

Using Context-Based Neural Networks to Maintain Coherence Among Entities' States in a Distributed Simulation

Avelino J. Gonzalez

Michael Georgiopoulos

Ronald F. DeMara

Intelligent Systems Laboratory

School of Electrical Engineering and Computer Science

University of Central Florida

Orlando, FL 32816-2450

gonzalez@ucf.edu

For entities to interact meaningfully in a distributed simulation environment, coherence among the entities' states must be maintained. Because continuous state updates for all entities in the simulation normally require large amounts of network bandwidth, motion equations (i.e., dead-reckoning models) are frequently used to reduce the number of communications updates. However, even with the use of such dead-reckoning models, networking and communications limitations still exist in currently fielded systems. An effective approach to reducing the communications requirements is achieved by replacing these predictive dead-reckoning models with neural networks. This paper presents the background and motivation for the research, the architecture and training algorithms of the networks, and the integration of the networks into a large-scale simulation environment. Quantitative measures from the experiments reveal that the use of neural networks can effectively reduce the number of communication updates required to maintain entity-state coherence. However, the neural networks may also be more difficult to scale than the currently used dead-reckoning algorithms.

Keywords: distributed systems, distributed simulation, communications requirements, dead reckoning, neural networks

1. Introduction

The combination of computer simulation and networking technologies has provided an effective means for training humans in large scale, collaborative tactical tasks through the use of Distributed Interactive Simulation (DIS). DIS is an architecture for building large-scale simulations from a set of independent simulator nodes [1] each representing one or more *entities* (e.g., an automobile, an aircraft, a platoon of soldiers) in the simulation. By communicating over a network via a common protocol, these entities are able to exist simultaneously and interact meaningfully in the same virtual environment while running in different simulations, often in different hardware. *Coherence* is the term used to indicate the ability of each of these distributed entities to know the location of all other entities with which it must be able to interact within the

simulation environment. For a successful distributed simulation exercise, coherence among the many distributed entities must be maintained at all times. Currently, however, the ability of live vehicles to interact with these virtual forces in mixed-reality embedded training applications is constrained by the communication bandwidth requirements needed for real-time interoperability.

To quantify the bandwidth requirements and deficits, a study [2] assessed the communication requirements needed to support embedded training and en-route mission rehearsal in a military context. The study simulates a general scenario of a battalion task force that has been rapidly deployed to a distant battlefield on eight aircraft. These aircraft fly in formation, each one carrying up to three ground vehicles. The aircraft are in communication with each other via a satellite link that also connects them to a ground station in the continental United States. This ground station provides core exercise support, including *computer generated forces* (CGFs) used to complement the scenario. In this environment, the available 64 Kbps bandwidth in the wireless links available to handle DIS traffic is shown to be insufficient for even small scale simulation exercises. Message latencies of more than 70 seconds were observed because of routing delays from simulation traffic. This traffic involved the ground vehicles aboard the aircraft and the satellite link to the ground station. Clearly, such large delays negatively impact the fidelity and, thereby, the feasibility of the embedded training exercises.

Techniques to actively manage bandwidth to reduce requirements are therefore sought. In distributed simulation, 50% to 80% of the network traffic can originate from updates transmitted to ensure coherence among the entities participating in the simulated exercise [3]. Hence, bandwidth is a critical resource, especially in embedded distributed simulations using wireless channels and ad hoc networks, where bandwidth becomes scarce as more entities participate in the distributed exercise [4, 5, 6].

Several attempts have been made to overcome bandwidth limitations in DIS. Approaches range from finding new methods or algorithms to reduce the network traffic, either by applying some lossless compression algorithms to that of elaborate subscribe/publish models of multicast communication [7]. The approach of concatenating logical data packets called *Protocol Data Units* (PDUs) into a larger physical packet that is later deconstructed at the destination into individual PDUs has been explored by Calvin *et al* [8]. Under their technique, a PDU is transmitted when either a timer expires or the aggregated packet reaches a maximum size. As a consequence, the necessary bit rates are reduced because fewer packet headers are transmitted, placing multiple PDUs in one single packet for transport. Ceranowicz *et al* [9] describe the Joint Experimental Federation and Millennium Challenge 2002, a simulation conducted in July and August of 2002 by 13,500 personnel at remote locations across the continental United States, where he reports the optimum limit of bytes that can be bundled. In one experiment, up to 4,500 bytes were bundled in each IP packet. He concludes that the tradeoffs were that bundling more data together increased latency and packet loss due to transmission errors, while smaller packets increased the transmission of overhead data. Bassiouni *et al* [10] also describe an approach to concatenate consecutive PDUs in a single packet even if their types are different, though redundancy in the fields that make up a PDU are not eliminated.

Other approaches partition PDUs into static and dynamic portions to allow transmission of the static data only once while the dynamic information are updated only as needed using *delta-PDUs*. This approach is described in DIS-Lite system which offers packet bundling and latency compensation for air vehicles [11]. However, it does not examine, take advantage nor learn from the type, timestamp and internal structure of PDUs or adapt to vehicle behaviors over time.

The remainder of this paper reviews the factors influencing network bandwidth and explains in greater detail how DIS addresses bandwidth constraints. It furthermore explains how neural networks can serve to improve the motion equation predictive models, and presents the results of our experiments supporting this claim.

2. Network Bandwidth Management

From the above discussion, mechanisms suitable for reduction of bandwidth requirement for distributed simulations include multicast schemes [12], packet bundling and replication strategies [13], protocol optimizations for DIS [14] and High-Level Architecture (HLA) [11]. Currently, most implementations of DIS employ a predictive model of vehicle movement called *dead reckoning* (DR) to reduce DIS packet traffic. The term "dead reckoning," borrowed from maritime navigation, describes the process by which the position of a ship can be estimated from an earlier known position, heading, speed, and elapsed time since that known position, assuming constant speed and heading. Dead reckoning has been expanded to include turn rates, but DIS currently employs the traditional dead reckoning with constant heading and speed. DIS dead reckoning models build on this concept by applying equations of motion to predict the state information of a simulated entity. Each entity in the environment maintains a dead reckoning model of each other entity in the environment with which it must interact. Therefore, it can predict the position, speed and direction of other entities with which it can interact without the need to continually receive such indications on the network. Each entity also internally maintains the same model of its own motion and compares its prediction to its true position, speed and direction. If a pre-determined error is exceeded, it proceeds to update all other entities in the network with its current true state. It does this via an *Entity State Protocol Data Unit*, or ESPDU. Clearly, dead reckoning has been instrumental in managing bandwidth requirements in existing DIS applications.

One DIS dead-reckoning model used to approximate an entity's position is given by the following equations of motion:

$$p = p_0 + (v_0 * \Delta t) + \frac{a_0 * (\Delta t)^2}{2} \quad (1)$$

$$v = v_0 + a_0(\Delta t)$$

where p = current position
 p_0 = initial position
 v = current velocity
 v_0 = initial velocity
 a_0 = initial acceleration
 Δt = elapsed time

More specifically, in DIS dead reckoning, each entity maintains simple, low fidelity dead-reckoning models such as equation (1) of its own state and of the state of all other entities with which it may interact. At periodic intervals, the predictions made by the models of their own state are compared to their own true positions. As illustrated in Figure 1 [15], if the difference between the entity's true position and the dead reckoning model's prediction exceeds some error

threshold, the local simulator will transmit an update of that entity vehicle's true state to the other simulators. This update takes the form of an ESPDU and contains information such as location, velocity, time of last update, etc. Upon the receipt of an ESPDU, the dead reckoning models will adjust to reflect the updated data and then continue to predict the near-term position and orientation of that entity from these data.

Because the number of ESPDUs required in a DIS distributed embedded training exercise depends directly on the predictive ability of the dead reckoning model, improving the predictive ability of an entity's state will directly result in the reduction of communications updates. This, in turn, will enable more entities to exist on the virtual battlefield for the same amount of network bandwidth.

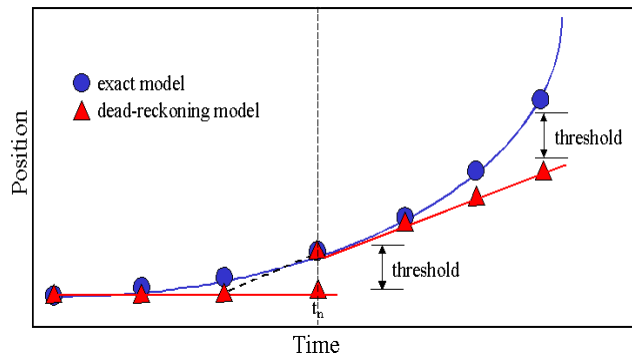


Figure 1 - DIS Dead-Reckoning Process [15]

Assume, for example, that there are three entities (A, B and C) in the distributed simulation, each residing in different simulation nodes linked by the DIS protocol. Further, assume that entities A, B and C need to know each others' state continuously during the simulation. Lacking any such bandwidth management methods, each entity would have to frequently and continually transmit ESPDUs containing its own state information to the other two entities in order to maintain coherence. However, with dead reckoning prediction, each entity (say, for example, entity A) communicates its own initial location, velocity and acceleration to the other entities (B and C) for use in their local predictive dead reckoning models of entity A. Entities B and C, using their dead reckoning models of entity A, can then predict its unperturbed near-term physical location without need for updates from A. Each entity also maintains a dead reckoning model of its own movement, predicting its own future position and speed. In the event that entity A deviates from its own predicted path, it will transmit an update of its new true state (location, direction and speed) to entities B and C for them to use in their predictive models of entity A. Entities B and C will do likewise with entity A, as well as with each other.

While the dead reckoning equations of motion approach has met with some success and it is widely used in DIS, it assumes that no external force acts to change the direction and speed of the entities involved. In other words, dead reckoning assumes that an entity moves in one direction at one speed constantly through the time period of interest. This is in fact not an accurate assumption, as the human driver (or intelligent program that "drives" the entity) represents an external force that can change the speed and direction of the entity in response to the environment, objectives and internal condition of the entity. To accurately predict the behavior of the human-controlled entity requires that we model the behavior of the controlling human as manifested in the observable actions of the entity. Therefore, a more responsive

predictive model that accounts for the presence of the human control of an entity is sought. The research presented in this paper addresses the improvement of the predictive utility of the dead reckoning model by using neural networks instead of equations of motion. Unlike the equations of motion in traditional dead reckoning, neural networks are able to predict motion that is not in a straight line, such as when vehicles round curves or maneuver to avoid obstacles. It also captures the human (external forces) that acts on the entity to change its speed and direction. Hence, we are seeking improved accuracy vis-à-vis how the (human) driver decides to control the entity. This is significant in mixed reality exercises when the location of live vehicles must be known to the virtual agents or humans in simulators in order to accurately target them. In this context, a difference of a few inches in the location of a live tank being targeted may determine whether it is hit and the effect of the hit.

3. Improving Predictive Models to Reduce Bandwidth

As shown in Figure 1, dead reckoning diverges whenever the entity does not follow a straight line. Because real entities rarely remain on a straight heading for very long, the utility of these dead reckoning predictive models can be inherently and severely limited. Neural networks, on the other hand, attempt to predict behavior based on past historical examples of similar performance. How accurate these predictions are can depend greatly on how repeatable the historical movements will be in the current simulation. Our approach to improving the management of network bandwidth involves replacing the dead-reckoning predictors in a DIS-based simulation system by a set of neural networks that learn from similar previous exercises. We employed past instances of entities performing similar maneuvers to train our networks. Furthermore, we experimented with the type and composition of the neural networks to find the best combination. We specifically compared a set of baseline neural networks to the results obtained by applying modular decomposition to the problem. That is, recognizing the situation unfolding before the entity and applying the neural network that best addresses that situation. This can be referred to as *context sensitive neural networks*.

There may be other alternative approaches besides neural networks that can improve the performance of dead reckoning models. In fact, state estimation techniques such as auto-regression, Kalman filters and others may prove to also be advantageous. However, the objective of this investigation was not to find the optimal replacement for dead reckoning, but rather, to evaluate whether neural networks, with their ability to learn from observation of past performance by others, could increase the predictive ability of the dead reckoning equations of motion. Moreover, it is true that the context-sensitive enhancements to the neural networks could also be extended to the dead reckoning algorithm. However, dead reckoning is the benchmark against which we want to measure our gains. Furthermore, by its very nature, dead reckoning implies straight ahead movement, making context shifting irrelevant.

The application on which we tested our approach focuses on the near-term movement behavior of a battlefield entity in a distributed simulation. In a mixed initiative, live-virtual distributed simulation, it is imperative that entities know the precise location of other entities with which they interact (coherence). This is particularly important for collaborative movement and targeting. In our implementation, we employed the Modular Semi-Automated Forces (ModSAF) v5.0 system [16] as the simulation infrastructure for our investigation. ModSAF is a DIS-based constructive simulation system designed to provide Computer-Generated Forces (CGF) for military training and research. Near-term movement behavior was selected because it

is highly observable and provides a direct correspondence to ESPDUs resulting from errors in entity position. This proves useful in finding a way to measure the performance of the approach conceived in this study as will be explained in the next section. However, since movement in the battlefield is a highly complex behavior that depends on many factors, the problem was scoped to specifically consider how a single entity (i.e., a ModSAF M1A2 tank) performs a *Road March*. A road march is a tactic by which a vehicle or group of vehicles (e.g., tanks, armored personnel carriers, trucks) moves on a defined path from waypoint to waypoint (usually on some road, although not necessarily paved) with the objective of reaching a pre-determined destination. Movement on a road march is typically done in single column formation and enemy presence is not expected (although possible). Parameters provided are the destination and possibly some intermediate waypoints. In this application, our system predicts the changes in an M1A2 entity's location and orientation given its own previous state as well as that of the simulated world, and compares these changes to those predicted by the DIS dead reckoning system.

ModSAF entities possess the capability to carry out some simple maneuvers such as road marches. It is from observation of a ModSAF entity performing this task that our neural networks learn the behavior that is later used for prediction. The ModSAF entities are programmed to act non-repeatably within a certain range of variability. The data logged for several runs of these entities were used to train the neural networks (in different ways), and the resulting neural networks were executed to measure the predictive effectiveness of our approach in subsequent runs.

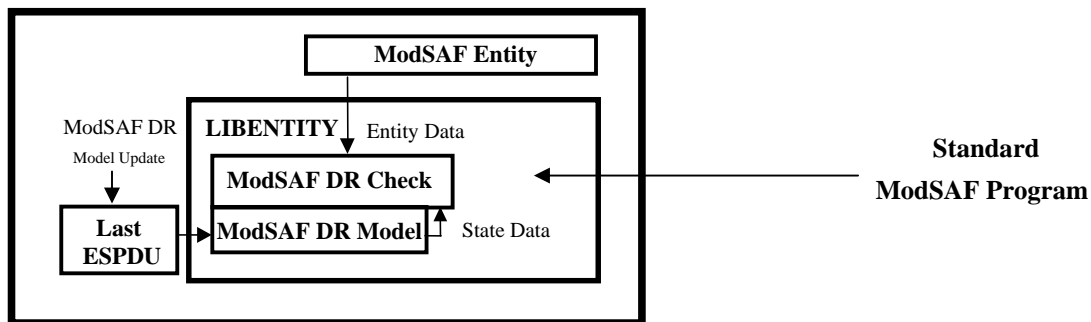


Figure 2 - Functional Relationship of Entity Control Models in ModSAF: Standard ModSAF with Dead-Reckoning Code

Figure 2 displays the standard ModSAF program architecture as it affects its DIS predictive feature. The comparison between the entity's true position and the position predicted by its intrinsic predictive equations (i.e., dead reckoning) occurs in ModSAF's *libentity* library, a program module that uses the dead reckoning equations of motion to predict the entity's location at any one time. As such, the neural network models used in this investigation simply replaced the dead reckoning code in the *libentity* library. The implementation of this functionality in ModSAF is illustrated in Figure 3, where the *libentity* library is replaced by the *libNN* library containing the neural networks and their execution engine.

The true position of the entity is contained in the ModSAF entity. This position (location and orientation) can be compared with the predictions of the two predictive models – the motion equations and our experimental neural networks - to determine when each predictive model's

prediction is significantly out of the acceptable error tolerance, and thus require the generation of an ESPDU. We often refer to them as *ghost* models to suggest their background position, unobtrusively keeping track of an entity’s movements and predicting where they would be on the next simulation cycle. This “ghost” process provides indication of when an ESPDU would be generated and transmitted by ModSAF. By keeping count and comparing the number of ESPDUs generated by the neural-network-based ghost model with the baseline dead reckoning ghost model, relative performance of the two predictive techniques can be compared. In other words, the measure of effectiveness is how many ESPDUs would have been generated with the neural network predictive system as compared to the dead reckoning method for the exact same circumstances. This comparison is particularly appropriate because it addresses the need to reduce the number of ESPDU transmissions, the reduction of which is the foremost objective of our work.

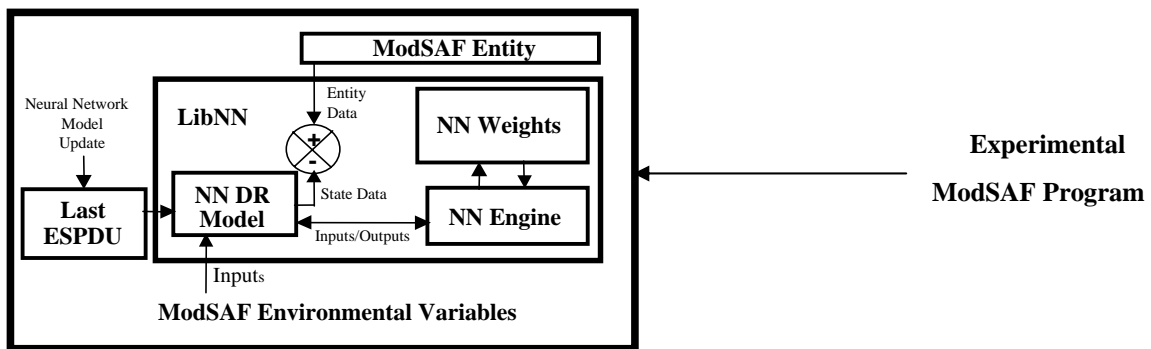


Figure 3 - Functional Relationship of Entity Control Models in ModSAF: Modified ModSAF with Neural Network Code

The neural networks used varied depending on the type of experiment being conducted. We discuss this by characterizing our work into seven different experimental phases. These are described as follows.

3.1 Phase I – data acquisition and benchmark DIS results

A data acquisition experiment was made to gather the requisite data for training and testing of the prototype neural networks of Phase II, as well as to serve as the benchmark DIS results against which the neural networks’ performance will be compared. In this experiment, a ModSAF entity was tasked with traveling through an extended road segment on a road march, and data on its state were collected throughout. The route used for this run (see black route in Figure 4) can be found in the section of terrain east of Barstow Road and west of Hill 720 in the National Training Center (NTC-0101) terrain database. In general, the route is represented by 45 route points and is approximately seven kilometers long. It takes the M1A2 tank entity about 15 minutes of simulation time to travel this route segment at 8 m/s. This included its location, speed and orientation. A total of 13,760 data slices were logged at a rate of 15 HZ during the benchmark run.

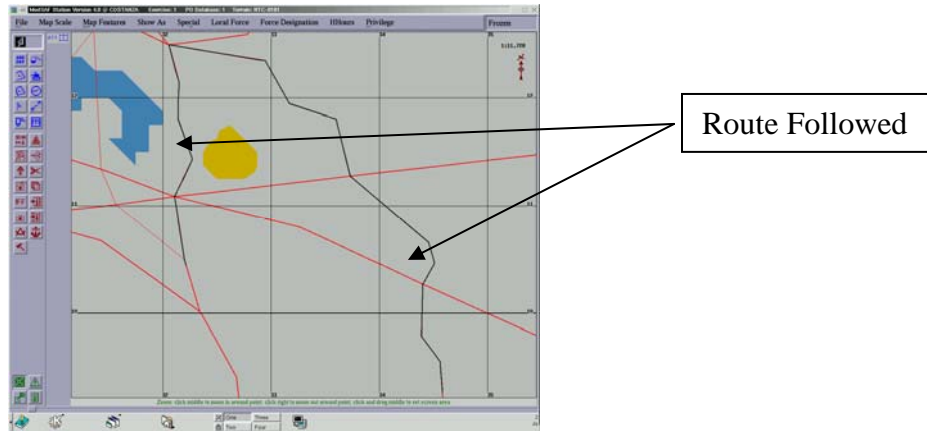


Figure 4 – Route used in Phase I of experiment

The benchmark results to which all other results are compared are those obtained from ModSAF executing its own DIS predictive model. The results are characterized by the number of ESPDUs ModSAF was forced to generate and transmit as a result of predictive inaccuracies by its DIS predictive functions. This is how ModSAF normally operates. The results of this benchmark experiment are shown in Table 1. They are segregated between deviations that result in the generation and transmission of an ESPDU as a result of a positional (location, or “Loc”) discrepancy and those generated because of a directional (rotational, or “Rot”) discrepancy. Note that the Loc and Rot deviations in Table 1 do not have to add up to the total number of ESPDUs, as one ESPDU may contain information updates for both location and rotation. In our calculations, this actually represents two different deviations from the model and should count as two, even if they were coincidentally reported as part of the same ESPDU.

Table 1 – Benchmark results of DIS predictive model

	Loc Deviations	Rot Deviations	Total Deviations	Tot. ESPDU
DIS equations	205	154	359	351

3.2 Phase II - Baseline neural network model and results in extended route segment

Phase II of this investigation focused on finding an effective modeling strategy for the neural network predictive model. Evaluation of this baseline model was done using the same extended route segment in the ModSAF virtual terrain used in the benchmark experiment of Phase I. For this Phase II baseline experiment, two sets of two neural networks each were used to predict the behavior of the entity traversing the same route. Of these four neural networks, two predict the change in the entity’s speed – one for straight-line sections and the other one for when the entity is in the midst of a turn. The other two neural networks predict the change in the entity’s orientation (Rot), one for straight-line road sections and the other for when the entity is in the

midst of a turn. Each neural network was of the recurrent, feed-forward architecture type with two hidden layers and a back-propagation gradient descent procedure in its training. Therefore:

- Network SS – straight road segment, change in entity speed
- Network SH – straight road segment, change in entity heading
- Network TS – turn segment, change in entity speed
- Network TH – turn segment, change in entity heading.

The rule used to distinguish between straight and turn segment types was acquired from the code used by ModSAF to generate the entity's behavior and is based on the straight-line distance to the next waypoint. Each of these situations can be considered to be different contexts in which the entity finds itself. This rule is valid for a typical series of waypoints that are not in a straight line. It is defined as:

if distance to next waypoint is ≤ 25 m,
context = turn
else, context = straight.

Each neural network used a sigmoid activation function at the hidden nodes and a linear activation function at the output nodes. The inputs were normalized according to equations 2-19 below. Distances are in meters, speeds are in m/sec, and directions in degrees from north. Fundamentally, the inputs for each of the networks were a function of the M1A2 entity's state at the last simulation clock and how this state related to the road characteristics and Road March parameters such as speed.

$$S_t = S_{t-1} + \Delta S_t \quad (2)$$

where $\Delta S_t = f(Ra_{t-1}, Rb_{t-1}, Rc_{t-1}, Rp_{t-1},$
 $Rs_{t-1}, HRab_{t-1}, HRbc_{t-1}, Hz_{t-1})$

$$\theta_t = \theta_{t-1} + \Delta \theta_t \quad (3)$$

where $\Delta \theta_t = f(Ra_{t-1}, Rb_{t-1}, Rc_{t-1}, Rp_{t-1},$
 $Rs_{t-1}, HRab_{t-1}, HRbc_{t-1})$

Where

$$Ra_t = S_t / (Da_t + M) \quad (4)$$

$$Rb_t = S_t / (Db_t + M) \quad (5)$$

$$Rc_t = S_t / (Dc_t + M) \quad (6)$$

$$Rp_t = S_t P_t / M \quad (7)$$

$$Rs_t = S_t / M \quad (8)$$

$$HRab_t = Hab_r \times Hxy_t \quad (9)$$

$$HRbc_t = Hbc_r \times Hxy_t \quad (10)$$

$$S_t = \text{entity speed at } t \quad (11)$$

$$Da_t = \text{distance to previous waypoint} \quad (12)$$

$$Db_t = \text{distance to current waypoint} \quad (13)$$

$$Dc_t = \text{distance to next waypoint} \quad (14)$$

$$M = \text{march order speed} \quad (15)$$

$$P_t = \text{perpendicular distance to road} \quad (16)$$

$$Hab_t = \text{direction of road segment } ab \quad (17)$$

$$Hbc_t = \text{direction of road segment } bc \quad (18)$$

$$Hxy_t = \text{entity orientation} \quad (19)$$

The architecture for each of the four networks is described in Table 2.

Table 2 – Neural Network Architecture for Phase I Networks

NN	Arch.	Predictors	Resp.
SS	8-20-5-1	$Ra_{t-1}, Rb_{t-1}, Rc_{t-1}, Rp_{t-1},$ $Rs_{t-1}, HRab_{t-1}, HRbc_{t-1}, Hz_{t-1}$	ΔS_t
SH	7-20-5-1	$Ra_{t-1}, Rb_{t-1}, Rc_{t-1}, Rp_{t-1},$ $Rs_{t-1}, HRab_{t-1}, HRbc_{t-1}$	$\Delta \theta_t$
TS	8-20-5-1	$Ra_{t-1}, Rb_{t-1}, Rc_{t-1}, Rp_{t-1},$ $Rs_{t-1}, HRab_{t-1}, HRbc_{t-1}, Hz_{t-1}$	ΔS_t
TH	7-20-5-1	$Ra_{t-1}, Rb_{t-1}, Rc_{t-1}, Rp_{t-1},$ $Rs_{t-1}, HRab_{t-1}, HRbc_{t-1}$	$\Delta \theta_t$

A diagram of the variables is shown in Figure 5 below.

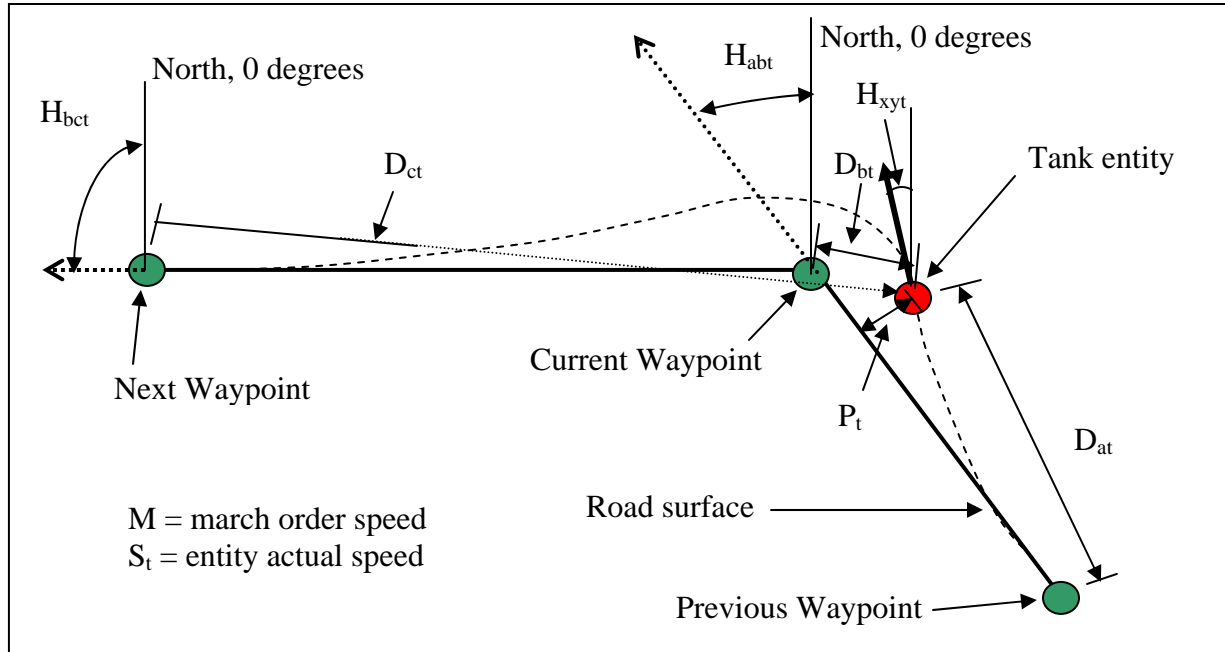


Figure 5 – Neural Network Input Variable (Predictor) Description

The same 13,760 samples gathered for the benchmark experiment were used to train and validate the four neural networks. Of these, we used 568 examples for training the straight-

heading (SH) neural network; 292 examples for training the turn-heading (TH) neural network; 100 examples for training the turn-speed (TS) neural network; and 760 examples for training the straight-speed (SS) neural network. An approximately equal number of examples were used for validating the training results of each neural network. We used the back-propagation gradient descent procedure in its training algorithm with momentum to train the various neural network models. The training rate η was selected as 0.01 and the initial momentum parameter α was set to 0.9. The momentum parameter was periodically adjusted to speed the rate of descent along the error surface.

The trained neural networks were then applied as ghost models to another simulation run of an entity traveling the same route segment to predict the movement of the simulated M1A2 entity in ModSAF. Its predictions were used to evaluate the need to issue ESPDUs to provide the correct state of the vehicle. The resulting ESPDU count was then compared to the benchmark DIS predictive results described above. The neural network predictive results are shown in Table 3.

Table 3 – Results of the NN predictive models in comparison to the DIS predictive equations

	Loc Deviations	Rot Deviations	Total ESPDUs	Total Deviations
DIS (from table 1)	205	154	351	359
Neural Networks	107	151	250	258

As evidenced in Table 3, approximately 28% fewer ESPDUs were required overall when a neural network system was employed compared to the DIS dead reckoning predictive system. Given that ESPDUs account for the majority of the network traffic in a distributed simulation exercise, our neural network predictive model results in a non-trivial reduction of bandwidth requirements.

It is interesting to note that our first attempt to predict location and rotation through neural networks (not described here) was a monolithic approach, consisting of two neural networks – one to predict location and the other rotation at all points along the route. The results for this experiment underperformed that of DIS - an ESPDU total of 504 for our neural networks compared to 359 for DIS dead reckoning. This approach was abandoned in favor of the four context-sensitive networks (SS, SH, TS, TH) described above.

3.3 Phase III – Applying further contextual sensitivity

This phase of the experimentation investigated the effectiveness of further decomposing the neural network predictive system into several neural networks trained and applied to a specific class of behavior. We compare the results to those obtained using the baseline set of neural networks described in Section 3.2. It should be noted that the baseline neural networks of Phase II do already employ a measure of contextual decomposition by virtue of having one set (of two neural networks) to predict when the entity is in a straight road segment, and another when it is in a turn. However, we take this approach one step further in this phase.

The use of a contextual approach to a modeling task has been done before ([17] among many others). It can be beneficial in a variety of ways, such as for improving performance. In other words, better performance can be achieved when it is decomposed into a number of

context-aware neural network modules. Once the task is decomposed, it is possible to apply the most appropriate network, depending on the current circumstances or context. Such switching has been discussed in the control literature [18, 19] as well as the literature on behavior-based robotics [20].

In addition to performance improvement, other motivations for adopting a contextual approach to a problem include a reduction in model complexity and construction of the overall system such that it is easier to understand, modify, and extend. Thus, the *divide-and-conquer* principle is used to reduce the complexity of a monolithic system. This enables the use of different neural network architectures to be applied to individual sub-problems, making it possible to exploit specialist capabilities. Sidani [17] used this successfully to represent human automobile driving behavior. This justification has been widely noted [21, 22] and is common to engineering design in general. Another motivation for adopting a contextual approach is the reduction of neural network training times [23]. Finally, in well-defined domains, the use of a priori knowledge can be used to suggest an appropriate decomposition of a task. This approach complements the knowledge acquisition efforts, [24] as it is easier to acquire knowledge for separate knowledge modules, rather than for the entire knowledge base at once [25].

The decomposition of a problem into context-sensitive components may be accomplished automatically or explicitly. Explicit decomposition of a task into contexts implies a strong understanding of the problem. Improved learning and performance can result if this division into sub-tasks is known prior to training [26]. An alternative approach is one in which the task is automatically decomposed according to the blind application of a data partitioning technique. Automatic decomposition is typically applied with the intent to improve performance, whereas explicit decomposition could have the aim of either improving performance or accomplishing tasks that might not be accomplished as easily or as naturally with the baseline network. Automated decomposition requires knowledge about the domain in order to do it effectively. Otherwise, a partition that leads to inefficient learning can result. Explicit partition (by a human) can result in effective execution, but human effort is required.

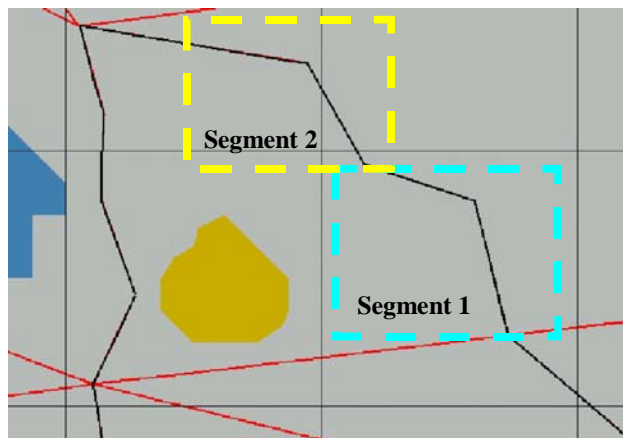


Figure 6 - Training and Testing Segments for Phase II

Phase III of our experiments used a short section of the same route used in the Phase I and II efforts. This was done to more carefully focus on the fine-grained details of the neural

networks' performance. Two turn sections found on the same extended route segment of Phase I were selected for this experiment - one immediately following the other. They are called *Segment 1* and *Segment 2*. See the graphical representation of these routes in Figure 6. Eleven scenarios using identical user-supplied parameters were executed for the purposes of training and testing the neural networks. In each of these scenarios, a single ModSAF M1A2 entity was placed at position X=22579 and Y=24328 of Segment 1, with an initial heading of 359°. Each entity was tasked with performing a Road March – a known behavior in ModSAF. The entity in each of the 11 scenarios exhibited dissimilar behavior because each was created independently in a non-repeatable version of ModSAF. This variability is attributed to the varying influence of the operating system service interrupts on ModSAF's simulation scheduling queue.

A preliminary analysis of the graphs of the 11 runs showed that they could be effectively categorized into how the ModSAF entity approached the upcoming turn [27]. The *Approach Category*, therefore, describes how far away from the “knee” of the curve (i.e., the intersection of the straight road segments, also called the *turn waypoint*) the entity begins to veer outside the road as it approaches the curve to make the turn. Four categories were identified: *Normal*, *Early*, *Late* and *Double*. The classification of the data according to the approach type is shown in Table 4. Figures 7, 8 and 9 respectively depict a *normal*, an *early* and a *late* approach category. The straight lines that meet at the turn waypoint (the “knee” of the curve) indicate the road surface. The more free-form line indicates the path taken by the ModSAF entity. This path was recorded and used in training the neural networks of Phase III.

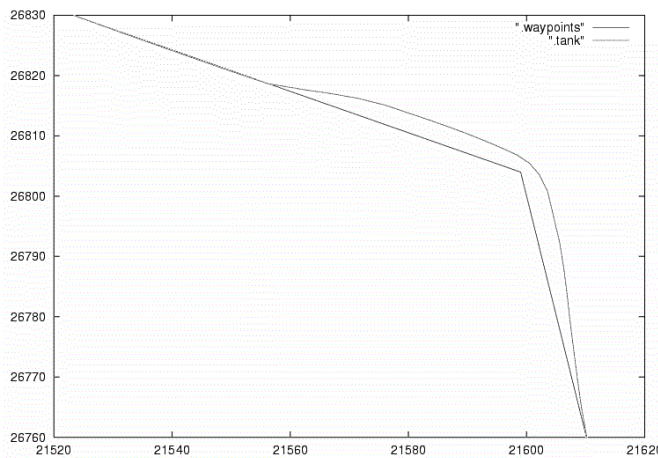


Figure 7 – ModSAF run depicting a *Normal* approach category (Scenario #1)

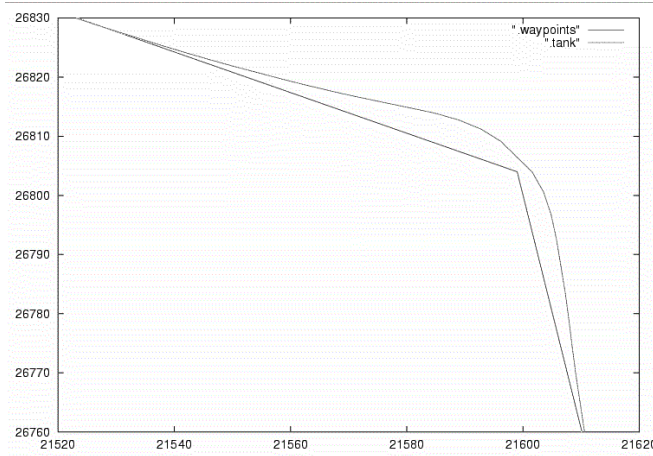


Figure 8 – ModSAF run depicting an *Early* approach category (Scenario #4)

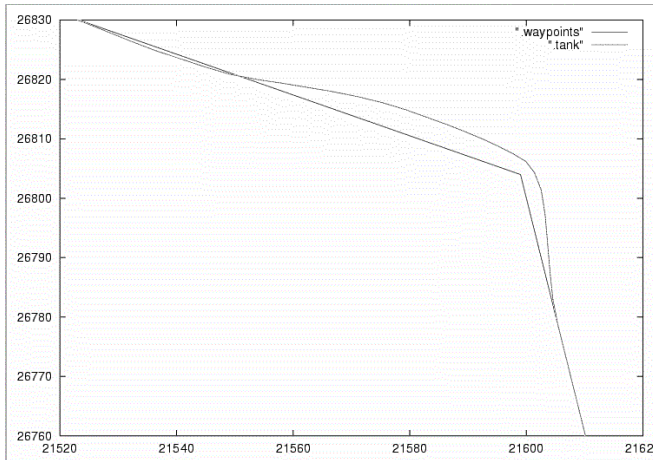


Figure 9 – ModSAF run depicting a *Late* approach category (Scenario #5)

Specifically, these categories are defined by the following rules:

- Early: ≥ 45 m to reach turn waypoint
- Normal: < 45 and > 30 m to reach turn waypoint
- Late: ≤ 30 m to reach turn waypoint

This categorization can be expressed as a rule-set that represents an explicit decomposition of the problem. The application of these rules resulted in predictive models for three of the categories (the *double* category was omitted for this experiment because there was only one instance of it). The predictive model for each category consisted of four neural networks, respectively representing the change in the M1A2 entity's speed and the change in the M1A2 entity's heading for straight lines and for turns. This further contextualizes the predictive models.

Table 4 - Classification of Data According to Approach Type

Scenario Number	Approach Category
0	Normal
1	Normal
2	Early
3	Late
4	Early
5	Late
6	Normal
7	Early
8	Double
9	Normal
10	Late

The neural networks used in this phase of the investigation were also of the feed-forward recurrent type with back-propagation gradient descent procedure in the training algorithm. The inputs were normalized according to equations 2-19 above. The architecture has two hidden layers as indicated in Table 5.

Table 5 – Neural Network Architectures for Phase II experiments.

NN	Arch.	Predictors	Response
Speed	7-20-5-1	$Ra_{t-1}, Rb_{t-1}, Rc_{t-1}, Rp_{t-1},$ $Rs_{t-1}, HRab_{t-1}, HRbc_{t-1}$	ΔS_t
Heading	7-20-5-1	$Ra_{t-1}, Rb_{t-1}, Rc_{t-1}, Rp_{t-1},$ $Rs_{t-1}, HRab_{t-1}, HRbc_{t-1}$	$\Delta \theta_t$

Specifically, a set of neural networks was trained only with the data from the training scenarios classified as being of the same type. However, not all scenarios classified similarly were used in the training of the neural networks. This resulted in three predictive systems (normal, early and late), each consisting of four neural networks (rotation in straight segments, rotation in turns, location in straight segments and location in the turns). In testing, however, each of the three predictive systems was applied to all 11 scenarios. Their results were compared to the benchmark DIS dead reckoning (DR) shown in Table 1. As in Phases I and II, the metric of comparison was the number of total deviations that would require an ESPDU to be issued by ModSAF to maintain coherence among its distributed entities. The best result is shown in (blue) boldface in Table 6 below.

Table 6 - Combined Segment 1 ESPDU Counts Using Networks Trained According to Approach Type Classification Scheme

Predictive Model (Neural network and DR)	Scenario by Classification of Segment 1 Approach										
	0 (N)	1 (N)	2 (E)	3 (L)	4 (E)	5 (L)	6 (N)	7 (E)	8 (D)	9 (N)	10 (L)
N – trained on Scenarios 0 & 1	24	24	45	47	38	41	33	42	56	41	42
E – trained on Scenarios 2 & 4	44	42	23	39	19	39	38	27	54	58	40
L – trained on Scenarios 3 & 5	52	45	49	18	36	16	46	42	44	68	17
DIS Dead Reckoning	40	42	41	31	33	35	42	38	41	45	34

where classification scheme N – normal, E – early, L – late, D – double, DR – Dead reckoning

Each row contains the results in number of ESPDUs generated by each of the three neural network predictive models and the DIS dead reckoning model when applied to each of the 11 scenarios.

The first “context-based” neural network predictive model (N) was trained with data from scenarios #0 and #1 only. Note that both scenarios corresponded to a normal (N) approach type. Likewise, the second model (E) was trained only with data from scenarios #2 and #4, which corresponded to an early (E) approach. The third predictive model (L) was trained with data from scenarios #3 and #5 – a late approach. The fourth row describes the results of the benchmark dead reckoning predictive model run on this segment, which of course, required no training. The Double approach was not implemented, as there was only one of these. The columns labeled 0 through 10 show the 11 ESPDUs test results for the four predictive neural nets employed on each scenario. The bold-faced (blue) number indicates the best of the column for each test scenario.

Improvements in performance over dead reckoning were achieved on all scenarios except for #8 - the Double approach, which was not implemented. From these results, it is apparent that the neural networks trained on one type of approach perform consistently better in tests on scenarios of the same category, whether or not that scenario was used in training or not. Prediction using this approach yielded an average ESPDU reduction of slightly over 44% over those scenarios involved in training (scenario 0 through scenario 5) and 27% over the same category scenarios not used in training (scenario 6 through scenario 10). Thus, it appears that the use of a context-based approach integrated with neural networks is of significant benefit to this problem. The relatively poor performance in scenario #8 can be explained by the fact that no network was trained on a double approach. Nevertheless, the Late neural network model (trained on 3 & 5) came close to matching the dead reckoning model. Removing scenario #8, the improvement in performance for scenarios #6, 7 and 9 is 33%.

Table 7 - Combined Segment 2 ESPDU Counts Using Networks Trained According to Approach Type Classification Scheme

Predictive Model	Scenario by Classification of Segment 2 Approach										
	0 (N)	1 (N)	2 (E)	3 (L)	4 (E)	5 (L)	6 (N)	7 (E)	8 (D)	9 (N)	10 (L)
N – trained on 0&1	91	95	66	101	110	102	105	121	105	108	101
E – trained on 2&4	53	53	47	50	70	50	46	56	45	46	49
L – trained on 3&5	42	43	36	43	72	44	45	51	48	42	44
DIS Dead	42	43	47	32	58	30	43	43	40	38	30

where classification scheme N – nominal, E – early, L – late, D - double

3.4 Phase IV – Extending to another segment: evaluating the generality of the networks

To determine the generality of the same context-based neural network predictive models vis-à-vis other road segments, the models trained on Segment 1 were evaluated over Segment 2 without further training. These results are shown in Table 7. These models did not consistently reduce PDU counts across all of the scenarios over Segment 2. Nevertheless, a pattern of lower PDU counts in scenarios whose approach classification correlates with the network classification is apparent. For example, the scenarios in Segment 2 classified as “L” yield consistently lower PDU counts with the network trained by Segment 1 Scenarios 3 and 5 (also both classified as *late*). So, while the total PDU count for Segment 2 is not reduced through this modeling scheme, there does appear to be a correlation between the type of approach represented by the trained model and the approach classification of the tested segment. Nevertheless, it is clear that this approach does not generalize well to other segments on which it was not trained.

3.5 Phase V – Evaluating the effectiveness of context-awareness

As a means of determining the relative effectiveness of contextual decomposition in our neural network predictive models, we compare these results to those obtained from the baseline neural network with the same parameters. For this comparison, this baseline network is that used for Phase II, which was trained with training data from the entire route. That is, the baseline model used all of the same model parameters but did not apply the classification rule to partition the training data into the three approach contexts or to control the model execution. As such, the baseline model was trained with roughly three times more data than were any of the models in the context-based approach. Thus, by comparing the two methods (i.e., a *context-based* approach and the baseline neural network approach) on the same curve segment, the utility of the context-based approach was evaluated. The results of this experiment may be seen in Tables 8 and 9, representing the evaluation of the model over Segment 1 and Segment 2, respectively.

Table 8 - Combined Segment 1 Results of Baseline Networks Trained with Scenarios 0-5 Compared with the Results of the Best Module.

	Scenario by Classification of Segment 1 Approach										
Predictive Model (Neural network and DIS)	0 (N)	1 (N)	2 (E)	3 (L)	4 (E)	5 (L)	6 (N)	7 (E)	8 (D)	9 (N)	10 (L)
Baseline NN model	32	34	34	32	25	29	36	29	48	55	31
Best context-based NN	24	24	23	18	19	16	33	27	44	41	17
DIS Dead Reckoning	40	42	41	31	33	35	42	38	41	45	34

where classification scheme N – nominal, E – early, L – late, D - double

The results of this experiment vis-à-vis Segment #1 shown in Table 8 indicate a reduction in ESPDUs for all scenarios. These improvements over the baseline network range from only 7% for scenario #7 to the best of over 45% (scenario #10). It is interesting to note that except for scenarios #3, #8 and #9, the baseline NN predictive model also indicates improvement over the DIS dead reckoning predictive model. These improvements of the baseline NN over DIS (not counting scenarios 3, 8 and 9) average 20%.

The results vis-à-vis segment 2 are consistent with what we saw on segment 1 in that the context-based neural networks do improve the baseline networks. Table 9 depicts these results. Improvements range from a low of 1.5% (scenario #4) to a high of 21% in scenario #3. However, it is evident that neither the context-based predictive neural networks nor the baseline networks outperform the DIS dead reckoning predictive models for Segment 2.

Table 9 - Combined Segment 2 Results of Monolithic Networks Trained with Scenarios 0-5

	Scenario by Classification of Segment 2 Approach										
Predictive Model (Neural network and DIS DR)	0 (N)	1 (N)	2 (E)	3 (L)	4 (E)	5 (L)	6 (N)	7 (E)	8 (D)	9 (N)	10 (L)
Baseline NN model	48	47	41	52	71	52	48	55	48	48	52
Best context-based NN	42	43	36	43	70	44	45	51	45	42	44
DIS Dead Reckoning	42	43	47	32	58	30	43	43	40	38	30

where classification scheme N – nominal, E – early, L – late, D - double

From these experiments, we can conclude that the decomposition of the neural networks into context-sensitive application of neural networks depending on some simple rules can be effective, often significantly. We can further conclude that the results do not generalize well to entity performance on other segments of the road on which the networks have not been trained.

3.6 Phase VI – Sensitivity to error thresholds and computational efficiency issues

To further examine the relationship between the predictive power of the dead reckoning models and this set of predictive neural network models, we conducted experiments over a range of error tolerances. Whereas the initial results reported earlier in Table 3 were measured according to DIS default values for a tank entity (i.e., 0.356 m, 0.734 m, and 3°), follow-on tests doubled

these error thresholds. Results are presented in Table 10 and reported only by total number of required updates.

As evidenced in Table 10, as the error tolerance increases, the predictive advantage that neural networks have over dead-reckoning models becomes less significant for this modeling task. Furthermore, in the error tolerance range where the neural network model does predict the entity's path with more accuracy, the improvement comes at a cost in processing time. This is shown in Table 11, which was derived by using the UNIX gettimeofday function. The processing speed was calculated on a Pentium III, 500 MHz machine, running RedHat Linux 6.2. The neural network based model required, on average, about a factor of 6 more processing time than did the dead-reckoning predictive model. Since the overall simulation time was approximately 15 minutes, the dead reckoning predictive model then, on average, produced about 23 updates per minute or rather, 1 update every 2.5 seconds (at a threshold of 0.356 m in the X direction and 0.734 m in the Y direction). Alternatively, the neural network predictive model required about 17 updates per minute, or approximately 1 update every 3.5 seconds at the same thresholds. Coupling this information with the information on processing time tradeoffs, it becomes clear that for applications where processing time is at a premium, the use of dead reckoning-models may be preferred, in spite of their poorer predictive performance. However, in certain applications where processing time is not the primary constraint e.g., entities communicating over a wireless network, then the increased processing costs incurred from using a neural network can offer a superior option.

Table 10 - Updates Required Over Increasing Error Thresholds

Factor of	Error Threshold			Updates Required	
	X-axis (m)	Y-axis (m)	All-axes (deg)	NN	DR
1	.356	.734	3	258	359
2	.712	1.468	6	193	237
3	1.068	2.202	9	157	188
4	1.424	2.936	12	138	156
5	1.78	3.67	15	119	137
7	2.492	5.138	21	98	109
9	3.204	6.606	27	88	92
11	3.916	8.074	33	70	79
13	4.628	9.542	39	69	75
15	5.34	11.01	45	62	73
20	7.12	14.68	60	51	60
25	8.9	18.35	75	48	55
30	10.68	22.02	90	44	48

Table 11 - Execution Time Using Neural Networks and Dead-Reckoning Equations

	Processing Time (in 10^{-5} seconds)
--	---

	NN Speed	NN Heading	Total NN	Total DR
Min	6.8902 9	6.1988 8	13.08981	2.2888
Mean	7.7769 2	7.4700 8	15.24700	2.7974
Max	20.599 3	29.799 9	50.3992	6.3598 1

3.7 Phase VII – Repeatability of human performance

In this phase of the investigation, we sought to generate experimental evidence that human performance is repeatable in many aspects. This is important if the entities whose movement is to be predicted are human controlled (either live or manned simulator). To accomplish this, we employed two human subjects in our research, code named *Bunker6* and *Seagram7*. Both test subjects are experienced, albeit retired, military officers (COL and LTC respectively). Bunker6’s experience was specifically in armor operations. He was experienced in driving and commanding battle tanks such as the M1 Abrams. Seagram7, on the other hand, although equally experienced in terms of seniority, was an artillery specialist, with some limited tank operation experience.

Each test subject was put through the same segment 1 turn on a tabletop simulator that closely resembled an M1A2 tank driver station. The simulator was interfaced with ModSAF 5.0, the same version used in the prior experimentation. In this experiment, the human subjects were tasked with driving the tank simulator on a road with similar aspects to the one in the National Training Center used in Phases I, through VI. Figure 10 depicts the traces of the tracks made by Bunker6 in three separate simulator runs. Figure 11 shows the variability in speed for the same runs. The dark solid line represents the centerline of the road surface.

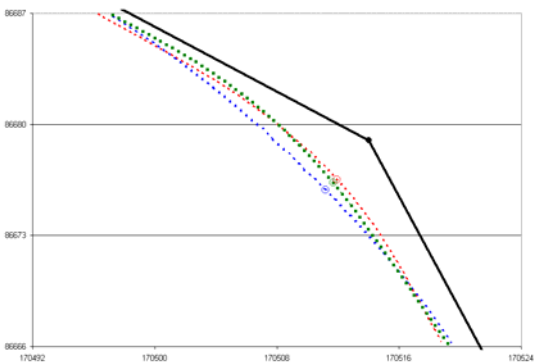


Figure 10 – Position traces for Bunker6

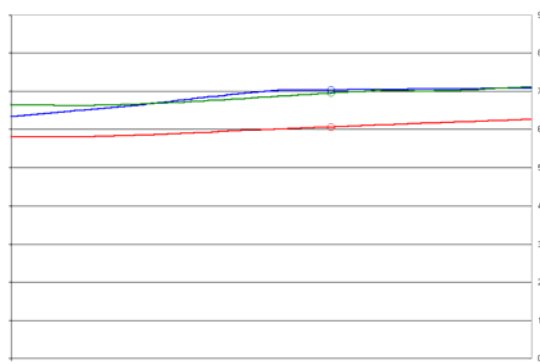


Figure 11 – Speed Traces for Bunker6

Note the consistency in Bunker6’s performance. This can be contrasted with the lower repeatability of a ModSAF entity during a Road March behavior indicated in Figures 12 and 13, which display the corresponding traces of speed and position for a ModSAF entity. It is clear

that human performance, at least as performed by Bunker6, is significantly more consistent and repeatable than that of the ModSAF entity doing a road march. The performance of the ModSAF entity represents the worse case, thus justifying our use of the ModSAF entity as the basis of our work.

We further evaluated the consistency in performance across human subjects. In other words, given that individual human drivers perform consistently when compared to themselves over several repetitions of the same task, how similar are performances of different humans executing the same task? To answer this question, we developed a neural network predictive model and trained it with Bunker6’s performance data. We then proceeded to test it against Bunker6’s data used in the training, as well as data not used in training. Most significantly, we also applied this model to predict Seagram7’s performance.

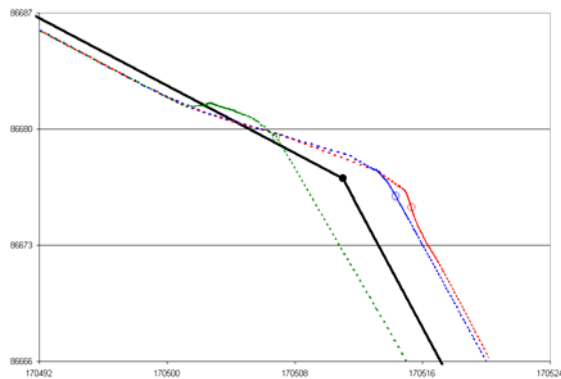


Figure 12 - Position traces for ModSAF entity

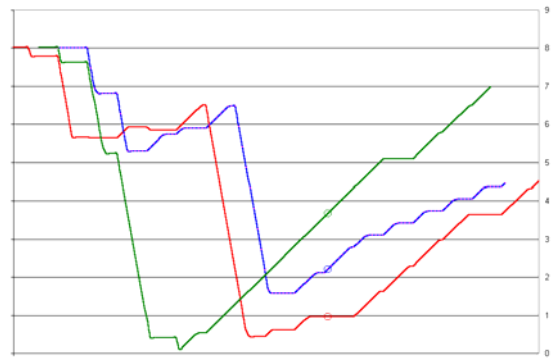


Figure 13 - Speed Traces for ModSAF entity

The neural network model selected for this experiment consisted of two networks – one to predict position change and the other to predict rotational change. Each employs five inputs and one output, plus five hidden nodes in one hidden layer. Table 12 displays the architecture of the feed-forward recurrent neural networks. They were trained using the back-propagation gradient descent procedure in the training algorithm.

Table 12 - Architecture by Neural Network Type

NN	Architecture	Predictors	Response
Speed	5-5-1	$Rwp_{t-1}, Rcl_{t-1}, Rs_{t-1}, HRab_{t-1}, HRbc_{t-1}$	ΔS_t
Heading	5-5-1	$Rwp_{t-1}, Rcl_{t-1}, Rs_{t-1}, HRab_{t-1}, HRbc_{t-1}$	$\Delta \theta_t$

The results of this Phase VII experiment are depicted in table 13.

Table 13 - ESPDUs Generated in Human Controlled Vehicle State Synchronization Experiments

Source Data	Newtonian-based dead-reckoning	Neural Network predictive model based
Bunker6 Training Data	3	0
Bunker6 Testing Data	3	1
Seagram7 Testing Data	3	3

The results of Table 13 suggest that the neural network captures the behavior of Bunker6 very well, but that predictive capability does not translate well to Seagram7. This suggests that human performance, while repeatable for specific human subjects, need not carry over well to other humans. One possible explanation specific to our test subjects is that the relative inexperience of Seagram7 resulted in inconsistent performance. Bunker6’s greater experience and skill in tank operation resulted in a more exact and consistent performance. However, the tests are not sufficiently extensive to warrant a general conclusion to that effect, and further testing is warranted.

4. Summary and Conclusions

In summary, a context-based neural-network modeling scheme to improve the predictive ability of DIS dead reckoning schemes was proposed and formulated. It was empirically developed and tested in a simulated environment. As part of this effort, the use of explicit model decomposition schemes was considered as a mechanism for improved predictive performance. The performance of the best modeling combination was evaluated in several ways. The results indicated that neural networks can indeed improve the predictive ability of ModSAF entities, thereby resulting in decreased network PDU traffic. Secondly, we determined that contextualization of the neural network predictive models can further improve the predictive ability of the neural networks.

Nevertheless, the ability of the neural network predictive models to generalize from entity to entity, road section to road section or even from person to person (albeit in limited testing) was less encouraging. As a practical matter, the advantages presented by this approach to predicting near term movements could be best realized by performing a rehearsal of the exercise and building the models through observation of prior performance in these rehearsals. In many cases, such rehearsals can be quite realistic, while not so in other cases. Results also indicated that the best benefit when cost is included in the decision comes from training the networks over the entire route, rather than in a fine-grained fashion as done in Phases II and III. Lastly, the computational cost of using neural networks was steep. However, as hardware speed and cost decrease over time, this will become less of an issue.

Second order equations of motion are likely to improve the efficiency and effectiveness of the traditional dead reckoning equations. However, they cannot predict the human’s actions. The neural networks can do this by learning from the human’s prior actions. Thus, the objective of the study reported herein was not to find the best solution to the problem but rather, to explore the feasibility of using neural networks with their ability to learn from historical data as an alternative to the dead reckoning process currently used in DIS. Future work in this aspect of the study includes investigating methods of automating the learning of the task decomposition and hence, the context-shifting rules. Also, the improvement of the neural networks’ performance

should be explored further. This includes considering alternative types of architectures, inputs, normalization schemes, and sampling strategies. For details of this work, see [28].

5. Acknowledgements

The authors acknowledge the extensive and essential contributions of Dr. Amy Henninger of IDA while she was a student in our laboratory as well as thereafter. Also acknowledged are the contributions of Dr. William Gerber and of Mr. Judd Tracy. This work was sponsored by the U.S. Army Simulation, Training, and Instrumentation Command (STRICOM, now called PEO STRI) as part of the Inter-Vehicle Embedded Simulation and Technology (INVEST) Science and Technology Objective (STO) contract N61339-98-K-0001. That support is gratefully acknowledged.

6. References

- [1] Smith, S. and Petty, M., "Controlling autonomous behavior in real-time simulation", In *Proceedings of the Second Conference in Computer Generated Forces and Behavior Representation*, Orlando FL, 1992.
- [2] Vargas, J.J., DeMara, R.F., Gonzalez, A.J. and Georgiopoulos, M., "Bandwidth and latency implications of integrated training and tactical communication networks," In *Proceedings of SAWMAS-04*, Cocoa Beach, FL, 2004.
- [3] Cebula D.P. and DiCaprio, P.N., "Tradeoffs involved with separating aggregated data packets into attribute cluster packets." In *Proceedings of the 14th Workshop on Standards for the Interoperability of Distributed Simulations*, Orlando FL, 1996.
- [4] Vargas, J.J., DeMara, R.F., Georgiopoulos, M., Gonzalez, A.J. and Marshall, H., "PDU bundling and replication for reduction of distributed simulation communication traffic," *Journal of Defense Modeling and Simulation*, 1(3), (2004), 173–185.
- [5] Liang, L.A.H., Cai, W., Lee, B-S. and Turner, S.J., "Performance analysis of packet bundling techniques in DIS." In *Proceedings of the 3rd IEEE International Workshop on distributed interactive simulation and real-time applications*, pp. 75–82, Greenbelt MD, 1999.
- [6] Stone, S., Zyda, M., Brutzman, D. and Falby, J., "Mobile agents and smart networks for distributed simulation." In *Proceedings of the 14th Workshop on Standards for the Interoperability of Distributed Simulations*, Orlando, FL, March 1996.
- [7] Wuerfel, R.D. and Johnston, R., "Real-time performance of RTI version 1.3." In *Proceedings of the 1998 Fall Simulation Interoperability Workshop*, September, 1998.
- [8] Calvin, J.O., Seeger, J., Troxel G.D. and Van Hook D.J., "STOW realtime information transfer and networking system architecture." In *Proceedings of the 12th Workshop on Standards for the Interoperability of Distributed Simulations*, Orlando FL, 1995.
- [9] Ceranowicz, A., Torpey, M., Helfinstine, W., Evans J. and Hines, J., "Reflections on building the joint experimental federation." In *Proceedings of the 2002 IITSEC*, Orlando FL, 2002.
- [10] Bassiouni, M., Chiu, M., Loper, M., Garnsey, M. and Williams, J., "Performance and reliability analysis of relevance filtering for scalable distributed interactive simulation." In

- ACM Transactions on Modeling and Computer Simulation (TOMACS)*, volume 7, (997), pp. 293–331.
- [11] Purdy S.G. and Wuerfel, R.D., “A comparison of HLA and DIS real-time performance.” In *Proceedings of 1998 Spring Simulation Interoperability Workshop*, Orlando FL, March 1998.
- [12] Wang B. and Hou, J., “Multicast routing and its QoS extension: problems, algorithms, and protocols.” *IEEE Network*, 14, (2000).
- [13] Wills, C., Mikhailov, M. and Shang, H., “N for the price of 1: Bundling Web objects for more efficient content delivery,” In *Proceedings of the Tenth International Conference on World Wide Web*, Hong Kong, 2001, pp. 257-265.
- [14] Taylor, D., “DIS-Lite & Query Protocol” In *Proceedings of the 13th DIS Workshop on Standards for the Interoperability of Distributed Simulations*, 1995.
- [15] Lin, K. and Ng, H., “Coordinate transformations in distributed interactive simulation (DIS)”, *Simulation*, vol. 61, No. 5, (1993), pp. 326-331.
- [16] Calder, R. B., Smith, J. E., Courtemanche, A. J., Mar, J. M. and Ceranowicz, A., “ModSAF behavior, simulation and control”, In *Proceedings of the 3rd Conference on Computer Generated Forces and Behavioral Representation*, pp. 347-356, Orlando FL, 1993.
- [17] Sidani T. A. and Gonzalez, A. J., “A framework for learning implicit expert knowledge through observation”, *Transactions of the Society for Computer Simulation*, Vol. 17, No. 2, (2000), 54-72
- [18] Murray-Smith R. and Johansen, T. A., Multiple Model Approaches to Modeling and Control, Taylor and Francis: UK, 1997.
- [19] Narendra, K. S., Balakrishnan, J. and Ciliz, K., “Adaptation and learning using multiple models, switching and tuning”, *IEEE Control Systems Magazine*, (1995), 37-51.
- [20] Brooks, R. A., “A robust layered control system for a mobile robot”, *IEEE Journal of Robotics and Automation*, RA-2, (1986), pp. 14-23.
- [21] Gallinari, P., “Modular neural net systems: training of”, In M.A., Arbib (editor), The Handbook of Brain Theory and Neural Networks, pp. 582-585, Bradford Books/MIT Press: Boston, MA, 1995.
- [22] Hrycej, T., Modular Learning in Neural Networks, John Wiley: Chichester, 1992.
- [23] Pratt, L. Y., Mostow J and Kamm, C. A., “Direct transfer of learned information among neural networks,” In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAI-91)*, pp. 584-589, Anaheim, CA, 1991.
- [24] Ourston, D., Blanchard, D., Chandler E. and Loh, E., “From CIS to software,” In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavior Representation*. Orlando, FL., May 1995, pp. 275-285.
- [25] Henninger A. E. and Gonzalez, A.J., “Automated acquisition tool for tactical knowledge,” In *Proceedings of the 10th Annual International Florida Artificial Intelligence Research Symposium*, pp. 307-311, Daytona Beach FL., 1997.

- [26] Hampshire J. B. and Waibel, A. H., “The Meta-PI Network: building distributed representations for robust multisource pattern recognition”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(7), (1992), 751-769.
- [27] Gonzalez, A. J., Gerber, W. J., DeMara, R. F. and Georgiopoulos, M., “Context-driven Near-term Intention Recognition”, *Journal of Defense Modeling and Simulation*, Volume 1, Number 3, pp. 153-170, August 2004
- [28] Henninger, A. E., “Neural Network Based Movement Models to Improve the Predictive Utility of Entity State Synchronization Methods for Distributed Simulations”, Doctoral Dissertation, Electrical and Computer Engineering Department, University of Central Florida, Fall, 2000.

Author Biographies

Avelino J. Gonzalez is a professor of Computer Engineering in the School of Electrical Engineering and Computer Science at the University of Central Florida, Orlando FL, USA. His area of expertise is intelligent systems and machine learning. He received his PhD in Electrical Engineering in 1979 from the University of Pittsburgh.

Michael Georgiopoulos is a professor of Electrical Engineering in the School of Electrical Engineering and Computer Science at the University of Central Florida, Orlando FL, USA. His area of expertise is neural networks. He received his PhD in Electrical Engineering from the University of Connecticut.

Ronald F. DeMara is a professor of Computer Engineering in the School of Electrical Engineering and Computer Science at the University of Central Florida, Orlando FL, USA. His area of expertise is computer architecture and intelligent systems. He received his PhD in Computer Engineering from the University of Southern California.