors' addresses: A. McCoy, T. Ward, and S. McLoone, Department of Electronic Engineering, National University of Ireland Maynooth, Maynooth, Co. Kildare, Republic of Ireland; email: {amccoy, tomas.ward, seamus.mcloone}@eeng.nuim.ie; D. Delaney, Eni S.p.A., Milan, Italy; email: declan.delaney@eni.it.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; I.5.1 [**Pattern Recognition**]: Models—*Neural nets*; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Distributed*

General Terms: Algorithms, Performance, Theory

Additional Key Words and Phrases: Distributed interactive applications, distributed interactive simulation, networked virtual environments, collaborative virtual environments, networked multiplayer computer games, predictive contract mechanisms, dead reckoning, network bandwidth reduction, consistency, scalability, multistep-ahead prediction, neural networks, wargames

ACM Reference Format:

McCoy, A., Ward, T., McLoone, S., and Delaney, D. 2007. Multistep-ahead neural-network predictors for network traffic reduction in distributed interactive applications. ACM Trans. Model. Comput. Simul. 17, 4, Article 16 (September 2007), 30 pages. DOI = 10.1145/1276927.1276929 http://doi.acm.org/10.1145/1276927.1276929

1. INTRODUCTION

A distributed interactive application (DIA) is a networked virtual reality system through which geographically dispersed participants can share information via individual and collaborative interaction with each other and their environment [Churchill et al. 2001]. DIAs offer the realization of simulated virtual worlds that embody a modern extension of communication encompassing the concepts of shared time, shared space, and shared presence [Singhal and Zyda 1999]. From distributed military simulations and networked virtual environments to large-scale online multiplayer computer games, the widespread availability of low-cost, high-performance computing power and networking technology have enabled DIAs to become a reality for millions of people worldwide.

Conceptually, any real-time DIA can be assumed to consist of at least four components, namely, a simulated virtual environment, a set of virtual entities that interact within this environment, a physical collection of geographically dispersed host machines that run local, synchronized instances of the simulated virtual environment, and a physical computer network to facilitate communication between them [Macedonia and Zyda 1997]. During the application's execution, the responsibility for managing and updating the changing state of a virtual entity resides with a corresponding controlling (or source) host machine, who acts as a human-user interface to the content of the virtual world by providing autonomous control over the virtual entity to a single human-user (in such a scenario, the virtual entity is often referred to as the human-user's avatar). Remote (or receiver) host machines accept entity state updates (ESUs), transmitted over the network from the controlling host on behalf of the virtual entity to synchronize the display of that entity across each host machine, supplementing the pretence of a single, seamless virtual environment to the human-users [Roehle 1997]. The concept is illustrated graphically in Figure 1.

The fundamental goal for any real-time DIA is to provide for and maintain a synchronized (or consistent) view of the simulated virtual environment across all participating host machines that accurately reflects changes to that environment (and any virtual entities interacting within it) in real time, whilst also obscuring the distributed nature of the application from the perception of

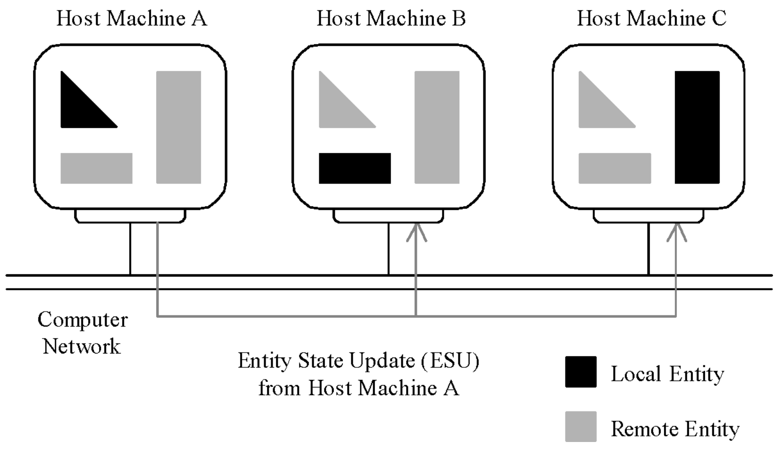


Fig. 1. Hosts must synchronize the display of local and remote entities by exchanging entity state updates.

the human-users [Sun et al. 1998]. The problems faced in achieving this goal stem from the inherent limitations associated with the transmission of data over a physical communication channel [Mathy et al. 1999]. In particular, the two primary factors inhibiting the large-scale deployment of a DIA are network latency and network bandwidth. Network latency refers to the delay in data transmission between two host machines. Network bandwidth refers to the rate at which data can be transmitted per unit time between two host machines, and is also referred to as channel capacity or throughput. The implications of variable network latency and limited network bandwidth are summarized within the consistency-throughput tradeoff, which states that one can have either a highly dynamic world or a consistent world, but not both [Singhal and Zyda 1999].

In an ideal world, one in which network latency is nonexistent and network bandwidth resources are infinite, host machines could transmit and receive an unlimited quantity of ESUs instantaneously, thereby guaranteeing consistency of the virtual environment. In reality, finite network (and computational) resources preclude such a scenario. Thus, throughout the literature, numerous techniques for reducing the perceivable effects of variable network latency, and for improving the efficient use of network bandwidth, have been proposed [Delaney et al. 2006a, 2006b]. Fundamentally, these techniques differ with respect to the approach they take towards maintaining consistency of the virtual environment [Fujimoto 2000; Greenberg and Marwood 1994]. Optimistic approaches seek to detect and subsequently resolve inconsistencies if and when they occur during the course of an application's execution. Conservative approaches, on the other hand, seek to take precautionary measures to prevent the occurrence of inconsistencies outright. For human-in-the-loop applications requiring real-time responsiveness, optimistic approaches are generally preferred [Bhola et al. 1998].

In this article, we focus on the use of one such optimistic approach, known as predictive contract mechanisms [Mellon and West 1995], for reducing the quantity of ESUs exchanged among host machines during the course of an application's execution. Under a predictive contract mechanism, controlling hosts are required to maintain two parallel models for each virtual entity under their control, namely, a high-fidelity model that represents the true entity state as computed in response to human-user input events, and a low-fidelity model that represents an approximation to the true entity state. The algorithm used to compute the approximation is known as the predictive contract, and is available at every remote host to extrapolate the time-dependent state of the virtual entity from the most recently received ESU in the absence of complete, up-to-date state information. By periodically comparing the deviation between the two models against a suitably defined error threshold, the controlling host can locally determine when to transmit ESUs to synchronize the presentation of the virtual entity. Hence, by employing a form of controlled inconsistency, predictive contract mechanisms sacrifice additional computational resources for reduced consumption of network resources in order to support scalability in real-time DIAs, in accordance with the information principle espoused by Singhal and Zyda [1999].

The most widely employed predictive contract mechanism in current use is known as dead reckoning. It was introduced during the development of SIMNET [Miller and Thorpe 1995] and subsequently formally defined within the *IEEE Standard for Distributed Interactive Simulation (DIS)* [IEEE 1995]. Dead reckoning makes use of instantaneous derivative information (such as velocity and acceleration) contained within the most recently received ESU(s) to extrapolate the time-dependent state (such as location) of a virtual entity using a polynomial predictor [Lin and Schab 1994a, 1994b]. At first glance, one might assume, then, that increasing the computational complexity of the polynomial predictor and adding higher-order derivative information (e.g., change in acceleration or jerk) would consequently improve the remote extrapolation accuracy, leading to an even further reduction in the consumption of network resources. In practice, however, increasing the computational complexity of the polynomial by adding higher-order derivative information increases the sensitivity of the predictor to rapidly changing entity state information, producing diminishing returns and increasing the likelihood of inaccurate extrapolation over time [Singhal and Zyda 1999]. Hence, in order to improve the extrapolation procedure, one can either replace the polynomial predictor with an alternative solution, or improve the accuracy of the derivative state information used by them.

The work presented in this article takes the latter approach. In particular, we propose a novel extension of dead reckoning, termed neuro-reckoning, that seeks to replace the use of instantaneous derivative information with predictive derivative information in order to improve the accuracy of entity state extrapolation at remote hosts. The concept is illustrated graphically in Figure 2. When the deviation between the high-fidelity model (representing the true entity state) and low-fidelity model (representing the approximated entity state) exceeds the error threshold, the controlling host must transmit an ESU for that entity over the network in order to synchronize the presentation of the entity at each remote host. Under dead reckoning, the controlling host would transmit the instantaneous velocity vector. Under neuro-reckoning, the controlling host

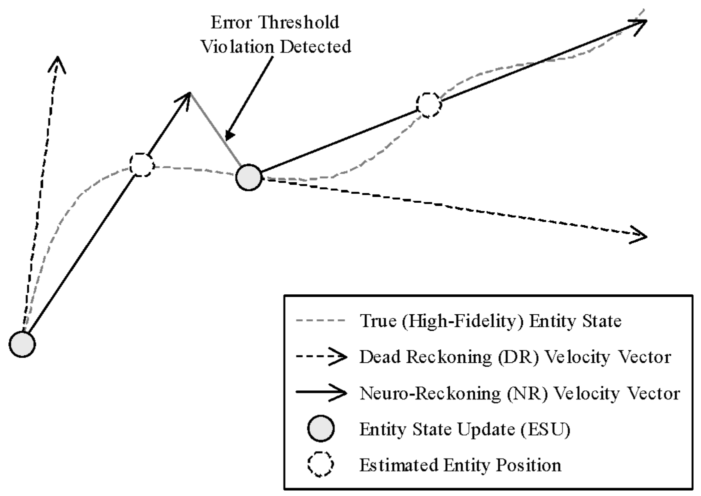

Error Threshold Violation Detected
True (High-Fidelity) Entity State
Dead Reckoning (DR) Velocity Vector
Neuro-Reckoning (NR) Velocity Vector
Entity State Update (ESU)
Estimated Entity Position

Fig. 2. An illustration of the fundamental concept underlying our proposed neuro-reckoning approach.

employs a bank of neural network predictors to estimate the likely position of the entity over some short time-span, given its current motion, and transmits a velocity vector that extrapolates through that point instead. By distributing predictive derivative information that implicitly encodes expected entity motion, neuro-reckoning aims to reduce the spatial error associated with remote extrapolation of entity state. Consequently, the ultimate goal of our approach is to further reduce the number of ESUs exchanged among host machines due to error threshold violation when compared with dead reckoning. Although the primary focus of our work is directed towards real-time, human-in-the-loop DIAs such as distributed military simulations and networked multiplayer computer games, the concept can be applied to any domain in which predictive contract mechanisms are used for network traffic reduction, including multiuser collaborative virtual environments, 3D cyberspaces and networked virtual communities, distributed visualization and interactive walkthrough applications, and 3D chatting systems.

The remainder of this article is organized as follows. In Section 2, we present a brief overview of the standard dead reckoning procedure. In Section 3, we present an overview of our proposed neuro-reckoning framework that includes a detailed description of the procedure used to generate predictive derivative information. In Section 4, we present simulation results comparing our approach with the use of IEEE DIS standard dead reckoning. Finally, in Section 5, we offer conclusions and directions for future research.

2. DEAD RECKONING

The polynomial predictors used in dead reckoning can be derived using the Taylor series expansion for some scalar, real-valued function $f(t) = x$ (where x represents some time-dependent entity state variable) expanded about the point t_0 in order to extrapolate the value of x_i at time $t_i = t_0 + ih$. This is as follows [Lin and Schab 1994a, 1994b].

$$x_i = \sum_{n=0}^{\infty} \frac{f^{(n)}(t_0)}{n!}(ih)^n = x_0 + \dot{x}_0 ih + \frac{\ddot{x}_0}{2}(ih)^2 + \frac{\dddot{x}_0}{6}(ih)^3 + \ldots, \tag{1}$$

where h is the simulation (dead reckoning) step-time, $n!$ denotes the factorial of n, and $f^{(n)}(t_0)$ denotes the nth derivative of $f(t)$ at the point t_0. If unavailable, the value of $f^{(n)}(t_0)$ can be approximated by numerical differentiation. The order of each dead reckoning equation is defined as the order of its truncation error (i.e., the cut-off point for derivatives in the Taylor series expansion) minus one [Lin and Schab 1994a, 1994b]. In general, the inclusion of higher-order terms produces diminishing results from a dead reckoning perspective, and for this reason first, and second-order equations are almost always preferred [Singhal and Zyda 1999]. The neuro-reckoning approach presented in this article extrapolates entity state using a first-order equation. In comparing the performance of our approach to that of standard dead reckoning, however, we consider both first- and second-order equations for the latter [Lee et al. 2000].

$$\text{first-order:} \qquad x_i = x_0 + \dot{x}_0 ih \tag{2}$$

$$\text{second-order:} \qquad x_i = x_0 + \dot{x}_0 ih + \frac{\ddot{x}_0}{2}(ih)^2 \tag{3}$$

Dead reckoning mechanisms rely on the use of an error threshold and a timeout to regulate the transmission of entity state updates (ESUs) [Ryan and Oliver 2006]. The use of an error threshold is designed to bound the remote extrapolation error of the polynomial predictor by imposing an upper bound on the tolerable deviation between the high-fidelity and low-fidelity models needed to trigger an update. The use of a timeout is designed to guarantee a minimum frequency of entity state update (ESU) transmission at the cost of potentially redundant network bandwidth usage by imposing an upper bound on the elapsed time between consecutive updates. The selection of an appropriate error threshold and an appropriate timeout defines the tradeoff between resource usage and consistency in a dead reckoning mechanism. Consequently, a number of adaptive error threshold selection schemes have been proposed throughout the literature [Cai et al. 1999; Chen and Chen 2005; Shim and Kim 2001; Yu and Choy 2001; Zhang and Georganas 2004], alongside several techniques for reducing the reliance of a dead reckoning mechanism on the use of a timeout [Srinivasan 1996; Van Hook et al. 1995]. In comparing the performance of our proposed neuro-reckoning approach to that of standard dead reckoning, however, we consider only fixed error thresholds and fixed timeouts.

Finally, it should be noted that dead reckoning mechanisms rely on the use of convergence algorithms to smoothly correct the display of inconsistencies arising due to inaccurate extrapolation [Lin et al. 1995]. The application of a convergence algorithm is usually left to the discretion of each remote host, however, and thus the effect of applying the convergence is not considered by the controlling host in determining when to transmit those entity state updates (ESUs) due to error threshold violation. Hence, in comparing the performance of our proposed neuro-reckoning approach to that of standard dead reckoning, we do not consider the direct application of convergence algorithms.

3. NEURO-RECKONING FRAMEWORK

3.1 Overview

As outlined in the Introduction, our proposed neuro-reckoning approach seeks to replace the use of instantaneous derivative information with predictive derivative information in order to improve the accuracy of entity state extrapolation at remote hosts. It does this by transmitting predictive velocity vectors in place of instantaneous velocity vectors, as shown in Figure 2. The procedure used to generate predictive velocity vectors shall be described in detail in the next section. For now, we abstract the concept of our neuro-reckoning approach into a self-contained component module that can be interfaced into the standard dead reckoning procedure with relative ease. The advantage of such an abstraction is twofold. Firstly, it allows us to graphically illustrate how our approach modifies the standard dead reckoning procedure in an intuitive manner. Secondly, it allows us to analyze the additional resource requirements that are introduced by our approach in isolation from those introduced by standard dead reckoning (see Section 3.4).

As an extension of dead reckoning, our proposed neuro-reckoning approach modifies the procedure for transmitting ESUs from the perspective of the controlling host only (i.e., the sender-side of the network connection, as opposed to the receiver-side). As far as remote hosts are concerned, the dead reckoning procedure remains unchanged; they receive ESUs containing position and velocity information (along with any other state variables that require replication) and extrapolate using a first-order polynomial predictor as described in Eq. (2). Figure 3 presents a flow-chart describing the high-level operation of our proposed neuro-reckoning approach from the perspective of the controlling host. The neuro-reckoning module interfaces directly to the high-fidelity model that represents the true entity state, as computed in response to human-user input events, and also to the network module responsible for transmitting ESUs once an error threshold violation or timeout is detected. The format of the data sent across the network remains unchanged from that sent by the standard dead reckoning procedure. In this respect, our proposed neuro-reckoning framework can be implemented without altering the underlying network software architecture of an application's existing dead reckoning mechanism(s).

3.2 Neuro-Reckoning Module

Figure 4 presents a block diagram describing the inner workings of the neuro-reckoning module introduced in the previous section. The notation used in the description (as well as in the rest of the article) is defined in Table I. The module interfaces to the high-fidelity model through the use of a tapped delay-line that accepts the current entity state vector at each simulation step. Once the module is executed, it outputs a predictive velocity vector for subsequent transmission to remote hosts, in the form of a standard ESU.

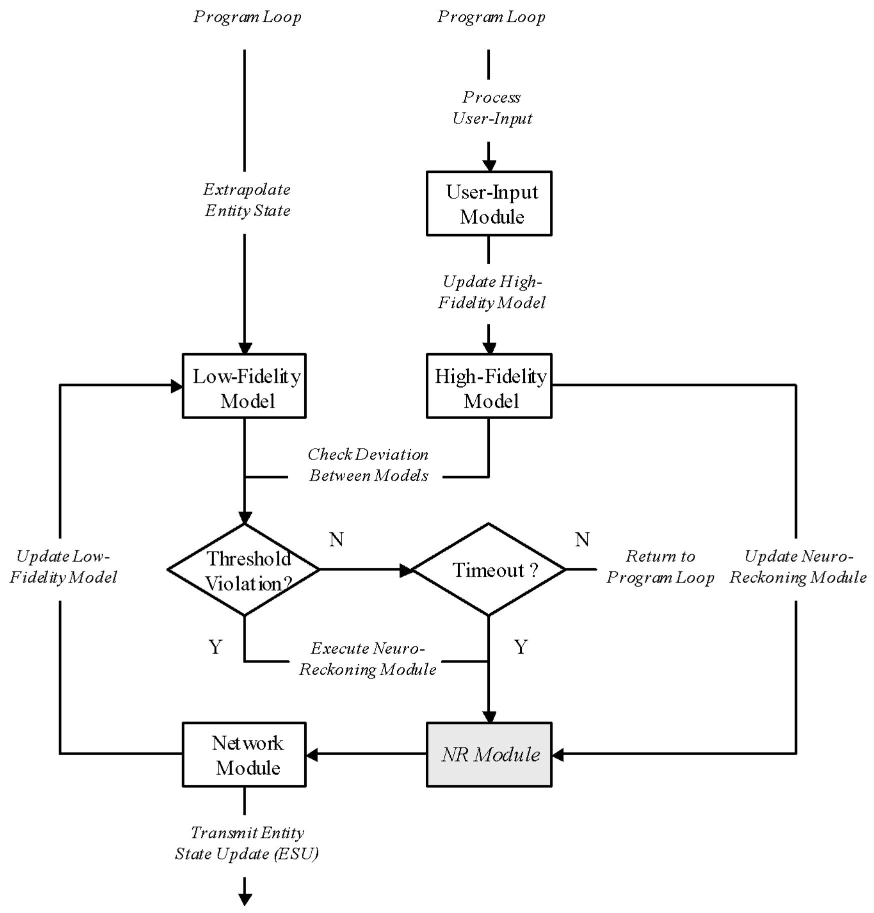


Fig. 3. Flow-chart describing the high-level operation of our proposed neuro-reckoning approach.

At its core, the module employs a bank of neural network predictors trained to estimate future changes (or difference vectors) in entity velocity (i.e., acceleration) up to and including some maximum prediction horizon q. The module maintains a sliding window that stores the $k+1$ most recent entity state vectors from the tapped delay-line up to and including the current time t. A process of feature extraction is performed over the window of entity state vectors to construct a suitable input to the bank of neural network predictors for the current time-step. Once each neural network predictor has been simulated and a corresponding estimate for the future change in entity velocity for that prediction horizon generated, the module simulates the effect of each velocity estimate on the entity position vector in a cumulative manner using a series of first-order polynomial predictors. This culminates in a final estimate for the likely position of the entity over a future time-span of $(q + 1)h$ seconds. This procedure can be described as a process of forward dead reckoning simulation through time. By normalizing the difference vector between the current and estimated entity position (thereby generating a unit direction vector) and scaling the result

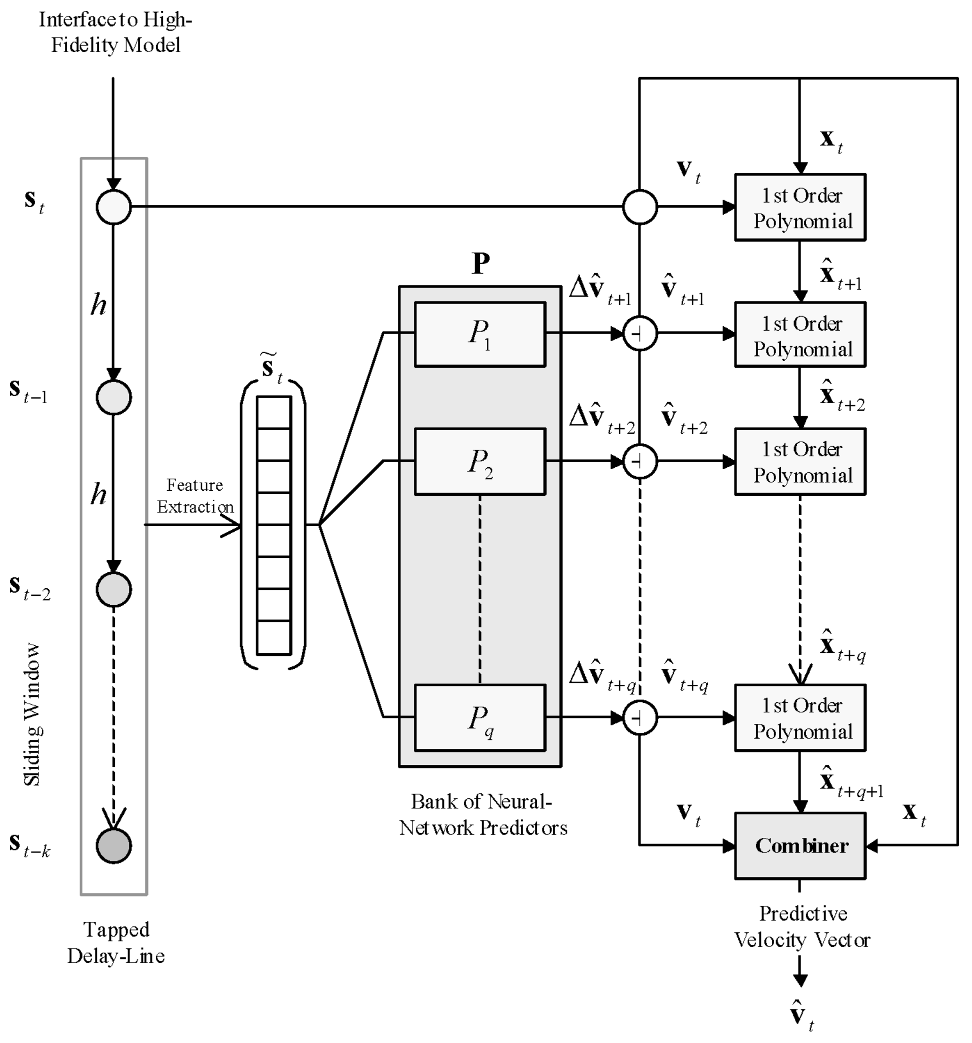


Fig. 4. Block diagram describing the inner workings of the neuro-reckoning module introduced previously.

by the current entity speed (i.e., magnitude of the current entity velocity vector), the neuro-reckoning module can construct a predictive velocity vector that extrapolates through the estimated position. The predictive velocity vector is then transmitted over the network as a standard ESU to remote hosts, instead of the current instantaneous velocity vector.

Eqs. (4)–(8) define the relationships between the variables listed in Table I, and therefore the system dynamics implemented by the block diagram given in Figure 4. The implementation of neural network predictors is discussed in the following section, so for now we simply denote the feature extraction process as $f(.)$ and the input-output functional mapping implemented by the ith neural

Table I. Summary of Notation

Symbol	State-Space	Description
h	$\Re^+$	simulation (dead reckoning) time-step
k	$\Im^+$	maximum time-delay
q	$\Im^+$	maximum prediction horizon
D	$\Im^+$	dimension of complete entity state vector
d	$\Im^+$	dimension of entity attribute vectors (e.g., position, velocity, etc.)
$\mathbf{s}_t$	$\Re^D$	complete entity state vector at time t where $\mathbf{s}_t = (\mathbf{x}_t, \mathbf{v}_t, \theta_t)$
$\mathbf{x}_t \subseteq \mathbf{s}_t$	$\Re^d$	entity position vector at time t
$\mathbf{v}_t \subseteq \mathbf{s}_t$	$\Re^d$	entity velocity vector at time t
$\theta_t \subseteq \mathbf{s}_t$	$\Re^d$	entity orientation vector at time t
$\hat{\mathbf{x}}_{t+i}$	$\Re^d$	estimated entity position vector at time $t+i$
$\hat{\mathbf{v}}_{t+i}$	$\Re^d$	estimated entity velocity vector at time $t+i$
$\Delta\hat{\mathbf{v}}_{t+i}$	$\Re^d$	estimated change in entity velocity vector from time t to time $t+i$
n	$\Im^+$	no. of input neurons (input-layer dimension) per neural network
m	$\Im^+$	no. of hidden neurons (hidden-layer dimension) per neural network
P_i	n/a	ith neural-network predictor
$\mathbf{P}$	n/a	set of all neural-network predictors $\mathbf{P} = \{P_i\}, \forall i \in \{1, \ldots, q\}$
$\tilde{\mathbf{s}}_t$	$\Re^{n \times 1}$	input column vector to the set of neural-network predictors at time t
$\mathbf{W}_i$	$\Re^{m \times n}$	input/hidden-layer interconnection weight matrix of ith neural-net
$\mathbf{U}_i$	$\Re^{d \times m}$	hidden/output-layer interconnection weight matrix of ith neural-net
$\mathbf{b}_i$	$\Re^{m \times 1}$	hidden-layer bias column vector of ith neural-network predictor
$\mathbf{c}_i$	$\Re^{d \times 1}$	output-layer bias column vector of ith neural-network predictor

Symbols defined with a ^ denote an estimated or predicted quantity, variable, or matrix. $\Re^+ = [0, \infty)$ denotes the set of nonnegative real numbers. $\Im^+ = \{0, 1, 2, \ldots\}$ denotes the set of nonnegative integers.

network predictor as $g_i(.)$. The operator $|.|$ denotes the magnitude or length of a vector.

$$\text{feature extraction:} \quad \tilde{\mathbf{s}}_t = f(\mathbf{s}_t, \ldots, \mathbf{s}_{t-k}) \tag{4}$$

$$\text{neural-network simulation:} \quad \Delta\hat{\mathbf{v}}_{t+i} = g_i(\tilde{\mathbf{s}}_t) \quad \forall\, i \in \{1, \ldots, q\} \tag{5}$$

$$\text{entity velocity vector estimation:} \quad \hat{\mathbf{v}}_{t+i} = \mathbf{v}_t + \Delta\hat{\mathbf{v}}_{t+i} \quad \forall\, i \in \{1, \ldots, q\} \tag{6}$$

$$\text{entity position vector extrapolation:} \quad \hat{\mathbf{x}}_{t+1} = \mathbf{x}_t + \mathbf{v}_t h$$
$$\hat{\mathbf{x}}_{t+i+1} = \hat{\mathbf{x}}_{t+i} + \hat{\mathbf{v}}_{t+i} h \quad \forall\, i \in \{1, \ldots, q\} \tag{7}$$

$$\text{predictive velocity vector computation:} \quad \hat{\mathbf{v}}_t = \frac{\hat{\mathbf{x}}_{t+q+1} - \mathbf{x}_t}{\left|\hat{\mathbf{x}}_{t+q+1} - \mathbf{x}_t\right|}\,|\mathbf{v}_t| \tag{8}$$

3.3 Neural-Network Predictors

The role of the feature extraction process is to construct a suitable column vector for use as input to the bank of neural network predictors during the current simulation step that is based on some combination of the entity state vectors currently stored in the sliding window. The approach adopted in this article is to extract both the d-dimensional entity velocity vector $\mathbf{v}_t$ and the d-dimensional entity orientation vector θ_t from each entity state vector $\{\mathbf{s}_t, \ldots, \mathbf{s}_{t-k}\}$ stored in the sliding window and to combine them into a single column vector for use as input to the bank of neural network predictors. Such an approach removes

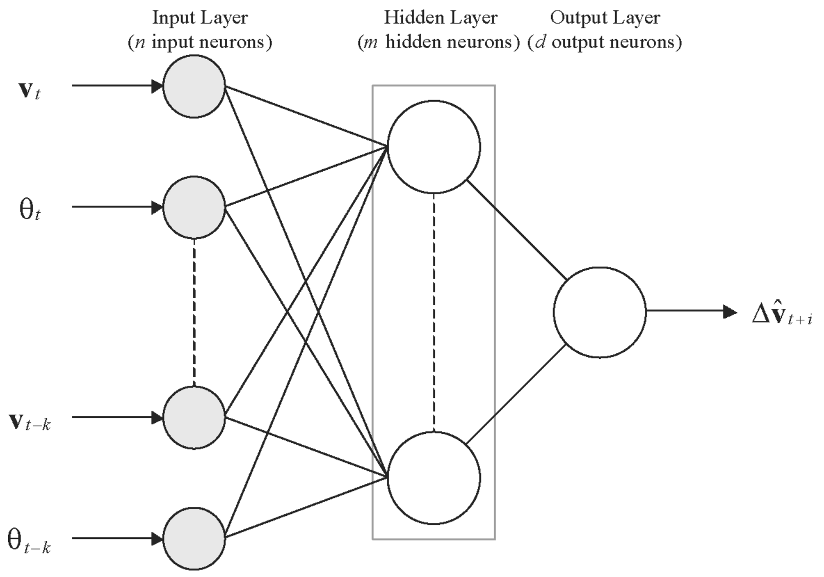


Fig. 5. Topology of the neural-network predictors that form the core of the neuro-reckoning module.

the dependence of neural network predictors on the (potentially large) spatial state-space of the virtual environment (i.e., the d-dimensional entity position vector $\mathbf{x}_t$). In particular, we wish to avoid localizing the neural networks to regions of the spatial state-space that would complicate training, and instead we rely purely on the characteristics of the entity motion itself. Hence, our neural network predictors can be used across multiple virtual environments, irrespective of different spatial layouts (e.g., different levels or new maps in a networked multiplayer computer game).

$$\tilde{\mathbf{s}}_t = f(\mathbf{s}_t, \ldots, \mathbf{s}_{t-k}) = (\mathbf{v}_t, \boldsymbol{\theta}_t, \ldots, \mathbf{v}_{t-k}, \boldsymbol{\theta}_{t-k})^{\mathrm{T}} \tag{9}$$

In order to implement the bank of neural network predictors that forms the core of the neuro-reckoning module described in the previous section, we employ a collection of static multilayer perceptrons (MLPs) [Principe et al. 2000]. MLPs have been shown to have good mapping capabilities for modeling both real and virtual entities in a number of application domains, including distributed military simulations [Henninger et al. 2001], networked virtual environments [Sas et al. 2003], networked multiplayer computer games [Thurau et al. 2003], and autonomous robot soccer [Behnke et al. 2003]. In our approach, each successive MLP accepts the same input column vector as its predecessors, but is trained to predict over a longer prediction horizon independently of the outputs from other networks. Such a scheme can best be described as a form of direct multistep-ahead prediction and is used primarily in contrast to the choice of a recursive single-step-ahead prediction scheme that may exhibit unstable behavior [Boné and Crucianu 2002].

The topology of the neural network predictors is shown in Figure 5. Each MLP contains a single input-layer, a single hidden-layer, and a single output-layer arranged in a (fully connected) feed-forward architecture. The number of input neurons is given by $n = 2d(k + 1)$, the number of hidden neurons denoted by m, and the number of output neurons given by d (see Table I). Such a feed-forward architecture is denoted as (n-m-d). Each MLP is batch-trained using the standard Levenberg-Marquardt backpropagation (LM-BP) algorithm for updating the network parameters based on the mean squared error (MSE) over some training dataset [Hagan and Menhaj 1994]. Nonlinear sigmoidal "hyperbolic tangent (tanh)" transfer functions are used by neurons contained within the hidden layer, while linear transfer functions are used by neurons contained within the output layer. Such an architecture is known to have universal mapping capabilities under certain conditions [Principe et al. 2000]. The topology shown in Figure 5 is represented in vector space. As per Table I, entity attribute vectors (e.g., velocity and orientation) are each of dimension d. Consequently, neurons pictured as residing in either the input- or output-layer each represent d components of entity state, and hence must be physically implemented as d separate neurons (for practical purposes).

Based on the aforementioned MLP architecture, the input-output functional mapping implemented by the ith neural-network predictor is given by the equation

$$\Delta\hat{\mathbf{v}}_{t+i} = g_i(\tilde{\mathbf{s}}_t) = \left(\underbrace{\mathbf{U}_i \left(\underbrace{\sigma(\mathbf{W}_i \tilde{\mathbf{s}}_t + \mathbf{b}_i)}_{\text{hidden layer}} \right) + \mathbf{c}_i}_{\text{output layer}} \right)^{\mathrm{T}} \quad \forall i \in \{1, \ldots, q\}, \tag{10}$$

where $\mathbf{W}_i$, $\mathbf{U}_i$, $\mathbf{b}_i$, and $\mathbf{c}_i$ are the parameters for the ith neural network as defined in Table I, and $\sigma(.)$ is the nonlinear sigmoidal "hyperbolic tangent (tanh)" transfer function used by neurons contained within the hidden layer of the network. Assuming an arbitrary, real-valued scalar variable x, then $\sigma(x)$ is defined by the following equation [Harrington 1993].

$$\sigma(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1, \tag{11}$$

where e^x is the transcendental exponential function [Mizutani and Dreyfus 2001]. As can be seen from examination of Eq. (10), $\sigma(.)$ is computed on an element-by-element basis over the net output vector from the m hidden neurons for the ith neural network that is given by the expression

$$\mathbf{W}_i \tilde{\mathbf{s}}_t + \mathbf{b}_i \tag{12}$$

and produces an $\Re^{m \times 1}$ column vector.

Both the inputs to and outputs from the neural network predictors can be standardized and destandardized, respectively, using Eq. (13), shown next that scales the value of any arbitrary real-valued, scalar variable x from the range $[x_{\min}, x_{\max}]$ to the range $[y_{\min}, y_{\max}]$ (values of ranges are purely application dependent)

$$y = (x - x_{\min}) \frac{(y_{\max} - y_{\min})}{(x_{\max} - x_{\min})} + y_{\min} \tag{13}$$

Table II. Theoretical Computational Analysis

Neuro-Reckoning (NR) Algorithm	(Approximate) Flops
Process 1: Input column vector standardization	$12d(k+1)$
Process 2: Neural-network simulation	$mq(4+6d+4dk+T_e)$
Process 3: Output vector destandardization	$6qd$
Process 4: Entity velocity vector estimation	qd
Process 5: Entity position vector extrapolation	$2d(q+1)$
Process 6: Predictive velocity vector computation	$6d+2T_s-1$

Process 1 and Process 3 are not shown in Figure 4, but are implicitly assumed to be required. Flop counts and execution times for our particular implementation are reported in Section 4.4.

Standardizing the inputs and outputs in this manner was observed to improve the training performance of neural network predictors [Shanker et al. 1996].

3.4 Theoretical Computational Analysis

Our proposed neuro-reckoning approach sacrifices additional computational resources for a further reduction in the consumption of network resources over that provided for by the standard dead reckoning procedure. Hence, we now perform a theoretical computational analysis of the (approximate) number of additional floating-point operations (or flops) required by our approach. In the following analysis, we assume that the differences in execution time (or cost) of performing arithmetic flops (i.e., addition, subtraction, multiplication, and division) are negligible. Furthermore, for some arbitrary real-valued, scalar variable x, we denote the number of flops required to execute the transcendental exponential function e^x by T_e and that required to execute the square root function $x^{1/2}$ by T_s. Evaluating a transcendental exponential function e^x can take considerably more processing cycles than a standard arithmetic operation and can account for a significantly large proportion of the total estimated execution time [Mizutani and Dreyfus 2001]. Fast approximations to the exponential function, such as that proposed by Cawley [2000], can thus be employed to reduce the execution time, if required.

The algorithmic operation of the neuro-reckoning (NR) procedure can be subdivided into 6 processes. As such, we perform a theoretical computational analysis for each process in turn with respect to the various parameters that define the process and the number of approximate flops needed to execute it. The results are summarized in Table II. For brevity, we have omitted the actual derivation of the number of flops per process. The interested reader can trivially verify the results by direct application of Eqs. (4)–(13) and by substitution of the equality $n = 2d(k+1)$ as described in Section 3.3. As can be seen, lowering any of the 6 parameters (k, q, d, m, T_e, or T_s) reduces the additional computational resources that are required by our proposed neuro-reckoning approach in return for a probable decrease in accuracy of the resulting predictive velocity vectors transmitted to and used by remote hosts for extrapolating entity state. Quantifying the exact analytical tradeoff between computational and network resources is impossible, as one can never be truly certain of how accurate an extrapolation will be (in general) when attempting to predict human-user behavior in advance. The number of approximate flops shown in Table II serves as a general estimate for the architecturally independent cost of implementing our proposed approach. Once an estimate for the execution time of a single flop on a given architecture is known, the cost of our algorithm can be estimated a priori.

4. RESULTS AND ANALYSIS

4.1 Overview

To demonstrate and validate our proposed neuro-reckoning approach within an application domain exhibiting a realistic scope, experiments are conducted under the guise of a simple first-person shooter (FPS)-style game scenario. This strategy was developed using the commercially available torque game engine (TGE) [Lloyd 2004]. The rules that govern the game are fairly typical of the type of "deathmatch" scenarios used within online FPS games, and yield data that can be considered a fair representation for the type of data that one would expect to observe in commercial networked multiplayer computer games and other similar real-time applications [McCoy et al. 2004]:

(1) Experiments consist of an open environment containing a collection of spawn points that designate randomly assigned starting locations for each entity.
(2) At the beginning of the game, entities are placed randomly into the game world at one of these spawn points.
(3) Entities possess projectile-based weaponry, along with a limitless supply of ammunition that is used for disabling an opponent in order to score a "hit-point".
(4) Projectiles are subject to simulated gravitational forces when in motion and inflict an amount of damage inversely proportional to their blast radius upon impact.
(5) Entities possess a health meter that quantifies the amount of damage needed to disable them.
(6) Disabled entities are randomly respawned at one of the spawn points following a time-period of no greater than 2 seconds.
(7) The goal is to be the first player to reach a prespecified "hit-point" score-limit, at which point the experiment may be repeated or considered completed.

Figure 6 shows a screenshot of the game in action, along with a graphical representation of the state-space in which the entities interact with one another with respect to implementing our proposed neuro-reckoning approach. Although the game world is modeled in true arbitrary 3-dimensional space, the vertical component of entity velocity is negligible and has little impact on the resulting motion trajectories generated by the interacting entities. Thus, for the purposes of training the set of neural network predictors, entity velocity and orientation are modeled as 2-dimensional state vectors (i.e., $d = 2$). As

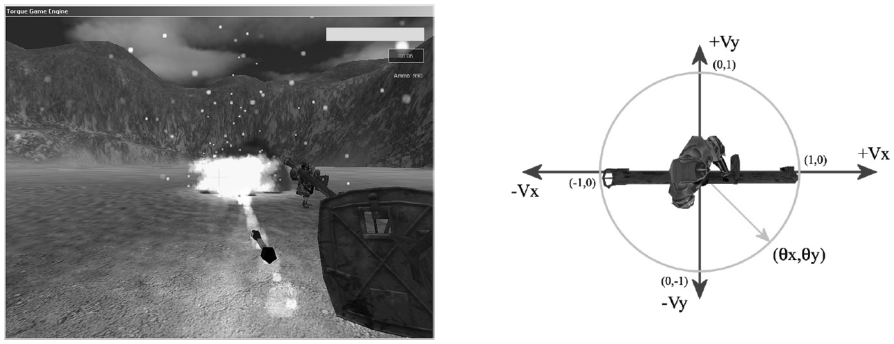


Fig. 6. Left: the game in action; right: the state-space for training the set of neural-network predictors.

noted by Behnke et al. [2003], we encode the entity orientation vector as a point (i.e., a sine and cosine coordinate pair) on a unit circle centered at the entity's current position to avoid the training complications associated with the discontinuity between π and -π when encoding a single angle. All measurements are defined in terms of torque world units (TWUs).

$$\text{velocity:} \quad \mathbf{v}_t = (v_x, v_y) \quad \text{where} \quad v_x, v_y \in [\,-15\,,\,+15\,] \tag{14}$$

$$\text{orientation:} \quad \boldsymbol{\theta}_t = (\theta_x, \theta_y) \quad \text{where} \quad \theta_x, \theta_y \in [\,-1\,,\,+1\,] \tag{15}$$

4.2 Neural-Network Implementation

Our implementation consisted of 4 human-users (each possessing a varying degree of expertise with respect to networked multiplayer computer games: users 1 and 4 being the most experienced, user 3 the least) that were required to individually compete in a series of 12 consecutive experiments (each consisting of a prespecified "hit-point" score limit of 10). Specifically, they competed against a scripted computer-controlled opponent (bot) operating under the influence of a dynamic shortest-path behavioral model [McCoy et al. 2005]. Data is collected at a rate of 20 samples per second (i.e., a constant simulation time-step of h = 50ms). Of the 12 experimental datasets collected for each user, the first 2 were discarded due to the probable appearance of transient behavior related to the initial learning strategies adopted by each human-user, leaving a total of 10 datasets per human-user. Of these 10 datasets, the final 2 were kept isolated from the data pertaining to training the neural networks for the purposes of providing an unbiased means of performance evaluation. The remaining 8 datasets were then combined, at which point a series of exemplars were extracted, and subsequently divided randomly into suitable training (70%), validation (15%), and testing (15%) sets. The exemplars (consisting of the inputs and corresponding target values required for supervised learning) were standardized to the interval [−1, 1]. Based on early evaluations performed using various network topologies, the maximum time-delay k was set at 3 (longer time-delays were seen to produce negligible improvement in training performance relative to the additional network complexity), the maximum prediction horizon q was set at 10, and the number of hidden neurons per network m was set at 8. Hence, the

Table III. Coefficient of Determination (r^2) Over the Training Set for Each User

Prediction Horizon	User 1 (7680) Δv_x	User 1 (7680) Δv_y	User 2 (8320) Δv_x	User 2 (8320) Δv_y	User 3 (7475) Δv_x	User 3 (7475) Δv_y	User 4 (8704) Δv_x	User 4 (8704) Δv_y
1	0.45	0.47	0.52	0.52	0.54	0.58	0.53	0.53
2	0.46	0.43	0.47	0.44	0.59	0.61	0.50	0.47
3	0.34	0.33	0.37	0.33	0.55	0.58	0.32	0.32
4	0.28	0.27	0.41	0.39	0.50	0.53	0.31	0.30
5	0.24	0.22	0.32	0.29	0.45	0.51	0.26	0.27
6	0.22	0.23	0.32	0.27	0.42	0.45	0.25	0.26
7	0.21	0.22	0.30	0.25	0.44	0.48	0.23	0.23
8	0.23	0.21	0.31	0.28	0.39	0.44	0.22	0.21
9	0.24	0.21	0.30	0.28	0.40	0.46	0.22	0.21
10	0.23	0.19	0.29	0.28	0.39	0.41	0.23	0.23

architecture of each neural network can be denoted as (16-8-2) (see Section 3.3). A separate set of neural network predictors was trained for each user.

4.3 Neural-Network Training Results

Each neural network predictor is trained to predict changes in entity velocity over a specific prediction horizon, where training continues until the validation set determines the "early-stopping" point for maximum generalization [Principe et al. 2000] (or, alternatively, until the training period reaches 100 epochs). The performance criterion used in the training procedure is the mean squared error (MSE) between predicted and actual changes in entity velocity over some target dataset. Unfortunately, the numerical value of the MSE depends upon the magnitude of the target data samples, limiting its effectiveness in comparing the performance of predictors trained over different prediction horizons [Battaglia 1996]. Hence, in order to compare the performance of each neural network predictor over the increasing prediction horizon in a meaningful fashion, we quantify the accuracy of each predictor using the coefficient of determination. The coefficient of determination measures the proportion (or percentage, if multiplied by 100) of the variance (or fluctuation) of one variable that is predictable by knowledge of another—in other words, the ratio of explained variation to total variation—and is defined as follows [Rodgers and Nicewander 1988].

$$r^2 = 1 - \frac{\sum_{i=1}^{n} (x_i - \hat{x}_i)^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2}, \quad 0 \le r^2 \le 1, \tag{16}$$

where: x_i is the ith target data sample;
$\hat{x}_i$ is the ith predicted data sample;
$\bar{x}$ is the mean (or average) of the set of target data samples; and
n is the total number of target (and of predicted) data samples.

Tables III to V present the coefficient of determination (r^2) as computed over the training, validation, and test sets, respectively, for each user. The

Table IV. Coefficient of Determination (r^2) over the Validation Set for Each User

Prediction Horizon	User 1 (1656)		User 2 (1761)		User 3 (1623)		User 4 (1904)	
	Δv_x	Δv_y	Δv_x	Δv_y	Δv_x	Δv_y	Δv_x	Δv_y
1	0.42	0.41	0.56	0.53	0.52	0.56	0.50	0.52
2	0.40	0.34	0.50	0.46	0.56	0.60	0.46	0.46
3	0.30	0.25	0.38	0.37	0.54	0.56	0.30	0.33
4	0.24	0.19	0.40	0.41	0.47	0.50	0.28	0.30
5	0.21	0.16	0.32	0.31	0.45	0.51	0.24	0.27
6	0.19	0.17	0.33	0.27	0.41	0.44	0.21	0.25
7	0.19	0.14	0.31	0.26	0.39	0.47	0.21	0.21
8	0.22	0.18	0.32	0.26	0.38	0.44	0.20	0.21
9	0.22	0.16	0.28	0.27	0.39	0.42	0.18	0.21
10	0.23	0.15	0.27	0.26	0.36	0.40	0.22	0.21

Table V. Coefficient of Determination (r^2) Over the Test Set for Each User

Prediction Horizon	User 1 (1656)		User 2 (1761)		User 3 (1623)		User 4 (1904)	
	Δv_x	Δv_y	Δv_x	Δv_y	Δv_x	Δv_y	Δv_x	Δv_y
1	0.46	0.44	0.52	0.50	0.50	0.60	0.52	0.54
2	0.47	0.40	0.45	0.40	0.55	0.59	0.45	0.49
3	0.34	0.31	0.34	0.32	0.47	0.53	0.28	0.36
4	0.27	0.26	0.36	0.34	0.43	0.50	0.25	0.34
5	0.24	0.22	0.27	0.29	0.38	0.48	0.20	0.30
6	0.22	0.21	0.27	0.27	0.35	0.42	0.18	0.26
7	0.22	0.20	0.25	0.24	0.36	0.41	0.17	0.23
8	0.23	0.20	0.24	0.28	0.35	0.44	0.18	0.21
9	0.24	0.19	0.24	0.28	0.33	0.44	0.18	0.21
10	0.24	0.19	0.24	0.27	0.34	0.40	0.18	0.22

computation is performed separately for each orthogonal component of the neural network output vector (i.e., Δv_x and Δv_y in this particular case) due to the fact that it measures the strength of the linear relationship between two sampled real-valued, scalar variables. The numbers in brackets denote the total number of target data samples for that particular user and dataset (i.e., training, validation, or test). From inspection of the tables, we can observe the high correlations that are evident at the lower prediction horizons for each user, coupled with the fact that as the prediction horizon increases, the capability of neural-network predictors to approximate the functional mapping (i.e., accuracy of the mapping between input to networks and the expected changes in entity velocity) decreases (in general). This is because they become unable to compensate for the increasing gap in available information that exists between the current time-step and the prediction horizon. Overall, neural-network predictors appear to exhibit good generalization on both the validation and test sets (with the test set being the most important) for each user relative to their performance on the respective training set for each user.

Table VI. Seconds Required to Compute 10^6 Iterations

Components	Workstation 1	Workstation 2	Laptop
Manufacturer	Intel	AMD	Intel
Processor	Pentium	Athlon	Pentium
Model / Speed	4 / 2.66 GHz	Barton (10) / 1 GHz	M 750 / 1.86 GHz
System Bus Clock	533 MHz	200 MHz	533 MHz
RAM / Bus Speed	1024 MB / 333 MHz	512 MB / 333 MHz	1024 MB / 400 MHz
Operating System	Win 2000 Pro	Win XP Pro	Win XP Pro
Compiler	MS Visual C++ 6.0	MS Visual C++ 6.0	MS Visual C++ 6.0
Execution Time	9.95	11.025	10.2

4.4 Resource Usage Results

Based upon the theoretical computational analysis in Section 3.4, we can now derive the total number of (approximate) floating-point operations (or flops) needed to execute our particular implementation. By substituting the parameter values defined in Section 4.2 (i.e., $d = 2$, $m = 8$, $k = 3$, and $q = 10$) into the counts defined in Table II and summing the results, we arrive at a total number of (approximate) flops needed to execute a single iteration of our neuro-reckoning procedure that is given by $\mathrm{T} = 3491 + 80\mathrm{T}_e + 2\mathrm{T}_s$. It should be noted, of course, that such an approximation is a conservative estimate that ignores the processing overhead associated with implementation details such as function calling and memory allocation (among others). The results of sequentially executing our neuro-reckoning procedure over 1 million iterations (10^6) in native compiled C++ source-code are summarized in Table VI. All computations were performed in double (64-bit) precision. As can be seen, a single iteration of our neuro-reckoning procedure takes approximately 10μs–11μs to execute on modern hardware (obtained by dividing the reported execution times by 10^6). Assuming a conservative estimate of anywhere between 10ms and 500ms for the mean network latency or round-trip time (RTT) delay in a modern DIA, a solitary execution of our proposed neuro-reckoning procedure results in only a 0.002% to 0.1% increase in delay times (for 500ms and 10ms, respectively).

In terms of additional memory requirements, each neural network predictor requires the storage of two interconnection weight matrices (i.e., $\mathbf{W}_i \in \Re^{m\times n}$ and $\mathbf{U}_i \in \Re^{d\times m}$) and two bias column vectors (i.e., $\mathbf{b}_i \in \Re^{m\times 1}$ and $\mathbf{c}_i \in \Re^{d\times 1}$) for a total of 154 parameters per network or 1540 parameters per set of networks (assuming $q = 10$). Assuming a single double (64-bit) storage requirement per parameter gives a combined total of approximately 12KB of storage required per set of neural network predictors. Given a networked multiplayer computer game such as Unreal Tournament [McCoy et al. 2003], where each host machine connected to the network is responsible for managing and posting updates for a single virtual entity only, the additional computational requirements and memory overhead introduced by the use of our proposed neuro-reckoning approach are more than suitable for real-time use. Given a large-scale distributed virtual environment or military simulation such as DIS [IEEE 1995] or the DMTITE architecture [Stytz and Banks 2001], where each host machine connected to the network may be responsible for managing and posting updates for large

Table VII. ESU Packet Generation Results for User 1 Over the Two Unbiased Datasets

Threshold	Trajectory A1				Trajectory B1			
	DR				DR			
(TWUs)	1st	2nd	NR	% Red	1st	2nd	NR	% Red
0.5	240	240	215	**−10.4**	191	180	162	**−10**
1	157	175	141	**−10.2**	123	128	114	**−7.3**
1.5	120	127	107	**−10.8**	95	104	83	**−12.6**
2	101	114	94	**−6.9**	86	100	77	**−10.5**
2.5	92	110	83	**−9.8**	72	81	73	1.4
3	81	99	72	**−11.1**	74	80	69	**−6.8**
3.5	78	94	72	**−7.7**	69	68	65	**−4.4**
4	69	87	68	**−1.4**	63	65	59	**−6.3**
4.5	66	79	58	**−12.1**	58	63	52	**−10.3**
5	59	75	54	**−8.5**	49	55	43	**−12.2**
Total	1063	1200	964	**−9.3**	880	924	797	**−9.4**

Table VIII. ESU Packet Generation Results for User 2 Over the Two Unbiased Datasets

Threshold	Trajectory A2				Trajectory B2			
	DR				DR			
(TWUs)	1st	2nd	NR	% Red	1st	2nd	NR	% Red
0.5	372	385	328	**−11.8**	356	356	307	**−17.5**
1	259	286	218	**−15.8**	240	266	206	**−20.5**
1.5	209	229	184	**−12**	196	222	173	**−17.2**
2	174	205	163	**−6.3**	170	188	145	**−16.7**
2.5	158	175	141	**−10.8**	145	169	128	**−19**
3	139	167	130	**−6.5**	139	155	118	**−15.1**
3.5	129	152	113	**−12.4**	123	144	108	**−16.3**
4	120	145	112	**−6.7**	115	125	103	**−14.2**
4.5	114	136	108	**−5.3**	106	124	94	**−17.5**
5	103	123	102	**−1**	93	116	84	**−18.4**
Total	1777	2003	1599	**−10**	1683	1865	1466	**−17.5**

numbers of simultaneous virtual entities or computer-generated forces (CGFs), up to 100 entities could theoretically execute our neuro-reckoning procedure (assuming a worst-case sequential execution scenario) in return for as little as 1ms of additional latency (i.e., 100 times 10μs).

4.5 Entity State Update Packet Generation Results

Shown in Tables VII to X are simulation results conducted over a series of increasing error thresholds for users 1 to 4, respectively. In each case, the simulations were performed over the 2 unbiased datasets (denoted as Trajectory A and Trajectory B) for that particular user, which were purposely kept isolated from the datasets used to train the neural network predictors, as described in Section 4.2. All presented error threshold values are measured in terms of

Table IX. ESU Packet Generation Results for User 3 Over the two Unbiased Datasets

Threshold (TWUs)	Trajectory A3 DR 1st	Trajectory A3 DR 2nd	Trajectory A3 NR	Trajectory A3 % Red	Trajectory B3 DR 1st	Trajectory B3 DR 2nd	Trajectory B3 NR	Trajectory B3 % Red
0.5	396	376	340	−**9.6**	220	210	179	−**14.8**
1	274	272	225	−**17.3**	150	141	109	−**22.7**
1.5	217	217	177	−**18.4**	122	119	89	−**25.2**
2	187	191	152	−**18.7**	102	100	85	−**15**
2.5	163	170	139	−**14.7**	88	85	79	−**7.1**
3	147	160	133	−**9.5**	76	77	68	−**10.5**
3.5	134	142	121	−**9.7**	74	74	61	−**17.6**
4	126	134	115	−**8.7**	68	68	58	−**14.7**
4.5	121	125	104	−**14**	64	65	55	−**14.1**
5	107	116	98	−**8.4**	56	68	53	−**5.4**
Total	1872	1903	1604	−**14.3**	1020	1007	836	−**17**

Table X. ESU Packet Generation Results for User 4 Over the Two Unbiased Datasets

Threshold (TWUs)	Trajectory A4 DR 1st	Trajectory A4 DR 2nd	Trajectory A4 NR	Trajectory A4 % Red	Trajectory B4 DR 1st	Trajectory B4 DR 2nd	Trajectory B4 NR	Trajectory B4 % Red
0.5	173	167	151	−**9.6**	187	181	150	−**17.1**
1	110	130	102	−**7.3**	119	121	107	−**10.1**
1.5	83	103	82	−**1.2**	101	98	86	−**12.2**
2	74	83	63	−**14.9**	86	88	72	−**16.3**
2.5	60	73	56	−**6.7**	70	75	63	−**10**
3	57	70	52	−**8.8**	65	66	57	−**12.3**
3.5	50	64	48	−**4**	54	62	50	−**7.4**
4	50	58	50	0	50	51	43	−**14**
4.5	46	56	50	8.7	43	52	45	4.7
5	46	53	42	−**8.7**	45	49	41	−**8.9**
Total	749	857	696	−**7.1**	820	843	714	−**12.9**

torque world units (TWUs). To put the error thresholds into perspective, the height of an entity within our virtual test environment is approximately 2.3 TWUs. A transmission timeout (or heartbeat) value of 5 seconds was used. Ideal network conditions were assumed so as to compare the performance of the following scenarios:

(1) ESU packet generation using a first-order dead reckoning (DR) model, as defined by the IEEE DIS standard [IEEE 1995] and given by Eq. (2).
(2) ESU packet generation using a second-order dead reckoning (DR) model, as defined by the IEEE DIS standard [IEEE 1995] and given by Eq. (3).

Table XI. Total ESU Packet Generation Results for Users 1 and 2 Over All Datasets

Threshold	User 1				User 2			
	DR				DR			
(TWUs)	1st	2nd	NR	% Red	1st	2nd	NR	% Red
0.5	2216	2222	1945	−**12.2**	3551	3654	3183	−**10.4**
1	1504	1618	1355	−**9.9**	2405	2643	2120	−**11.9**
1.5	1179	1306	1054	−**10.6**	1918	2167	1710	−**10.8**
2	1001	1136	895	−**10.6**	1621	1887	1462	−**9.8**
2.5	878	976	814	−**7.3**	1430	1660	1284	−**10.2**
3	805	877	733	−**8.9**	1293	1513	1165	−**9.9**
3.5	724	807	662	−**8.6**	1182	1396	1057	−**10.6**
4	657	751	617	−**6.1**	1082	1295	994	−**8.1**
4.5	611	709	580	−**5.1**	1005	1238	919	−**8.6**
5	569	659	530	−**6.9**	953	1158	876	−**8.1**
Total	10144	11061	9185	−**9.5**	16440	18611	14770	−**10.2**

(3) ESU packet generation using our proposed first-order neuro-reckoning (NR) model, as previously described throughout Section 3.

In each table, the heading "% Red" refers to the percentage of reduction (or increase) in the number of ESU packets generated, where negative values (as highlighted in bold) are indicative of superior performance by our proposed neuro-reckoning approach against the best-performing standard dead reckoning model (i.e., either the first- or second-order model, whichever generates fewer packets) for that particular error threshold. All simulations were performed at a rate of 20Hz (i.e., the original sampling rate of the data) using a maximum prediction horizon of $q = 10$ for each set of neural network predictors.

From inspection of the results, we can observe the fact that the first-order DR model consistently outperforms the second-order DR model over many of the larger simulated error thresholds. This is as a direct consequence of the rapidly changing entity velocity (or acceleration) that is present in our first-person shooter (FPS) application domain [Palant et al. 2006; Pantel and Wolf 2002]. Further, it is noted that for the vast majority of simulated error thresholds (and particularly for tight ones), our proposed neuro-reckoning approach offers a reduction in the number of ESUs generated over the best-performing DR model that ranges from small (in the region of 2% or lower packet reductions) to very large relative bandwidth savings (in the region of 20% or even higher packet reductions). From inspection of the results, it is obvious that (as with any predictive contract mechanism) the reported percentage reductions are dependent on both the error threshold and the specific dynamics of the entity trajectory in a highly nonlinear fashion. Thus, quantifying the exact relationship between error threshold and reported packet reduction rates is a difficult (if not impossible) task. For this reason, Tables XI and XII present the ESU packet generation results totalled over all recorded datasets for users 1 and 2 and users 3 and 4, respectively, in order to better gauge the expected performance benefits of our approach over an extended period. From inspection of these results, we can

Table XII. Total ESU Packet Generation Results for Users 3 and 4 Over All Datasets

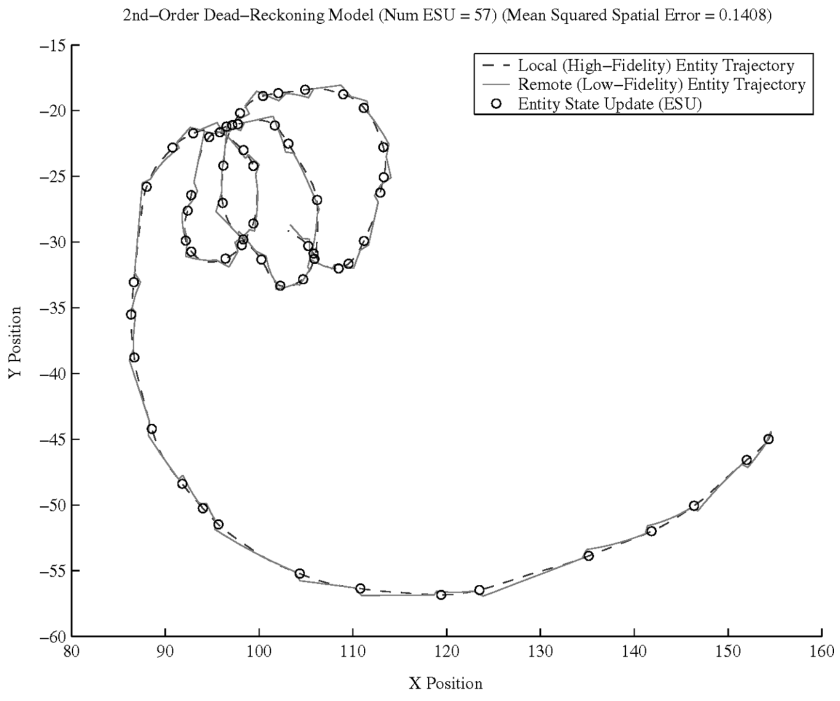
Threshold (TWUs)	User 3				User 4			
	DR				DR			
	1st	2nd	NR	% Red	1st	2nd	NR	% Red
0.5	3665	3462	3178	−**8.2**	2100	2031	1751	−**13.8**
1	2552	2502	2033	−**18.7**	1385	1479	1233	−**11**
1.5	2041	2017	1637	−**18.8**	1104	1181	973	−**11.9**
2	1724	1772	1424	−**17.4**	927	1026	826	−**10.9**
2.5	1526	1583	1280	−**16.1**	802	896	737	−**8.1**
3	1376	1456	1178	−**14.4**	733	817	673	−**8.2**
3.5	1261	1336	1077	−**14.6**	655	747	612	−**6.6**
4	1160	1254	1008	−**13.1**	600	682	567	−**5.5**
4.5	1094	1171	939	−**14.2**	565	643	541	−**4.2**
5	1016	1118	904	−**11**	539	596	490	−**9.1**
Total	17415	17671	14658	−**15.8**	9410	10098	8403	−**10.7**

immediately note the larger packet reductions typically observed at lower error thresholds, implying an excellent potential for use in applications requiring a relatively high degree of tight synchronization between locally and remotely modeled entity state. In general, we can observe a noticeable trend between both decreasing percentage reductions and increasing error threshold, which suggests a relatively stable predictive accuracy with respect to gross performance gain (i.e., expected overall packet reduction) resulting from the use of our proposed neuro-reckoning approach in contrast to standard dead reckoning algorithms. It should be noted that although the number of users limits our ability to generalize the results, it is sufficient for demonstrating the efficacy of the neuro-reckoning process for cases where good neural network predictors are available.

Figure 7 presents a sample dataset recorded during the trials performed by user 2, illustrating the same entity trajectory reconstructed using both the standard second-order dead reckoning (DR) model (the second-order which performed slightly better than the first-order one for this particular case) (shown in the top of the figure), and our proposed first-order neuro-reckoning (NR) model (shown bottom). From inspection of the two plots, we can observe how the neuro-reckoning model accurately compensates for expected changes in the future entity trajectory, significantly reducing both the number of ESUs and the mean squared spatial error (shown above each plot) of the reconstructed entity trajectory in comparison with the IEEE DIS standard dead reckoning model.

4.6 Performance Comparison Between Sets of Neural-Network Predictors

Shown in Tables XIII to XVI are simulation results comparing the performance of each set of neural network predictors over a series of increasing error thresholds for users 1 to 4, respectively (as outlined in the previous section). In each case (and with respect to each particular user), the simulations were performed using each of the three alternate sets of neural networks that were trained using



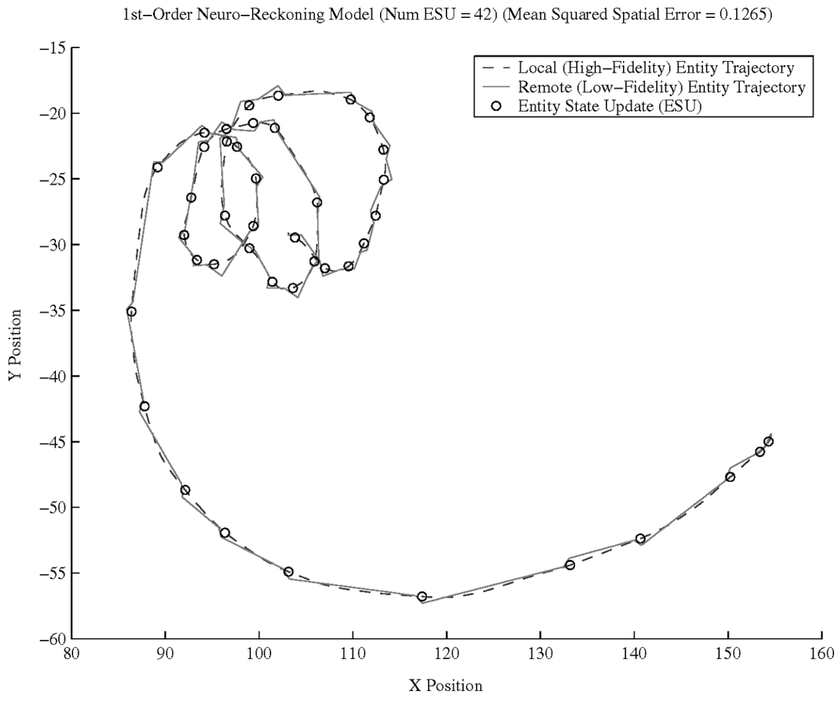


Fig. 7. Top: second-order dead reckoning (DR) model; bottom: first-order neuro-reckoning (NR) model.

Table XIII. Total ESU Packet Generation Results for User 1 Over All Datasets

	User 1							
Threshold	DR		User 2 NNs		User 3 NNs		User 4 NNs	
(TWUs)	1st	2nd	NR	% Red	NR	% Red	NR	% Red
0.5	2216	2222	1991	−**10.2**	2063	−**6.9**	1926	−**13.1**
1	1504	1618	1352	−**10.1**	1374	−**8.6**	1334	−**11.3**
1.5	1179	1306	1030	−**12.6**	1084	−**8.1**	1049	−**11**
2	1001	1136	879	−**12.2**	930	−**7.1**	883	−**11.8**
2.5	878	976	794	−**9.6**	814	−**7.3**	818	−**6.8**
3	805	877	720	−**10.6**	739	−**8.2**	735	−**8.7**
3.5	724	807	638	−**11.9**	660	−**8.8**	663	−**8.4**
4	657	751	592	−**9.9**	608	−**7.5**	614	−**6.5**
4.5	611	709	563	−**7.9**	559	−**8.5**	574	−**6.1**
5	569	659	524	−**7.9**	523	−**8.1**	527	−**7.4**
Total	10144	11061	9083	−**10.5**	9354	−**7.8**	9123	−**10.1**

Table XIV. Total ESU Packet Generation Results for User 2 Over All Datasets

	User 2							
Threshold	DR		User 1 NNs		User 3 NNs		User 4 NNs	
(TWUs)	1st	2nd	NR	% Red	NR	% Red	NR	% Red
0.5	3551	3654	3226	−**9.2**	3347	−**5.7**	3189	−**10.2**
1	2405	2643	2169	−**9.8**	2172	−**9.7**	2135	−**11.2**
1.5	1918	2167	1764	−**8**	1725	−**10.1**	1740	−**9.3**
2	1621	1887	1502	−**7.3**	1478	−**8.8**	1465	−**9.6**
2.5	1430	1660	1342	−**6.2**	1304	−**8.8**	1315	−**8**
3	1293	1513	1195	−**7.6**	1166	−**9.8**	1183	−**8.5**
3.5	1182	1396	1096	−**7.3**	1087	−**8**	1097	−**7.2**
4	1082	1295	1020	−**5.7**	1004	−**7.2**	992	−**8.3**
4.5	1005	1238	958	−**4.7**	934	−**7.1**	949	−**5.6**
5	953	1158	901	−**5.5**	891	−**6.5**	886	−**7**
Total	16440	18611	15173	−**7.7**	15108	−**8.1**	14951	−**9.1**

the data collected for each of the other three human-users. Thus, the generality of the neural networks can be properly ascertained. Once again, the term "% Red" refers to the percentage of reduction (or increase) in the number of ESUs generated with respect to our proposed neuro-reckoning approach in comparison with the best-performing dead reckoning model for that particular error threshold. The packet counts are generated and totalled over all recorded trajectories for that particular user, and the results can be compared with those generated using the original sets of neural networks, as presented previously in Tables XI and XII.

From inspection of the results, it is immediately noted that all three alternate sets of neural networks offer a reduction in the number of ESU packets generated for each individual human-user in comparison with those generated

Table XV. Total ESU Packet Generation Results for User 3 Over All Datasets

	User 3							
	DR		User 1 NNs		User 2 NNs		User 4 NNs	
Threshold (TWUs)	1st	2nd	NR	% Red	NR	% Red	NR	% Red
0.5	3665	3462	3083	**−10.9**	2994	**−13.5**	2974	**−14.1**
1	2552	2502	2153	**−13.9**	2038	**−18.5**	2085	**−16.7**
1.5	2041	2017	1794	**−11.1**	1678	**−16.8**	1730	**−14.2**
2	1724	1772	1543	**−10.5**	1449	**−16**	1496	**−13.2**
2.5	1526	1583	1379	**−9.6**	1288	**−15.6**	1347	**−11.7**
3	1376	1456	1259	**−8.5**	1195	**−13.2**	1230	**−10.6**
3.5	1261	1336	1166	**−7.5**	1105	**−12.4**	1138	**−9.8**
4	1160	1254	1072	**−7.6**	1038	**−10.5**	1062	**−8.4**
4.5	1094	1171	1013	**−7.4**	952	**−13**	987	**−9.8**
5	1016	1118	958	**−5.7**	914	**−10**	929	**−8.6**
Total	17415	17671	15420	**−11.5**	14651	**−15.9**	14978	**−14**

Table XVI. Total ESU Packet Generation Results for User 4 Over All Datasets

	User 4							
	DR		User 1 NNs		User 2 NNs		User 3 NNs	
Threshold (TWUs)	1st	2nd	NR	% Red	NR	% Red	NR	% Red
0.5	2100	2031	1864	**−8.2**	1775	**−12.6**	1918	**−5.6**
1	1385	1479	1256	**−9.3**	1193	**−13.9**	1264	**−8.7**
1.5	1104	1181	1009	**−8.6**	957	**−13.3**	979	**−11.3**
2	927	1026	865	**−6.7**	816	**−12**	844	**−9**
2.5	802	896	758	**−5.5**	723	**−9.9**	727	**−9.4**
3	733	817	673	**−8.2**	670	**−8.6**	661	**−9.8**
3.5	655	747	633	**−3.4**	615	**−6.1**	613	**−6.4**
4	600	682	593	**−1.2**	566	**−5.7**	569	**−5.2**
4.5	565	643	557	**−1.4**	524	**−7.3**	541	**−4.2**
5	539	596	505	**−6.3**	498	**−7.6**	500	**−7.2**
Total	9410	10098	8713	**−7.4**	8337	**−11.4**	8616	**−8.4**

using the best-performing dead reckoning model, at each particular error threshold. These reductions are observed over the entire range of simulated error thresholds, and would seem to suggest good general predictive capability on the part of each set of neural networks resulting from similar behavioral patterns emerging for each user. Such a result can likely be explained as follows: After an initial period spent exploring various transient strategies, users typically converge towards similar steady-state strategies associated with achieving their goal [Delaney et al. 2003]. In other words, users tend to adopt strategies that yield a high probability of success in return for a low investment of personal effort. Such an observation can be interpreted as a clear example of the *minimum energy principle* and can be seen in many virtual environments where users are presented with repetitive tasks [Sas et al. 2004]. Interestingly, the results seem largely independent of prior user experience.

Table XVII. Best Performing Maximum Prediction Horizon at Each Error Threshold

Threshold	User 1		User 2		User 3		User 4	
(TWUs)	NR	q	NR	q	NR	q	NR	q
0.5	1936	8	3100	6	2922	6	1751	10
1	1354	9	2109	8	2018	8	1227	9
1.5	1054	10	1692	9	1637	10	973	10
2	895	10	1460	9	1424	10	826	10
2.5	813	9	1284	10	1280	10	737	10
3	733	10	1165	10	1178	10	667	8
3.5	662	10	1057	10	1077	10	612	10
4	610	9	993	9	1008	10	567	10
4.5	580	10	919	10	939	10	541	10
5	530	10	876	10	904	10	490	10
Total	9185	10	14770	10	14613	8	8403	10

From further inspection of Tables XIII to XVI, it can also be observed that for several of the human-users, superior performance results (in terms of overall packet reduction totalled over all simulated error thresholds) are actually achieved using at least one of the alternate sets of neural networks (that were trained on the data collected from each of the other three users) over those achieved using the original sets (as presented previously in Tables XI and XII). Such a result implies that while similar behavioral patterns were observed for each user, larger quantities of the associated dynamics were captured by certain neural networks over others. These results would appear to validate the plausibility of training general sets of neural network predictors for scenarios involving multiple user-convergence to a collection of steady-state strategies.

4.7 Analyzing the Choice of Maximum Prediction Horizon

Presented in Table XVII are simulation results conducted over a series of increasing error thresholds for users 1 to 4. The packet counts are generated and totalled over all recorded trajectories for each particular user, using the best-performing prediction horizon q at each error threshold (up to a maximum of $q = 10$). In each case, the best-performing prediction horizons are listed under the column entitled "q". Once again, these results can be compared to the original results (generated using a uniform maximum prediction horizon of $q = 10$), as presented in Tables XI and XII for users 1 and 2 and users 3 and 4, respectively.

From inspection of the results, it is noted that the best performance gains (i.e., in terms of greatest reduction in total number of packets generated) are typically observed at higher maximum prediction horizons (i.e., $q = 8$ to $q = 10$) for the majority of simulated error thresholds. The exceptions appear to occur for the lower simulated error thresholds, implying that for a tight error threshold, superior performance results may be achieved by lowering the maximum prediction horizon from the default setting of $q = 10$ (as used in our particular implementation) to something smaller (e.g., a setting of $q = 6$ gives the best performance gains for users 2 and 3).

A suitable choice of maximum prediction horizon q bears consequence for the local resource requirements (i.e., number of neural network predictors) and thus the remote predictive accuracy of our proposed neuro-reckoning approach. Unfortunately, there does not appear to be any way of quantitatively predetermining the choice of maximum prediction horizon to use for a given error threshold in advance, as results can vary considerably. From initial inspection, it would appear a reasonable choice to assume the use of a longer maximum prediction horizon (e.g., $q \geq 8$), as these would appear to provide a more constant performance gain over both tight and relaxed error thresholds.

5. CONCLUSIONS AND FUTURE WORK

We have presented an extension of dead reckoning, termed neuro-reckoning, that seeks to replace the traditional use of instantaneous derivative information with predictive derivative information for network traffic reduction in DIAs. By estimating expected future entity dynamics in advance using a series of trained neural network predictors, and distributing this information to remote hosts in the form of a predictive velocity vector, our proposed neuro-reckoning approach succeeds in reducing the spatial error associated with remote extrapolation of entity position. Consequently, a further reduction in the number of ESUs needed to maintain consistency of shared state can be achieved.

Presented simulation results validate the potential of our proposed framework for improving the accuracy of remote extrapolation and for further reducing network traffic over the use of the IEEE DIS standard dead reckoning procedure throughout a large majority of our experimental setups. A reduction in network bandwidth requirements over a wide range of error thresholds indicates viable potential for satisfying many spectrums of consistency requirements throughout a broad range of real-time distributed applications. Furthermore, presented computational analysis results indicate the potential of our proposed framework for real-time use in a variety of distributed applications, requiring only a relatively modest amount of additional computational resources [Chui and Heng 2004].

Our proposed neuro-reckoning approach is based upon the reasonable assumption that previously observed patterns of human-user behavior provide an accurate indication of future human-user behavior, and that these patterns may be learnable by employing statistical machine-learning techniques [Zukerman and Albrecht 2001]. In situations where the previous assumption does not hold true, neuro-reckoning may exhibit poor predictive performance, consequently leading to a possible increase (rather than reduction) in the number of ESUs generated due to error threshold violation.

Future work shall include the investigation of confidence-interval estimation methods in order to quantify the reliability of the predictions made by each set of neural network predictors [Papadopoulos et al. 2001], providing for a more general framework capable of gracefully resorting to use of the standard dead reckoning protocol in situations where the accuracy of neural-network predictors is questionable. For scenarios in which discrete classes of entity dynamics may be identified, the investigation of automatic context-switching schemes [Henninger et al. 2001] based on input state-space clustering [Thurau et al. 2003] may prove beneficial in providing an improvement in the prediction accuracy and learning capability of neural networks. Finally, future work shall also involve the investigation of alternative neural network topologies and machine-learning techniques, alongside a further analysis on the complex relationships between stability and expected performance gain versus the tightness of the error threshold.

REFERENCES

BATTAGLIA, G. J. 1996. Mean square error. *AMP J. Technol. 5*, 31–36.

BEHNKE, S., EGOROVA, A., GLOYE, A., ROJAS, R., AND SIMON, M. 2003. Predicting away robot control latency. In *Proceedings of the 7th Robocup International Symposium* (Padua, Italy, Jul.). 712–719.

BHOLA, S., BANAVAR, G., AND AHAMAD, M. 1998. Responsiveness and consistency tradeoffs in interactive groupware. In *Proceedings of the 7th ACM Conference on Computer Supported Cooperative Work (CSCW 9)* (Seattle, WA, Nov.). 79–88.

BONÉ, R. AND CRUCIANU, M. 2002. Multistep-Ahead prediction with neural networks: A review. *Approches Connexionnistes en Sciences Économiques et en Gestion 2*, 97–106.

CAI, W., LEE, F. B. S., AND CHEN, L. 1999. An auto-adaptive dead reckoning algorithm for distributed interactive simulation. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation (PADS)* (Atlanta, GA, May). 82–89.

CAWLEY, G. C. 2000. On a fast, compact approximation of the exponential function. *Neural Comput. 12*, 9, 2009–2012.

CHEN, L. AND CHEN, G. 2005. A fuzzy dead reckoning algorithm for distributed interactive applications. In *Proceedings of the 2nd International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)* (Changsha, China, Aug.). 961–971.

CHUI, Y.-P. AND HENG, P.-A. 2004. Attitude dead reckoning in a collaborative virtual environment using cumulative polynomial extrapolation of quaternions. *Concur. Comput. Practice Exper. 16*, 5, 1575–1599.

CHURCHILL, E. F., SNOWDON, D. N., AND MUNRO, A. J. 2001. *Collaborative Virtual Environments: Digital Places and Spaces for Interaction*. Springer, London.

DELANEY, D., WARD, T., AND MCLOONE, S. 2006a. On consistency and network latency in distributed interactive applications: A survey—Part I. *Presence: Teleoper. Virtual Environ. 15*, 2, 218–234.

DELANEY, D., WARD, T., AND MCLOONE, S. 2006b. On consistency and network latency in distributed interactive applications: A survey—Part II. *Presence: Teleoper. Virtual Environ. 15*, 4, 465–482.

DELANEY, D., WARD, T., AND MCLOONE, S. 2003. Reducing update packets in distributed interactive applications using a hybrid model. In *Proceedings of the 16th ISCA International Conference on Parallel and Distributed Computing Systems (PDCS)* (Reno, NV, Aug.). 417–422.

FUJIMOTO, R. M. 2000. *Parallel and Distributed Simulation Systems*. Wiley Interscience, New York.

GREENBERG, S. AND MARWOOD, D. 1994. Real time groupware as a distributed system: Concurrency control and its effect on the interface. In *Proceedings of the 5th ACM Conference on Computer Supported Cooperative Work (CSCW)* (Chapel Hill, NC, Oct.). 207–217.

HAGAN, M. T. AND MENHAJ, M. B. 1994. Training feedforward networks with the marquardt algorithm. *IEEE Trans. Neural Netw. 5*, 6, 989–993.

HARRINGTON, P. B. 1993. Sigmoid transfer functions in backpropagation neural networks. *Analyt. Chem. 65*, 15, 2167–2168.

HENNINGER, A. E., GONZALEZ, A. J., GEORGIOPOULOS, M., AND DEMARA, R. F. 2001. A connectionist-symbolic approach to modeling agent behavior: Neural networks grouped by contexts. In *Proceedings of the 3rd International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT)* (Dundee, Scotland, Jul.). 198–209.

IEEE 1995. *Standard for Distributed Interactive Simulation—Application Protocols*. *IEEE Standard 1278.1–1995*. Institute of Electrical and Electronics Engineers, Piscataway, NJ.

LEE, B.-S., CAI, W., TURNER, S. J., AND CHEN, L. 2000. Adaptive dead reckoning algorithms for distributed interactive simulation. *Int. J. Simul. Syst. Sci. Technol. 1*, 1-2, 21–34.

LIN, K.-C., WANG, M., WANG, J., AND SCHAB, D. E. 1995. The smoothing of dead reckoning image in distributed interactive simulation. In *Proceedings of the AIAA Flight Simulation Technologies Conference* (Baltimore, MD, Aug.). 83–87.

LIN, K.-C. AND SCHAB, D. E. 1994a. Study on the network load in distributed interactive simulation. In *Proceedings of the AIAA Flight Simulation Technologies Conference* (Scottsdale, AZ, Aug.). 202–209.

LIN, K.-C. AND SCHAB, D. E. 1994b. The performance assessment of the dead reckoning algorithms in DIS. *Simul. 63*, 5, 318–325.

LLOYD, J. 2004. The torque game engine. *Game Devel. Mag. 11*, 8, 8–9.

MACEDONIA, M. R. AND ZYDA, M. J. 1997. A taxonomy for networked virtual environments. *IEEE Multimedia 4*, 1, 48–56.

MATHY, L., EDWARDS, C., AND HUTCHINSON, D. 1999. Principles of QoS in group communications. *Telecommun. Syst. 11*, 1-2, 59–84.

MCCOY, A., DELANEY, D., MCLOONE, S., AND WARD, T. 2005. Dynamic hybrid strategy models for networked multiplayer games. In *Proceedings of the 19th European Conference on Modelling and Simulation (ECMS)* (Riga, Latvia, Jun.). 727–732.

MCCOY, A., DELANEY, D., MCLOONE, S., AND WARD, T. 2004. Investigating behavioural state data-partitioning for user-modelling in distributed interactive applications. In *Proceedings of the 8th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT)* (Budapest, Hungary, Oct.). 74–82.

MCCOY, A., DELANEY, D., AND WARD, T. 2003. Game-State fidelity across distributed interactive games. *ACM Crossroads 9*, 4, 4–9.

MELLON, L. AND WEST, D. 1995. Architectural optimizations to advanced distributed simulation. In *Proceedings of the 27th Winter Simulation Conference (WSC)* (Arlington, VA, Dec.). 634–641.

MILLER, D. C. AND THORPE, J. A. 1995. SIMNET: The advent of simulator networking. *Proc. IEEE 83*, 8, 1114–1123.

MIZUTANI, E. AND DREYFUS, S. E. 2001. On complexity analysis of supervised MLP-learning for algorithmic comparisons. In *Proceedings of the 14th INNS-IEEE International Joint Conference on Neural Networks (IJCNN)* (Washington, DC, Jul.). 347–352.

PALANT, W., GRIWODZ, C., AND HALVORSEN, P. 2006. Evaluating dead reckoning variations with a multi-player game simulator. In *Proceedings of the 16th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)* (Newport, RI, May). 20–25.

PANTEL, L. AND WOLF, L. C. 2002. On the suitability of dead reckoning schemes for games. In *Proceedings of the 1st ACM Workshop on Network and System Support for Games (NetGames)* (Braunschweig, Germany, Apr.). 79–84.

PAPADOPOULOS, G., EDWARDS, P. J., AND MURRAY, A. F. 2001. Confidence estimation methods for neural networks: A practical comparison. *IEEE Trans. Neural Netw. 12*, 6, 1278–1287.

PRINCIPE, J. C., EULIANO, N. R., AND LEFEBVRE, W. C. 2000. *Neural and Adaptive Systems: Fundamentals Through Simulations*. John Wiley, New York.

RODGERS, J. L. AND NICEWANDER, W. A. 1988. Thirteen ways to look at the correlation coefficient. *Ameri. Statis. 42*, 1, 59–66.

ROEHLE, B. 1997. Channeling the data flood. *IEEE Spectrum 34*, 3, 32–38.

RYAN, P. AND OLIVER, W. 2006. Modelling of dead reckoning and heartbeat update mechanisms in distributed interactive simulation. In *Proceedings of the European Simulation Interoperability Workshop* (Stockholm, Sweden, Jun.). 06E-SIW-025.

SAS, C., O'HARE, G., AND REILLY, R. 2004. A performance analysis of movement patterns. In *Proceedings of the 4th International Conference on Computational Science (ICCS)* (Krakow, Poland, Jun.). 954–961.

SAS, C., REILLY, R., AND O'HARE, G. 2003. A connectionist model of spatial knowledge acquisition in a virtual environment. In *Proceedings of the 2nd Workshop on Machine Learning, Information Retrieval and User Modelling (MLIRUM)* (Johnstown, PA, Jun.). 40–48.

SHANKER, M., HU, M. Y., HUNG, M. S. 1996. Effect of data standardization on neural network training. *OMEGA (Int. J. Manage. Sci.) 24*, 4, 385–397.

SHIM, K.-H. AND KIM, J.-S. 2001. A dead reckoning algorithm with variable threshold scheme in networked virtual environment. In *Proceedings of the IEEE International Conference on Systems, Management and Cybernetics (SMC)* (Tucson, AZ, Oct.). 1113–1118.

SINGHAL, S. K. AND ZYDA, M. J. 1999. *Networked Virtual Environments: Design and Implementation*. Addison-Wesley, Reading, MA.

SRINIVASAN, S. 1996. Efficient data consistency in HLA/DIS++. In *Proceedings of the 28th Winter Simulation Conference (WSC)* (Coronado, CA, Dec.). 946–951.

STYTZ, M. R. AND BANKS, S. B. 2001. The distributed mission training integrated threat environment system architecture and design. *ACM Trans. Model. Comput. Simul. 11*, 1, 106–133.

SUN, C., JIA, X., ZHANG, Y., YANG, Y., AND CHEN, D. 1998. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Trans. Comput.-Hum. Interact. 5*, 1, 63–108.

THURAU, C., BAUCKHAGE, C., AND SAGERER, G. 2003. Combining self organizing maps and multilayer perceptrons to learn bot-behavior for a commercial game. In *Proceedings of the 4th International Conference on Intelligent Games and Simulation (GAME-ON)* (London, Nov.). 119–123.

VAN HOOK, D. J., CALVIN, J. O., AND SMITH, J. E. 1995. Data consistency mechanisms to support distributed simulation. In *Proceedings of the 12th Workshop on Standards for the Interoperability of Distributed simulations* (Orlando, FL, Mar.). 797–806.

YU, S.-J. AND CHOY, Y.-C. 2001. A dynamic message filtering technique for 3D cyberspaces. *Comput. Commun. 24*, 18, 1745–1758.

ZHANG, M.-J. AND GEORGANAS, N. D. 2004. An orientation update message filtering algorithm in collaborative virtual environments. *J. Comput. Sci. Technol. 19*, 3, 423–429.

ZUKERMAN, I. AND ALBRECHT, D. W. 2001. Predictive statistical models for user modeling. *User Model. User-Adapt. Interact. 11*, 1-2, 5–18.

Received January 2006; revised March 2007; accepted March 2007