Predicting Agent Spatial Information: A Comparison Between Neural Networks and Dead Reckoning Algorithms

Amy E. Henninger (amy@soartech.com)
Soar Technology, Inc. 3361 Rouse Road Suite 240
Orlando, FL 32826

Avelino J. Gonzalez (ajg@isl.engr.ucf.edu)
School of Electrical Engineering and Comupter Science
University of Central Florida, Orlando, FL 32816

Douglas A. Reece (reeced@saic.com)
SAIC 12479 Research Parkway
Orlando, Florida 32817

Abstract

In tune with the 24th Annual Cognitive Science Conference's emphasis on application, this paper presents an empirical comparison between two methods used in agent tracking. The need to predict an agent's intents or future actions has been well documented in multi-agent system's literature and has been motivated by both systematically-practical and psychologically-principled concerns. However, little effort has focused on the comparison of predictive modeling techniques. This paper compares the performance of two predictive models both developed for the same, well-defined modeling task. Specifically, this paper compares the performance of a neural network based model and dead-reckoning model, both used to predict an agent's trajectory and position. After introducing the background and motivation for the research, this paper reviews the form of the dead-reckoning algorithms, the architecture and training algorithms of the neural networks, the integration of the models into a large-scale simulation environment, and the means by which the performance measures are generated. Quantitative measures from our experiments indicate that, for the task considered, the neural network based model provides greater predictive utility, but at an increased cost in processing time. Performance measures are presented over increasing levels of error tolerance.

Introduction

Intelligent agents typically operate in an environment populated by other intelligent agents. Agents may help each other, hinder each other, get in each other's way, or ignore each other, often without directly communicating their intent. In order for an agent to achieve its goals, it is thus sometimes necessary for the agent to determine where the other agents are, what they are doing, and what their plans are. For example, an agent may want to infer what plan an opponent is executing so that the agent can select countermoves. Han and Veloso (1995), Rao (1994), Rao and Georgeff (1995), Tambe and Rosenbloom (1995), and Tambe (1996) have studied various forms of recognizing an agent's intents.

Sometimes it is necessary to infer facts that are normally observable, such as agent location, because of sensor or other limitations. For example, a pilot agent may need to predict where a threat aircraft is flying after it enters a cloud. There are many approaches to predicting agent trajectories, including Newtonian mechanics (Lin and Ng, 1993), neural networks (Kim et al, 1999), Hidden Markov Models (Washington, 1998) and others. This paper addresses a particular application of trajectory prediction in distributed simulation and compares the effectiveness of a neural network to a commonly used Newtonian approach for this application.

The remainder of this section defines the trajectory estimation problem in the distributed simulation application and describes a previous use of neural networks for estimating agent trajectory in a visual scanning application. The paper then describes a neural network approach for trajectory estimation in distributed simulation and presents results and comparisons with Newtonian dead reckoning.

Dead Reckoning in Distributed Simulation

In a Distributed Interactive Simulation (DIS) (DIS Steering Committee, 1994), simulation software for each agent runs independently of other agents and broadcasts the ground truth about the state of the agent through network packets known as protocol data units (PDUs). Each simulation in DIS uses trajectory estimation so that the state of the agents does not have to be broadcast frequently. Lin and Ng

(1993) explain how dead-reckoning can be used to maintain coherence among entities' states in a DIS environment. Each simulator uses Newtonian equations of motion such as equation 1

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

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

to predict the trajectory of other agents. Each simulator also uses the same equation to model the trajectory of its own agent; the output of this equation can be compared to the output of the true dynamics model for the agent to determine when the models diverge. When, and only when, the error between models reaches a certain threshold, the simulator broadcasts new state information for its agent. Figure 1 shows this process in a DIS simulation called ModSAF (Calder et al, 1993) that we used for our experiments.

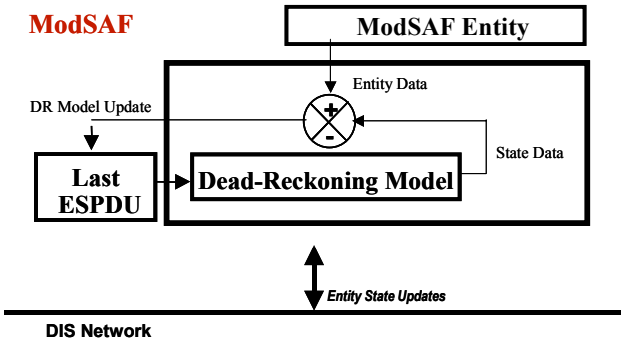

Figure 1. Dead-Reckoning Implementation in ModSAF

Figure 2 shows how at a series of time steps, the true position of an agent computed by the agent dynamics model (shown by the curve) deviates from a linear dead reckoning model. When the error exceeds the threshold, the models are brought into correspondence by the issuance of an entity state PDU (ESPDU). Thus in the figure, only 3 ESPDUs are broadcast instead of one at every time step. The goal of the research presented here is to reduce the number of ESPDUs sent in by DIS simulations below the number needed using Newtonian dead reckoning.

Figure 2. DIS Dead-Reckoning Process

Neural Networks for Trajectory Estimation

Kim et al (1999) developed a system to generate short-term predictions of an agent's trajectory such that it can be used to predict the agent's position at any future instance, given some window of time. They use this model as part of a helicopter agent's perceptual system to enhance the agent's ability to visiually track ground vehicles, and their motivation for this model is both psychologically and practically rooted. Psychologically, this model can be used to simulate a helicopter pilot's gaze shifting as he attempts to visually track and reaquire multiple targets. Thus, instead of operating in a state of omniscience, the agent is required to juggle the act of determining spatial information across multiple agents, as would be the human helicopter pilot. The functional ramification of this approach is that the total number of perceptual inputs to the agent is reduced at any given instance. In other words, instead of getting continuous perceptual information on all of the ground entities within the helicopter agent's field of view, by using this predictive model, the agent only requires updated information on entities when its attention is focused on those entities.

The high level architecture of this system is presented in Figure 3. The agent architecture is embedded in the

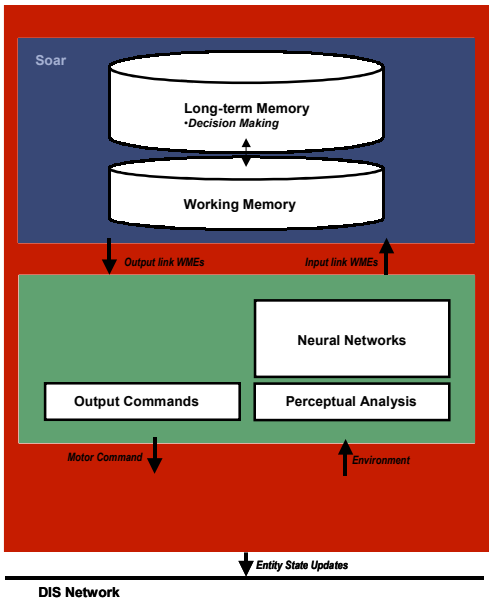

Figure 3. Visual attention for helicopter agent

Modular Semi-automated Forces (ModSAF) simulator, a system used by the military for training and research. ModSAF is elaborated on in section 2, "Methodology". The agent's intelligence is modeled in Soar (Rosenbloom et al, 1993; Newell, 1990). As a model of natural intelligence, the Soar software architecture combines the abilities to react immediately to situations, use knowledge in deliberative decision making, step back from the immediate situation to perform various forms of problem solving and planning, and learn from experience. As an indicator of the maturity and sophistication of Soar-based agents, the system has been used successfully as the production model in a number of large-scale military exercises (Hill et al, 1997; Jones et al, 1999; Nielsen et al, 2000).

The inputs to the neural networks developed for this application consist of entity data (e.g., call-sign, sim-time, position, velocity, etc.) and abstracted terrain information germane to both "on-roads" and "cross-country" travel and correlated to the entity's visual field (hill, road, lake, etc). All together, the input vector consists of 196 fields and the output vector consists of 15 output fields corresponding to discretized changes in heading ranging from –35° to 35°. The selected heading change, coupled with an assumed constant speed and "delta" time since last prediction, can be used to predict the entity's expected location at some time, **t**. With this prediction, the virtual helicopter pilot is able to look away from the ground entity for up to 7 seconds, within some error threshold.

Methodology

This paper seeks to compare the performance of a neural-network based model with the dead-reckoning model. Like both systems described in sections on dead-reckoning and neural networks for trajectory estimation, this experiment is implemented in ModSAF, a training and research system developed by the Army's Simulation, Training, and Instrumentation Command (STRICOM). ModSAF provides a set of software modules for constructing computer-generated force behaviors at the company level and below. Typically, ModSAF models are employed to represent individual soldiers or vehicles and their coordination into orderly-moving squads and platoons; but, their tactical actions as units are planned and executed by a human controller. The human behaviors represented in ModSAF include move, shoot, sense, communicate, tactics, and situation awareness. The authoritative sources of these behaviors are subject matter experts and doctrine provided by the Army Training and Doctrine Command (TRADOC). ModSAF uses state transition constructs inspired by finite state machines (FSMs) to represent the behavior and functionality of a process for a pre-defined number of states.

The scenario used for the comparison was a road-march for a tank entity 45-segment route shown in Figure 4 It is approximately 7 kilometers long and takes a tank entity about 15 minutes of simulation time to travel at a March Order speed of 8 m/s. From this 15 minute period, a total of 13760 movement updates were performed, generated at a rate of 15 HZ.

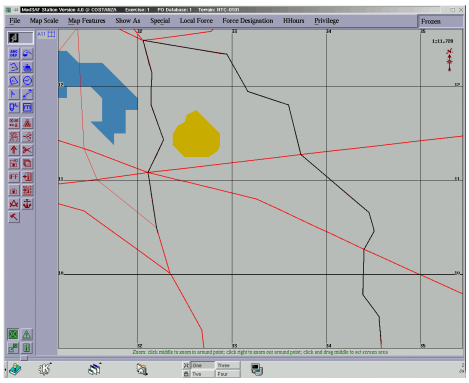

Figure 4. Route Used for Experiment

For this application, a feed-forward architecture developed with back-propagation training was used to develop the neural networks. One of these networks predicts the change in an entity's speed and the second network predicts the change in the entity's heading. Each network used a sigmoid function at the hidden nodes and a linear transformation at the output nodes. The configuration of the networks in each of the models may be seen in Table 1

Table 1. Neural Network Architecture

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

where the inputs were normalized according to equations 2 – 19 below. Fundamentally, the inputs for each of the networks were a function of the entity's state at the last simulation clock and how this state related to the road characteristics (width, heading, length of segment, etc) and March Order parameters (speed, end-point, etc). The specific predictors are expressed in 4 – 10, and the parameters making up those inputs are explained in 11 – 19.

$$S_t = S_{t-1} + \Delta S_t \qquad (2)$$
$$\text{where } \Delta S_t = f(Ra_{t-1}, Rb_{t-1}, Rc_{t-1}, Rp_{t-1}, Rs_{t-1}, HRab_{t-1}, HRbc_{t-1}, Hz_{t-1})$$

$$\theta_t = \theta_{t-1} + \Delta\theta_t \quad (3)$$
where $\Delta\theta_t = f(Ra_{t-1}, Rb_{t-1}, Rc_{t-1}, Rp_{t-1}, Rs_{t-1}, HRab_{t-1}, HRbc_{t-1})$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Of these, 860 examples were used for training the speed network, 860 examples for training the heading network, and 859 examples were used for validating the training of both of these networks. The training rate was selected as 0.01 and the initial momentum parameter was .9. The momentum parameter was periodically adjusted to speed the rate of descent along the error surface. The training and validation results for each of the networks may be seen in Table 2.

Table 2. Training and Validation Errors

	Delta Speed ΔS Error(m/s)	Delta Heading $\Delta\theta$ Error(rads)
Training	0.259977±2.04558	0.004578±0.00781
Validation	0.206374±0.82532	0.014221±0.06766

Experimental Results

The neural network models were implemented in such a way that their performance for predicting entity location could be compared with the dead-reckoning model. This implementation is presented in Figure 5.

There are two ways of generating an error in our system. The first is according to the enity's location. This error is measured in terms of comparing the entity's dead-reckoned XYZ with the entity's true XYZ and is proportioned according to the width of the entity along its X, Y, and Z axes. For example, an M1A2 tank is 3.56m in width (defined along X-axis of tank), 7.34m in length (defined along Y-axis of tank), and 2.33m in height (defined along Z-axis of tank). A typical threshold for dead-reckoning error tolerance in DIS is 10% of the vehicle's dimensions. In this case then, the error tolerance for this entity's location would translate into .356m along the X-axis, .734m along the Y-axis of the tank, and .233m along the Z-axis of the tank.

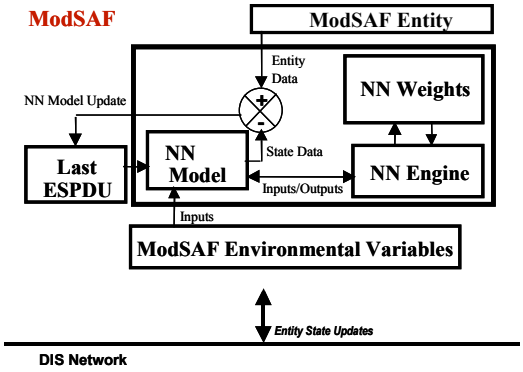

Figure 5. Neural Network Implementation used for Experiments in ModSAF

The second way of determining an update threshold is with respect to the orientation of the vehicle. In this case, the components of the dead-reckoned euler angles are compared with the components of the entity's true orientation. For tracked ground entities in ModSAF, this measure is defaulted at 3°. That is, if the dead-reckoned prediction is more than 3° off about X-axis, Y-axis, or Z-axis, an error is generated. Overall, at these error tolerances, the number of updates (ESPDUs) required by the dead-reckoning model was 351. Using these same error thresholds, the neural network models required 263 updates. Thus, the neural networks required 25% fewer updates than the dead-reckoning models. This information is presented in Figure 6 according to type of update. In the DR case, a small number of the required updates occurred simultaneously between location and rotation.

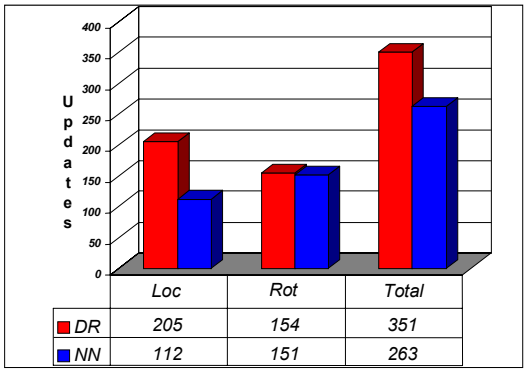

Figure 6: Number of Updates Required by DR Model versus NN Model

Although the neural network based model was able to predict the entity's path with more accuracy, as evidenced in Table 3, this increase in predictive utility comes at a cost in processing time.

Table 3. Execution Time Using Neural Networks and Dead-Reckoning Equations

	Processing Time (in 10^{-5} seconds)			
	NN Speed	NN Heading	Total NN	DR
Min	6.89029	6.19888	13.08981	2.2888
Mean	7.77692	7.47008	15.24700	2.7974
Max	20.5993	29.7999	50.3992	6.35981

Using the UNIX "gettimeofday" function, the processing speed was calculated on a Pentium III, 500 Mhz machine, running RedHat Linux 6.2. As shown in Table 3, the neural network based model required, on average, about 6 times more processing time than did the dead-reckoning based model. As stated in Section 2, the simulation time was approximately 15 minutes. On average then, the dead reckoning model produced about 23 updates per minute or rather, 1 update every 2.5 seconds (at a threshold of .356m in the X direction and .734m in the Y direction). Alternatively, the neural network based 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.

To further examine the relationship between the predictive power of the dead-reckoning models and this set of neural network models, we conducted experiments over a range of error tolerances. So, whereas the initial results, reported in Figure 6, were measured according to DIS default values for a tank entity (i.e., .356m, .734m, and 3°), follow-on tests incremented these error thresholds by those exact amounts. Results are presented in Table 4 and reported only by total number of required updates.

Table 4. 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	263	351
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

As evidenced in Table 4, as the error tolerance increases, the predictive advantage that neural networks have over dead- reckoning models becomes less significant for this modeling task.

Summary and Conclusions

As one might expect, the choice of tool must be driven by the modeling constraints. The results reported above suggest heuristics for when to apply which modeling technique. For example, in an application where processing time is not the primary constraint e.g., multi-agent systems communicating over a wireless network, then the increased processing costs incurred from using a neural network may be defendable. Alternatively, in an application where processing time is a limiting factor, then dead-reckoning models may be the more prudent approach. It is interesting to note, also, that the differences in predictive utility of the two modeling approaches becomes less prominent as the error threshold is increased. This speaks to the power of dead-reckoning models to generalize and scale.

It is important to recognize, of course, that the modeling task in this research is limited in scope. Also, a different neural network could have yielded different results. We can not claim that this is the best network architecture or configuration for this specific modeling task. We can only claim that it was one of the more promising configurations with which we experimented. Other configurations may be better. One approach Henninger et al (1999) found particularly effective in improving the neural network based model's performance was to work with modularized models. This approach has been advocated in the control literature (Murray-Smith and Johansen, 1997; Narendra et al, 1995) and robotics literature (Brooks, 1986), and we have started exploring this approach. One of the benefits of adopting this approach is the ability to mix different modeling techniques as they best apply to the problem locally. For example, combinations of architectures and/or algorithms that can be applied to individual sub-problems, make it possible to exploit specialist capabilities. In the problem discussed in this paper, one interesting test would be to use dead-reckoning algorithms in straight parts of the road data base and then use neural networks to guide the turn, as this appears to be where the majority of updates are required.

Acknowledgements

This work was sponsored by the U.S. Army Simulation, Training, and Instrumentation Command, contract N61339-98-K-0001. That support is gratefully acknowledged.

References

Brooks, R.A. 1986. A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation, RA-2, 14-23.

Calder, R.B., Smith, J. E., Courtemanche, A.J., Mar, J.M., and Ceranowicz, A. (1993). ModSAF Behavior Simulation and Control. In Proceedings of the 3rd Conference on Computer Generated Forces and Behavioral Representation (Orlando FL), 347-356.

DIS Steering Committee 1994. "The DIS Vision: A Map to the Future of Distributed Simulation", Technical Report, IST-ST-94-01. Institute for Simulation and Training, University of Central Florida.

Han, K. and Veloso, M. 1995. Automated robot behavior recognition applied to robot soccer. Sixteenth International Joint Conference on Artificial Intelligence. Workshop on Team Behaviour and Plan Recognition, 53-64.

Henninger, A., Gonzalez, A., and Georgiopoulos, M. 1999. Modeling Semi-automated forces with neural networks: Performance improvement through a modular approach. Proceedings 9th Conference on Computer Generated Forces and Behavioral Representation, (Orlando FL), 261-268.

Hill, R. W., Chen, J., Gratch, J., Rosenbloom, P., and Tambe, M. 1997. Intelligent agents for the synthetic battlefield: A company of rotary-wing aircraft. In Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence, 1006-1012. Menlo Park, CA: AAAI Press.

Jones, R. M., Laird, J. E., Nielsen, P. E., Coulter, K. J., Kenny, P., and Koss, F. V. 1999. Automated intelligent pilots for combat flight simulation. *AI Magazine,* 20(1): 27-41.

Kim,Y., Hill, R., and Gratch, J. 1999. How long can an agent look away from a target? Proceedings 9th Conference on Computer Generated Forces and Behavioral Representation, (Orlando FL), 35-38.

Lin, K., and Ng, H. 1993. Coordinate transformations in distributed interactive simulation (DIS). Simulation, vol. 61(5):326-331.

Murray-Smith, R., and Johansen, T.A. 1997. Multiple Model Approaches to Modelling and Control. Taylor and Francis, UK.

Narendra, K. S., Balakrishnan, J., and Ciliz, K. 1995. Adaptation and learning using multiple models, switching and tuning. IEEE Control Systems Magazine June, 37-51.

Nielsen, P., Smoot, D., Martinez, R., and Dennison, J. 2000. Participation of TacAir-Soar in Road Runner and Coyote exercises at Air Force Research Lab, Mesa, AZ. Proceedings of the 9th Conference on Computer Generated Forces and Behavioral Representation, (Orlando FL), 173-180.

Newell, A. 1990. Unified Theories of Cognition. Harvard University Press, Cambridge, MA.

Rao, A. 1994. Means-end plan recognition. In Proceedings of KR-94, the Fourth Internatioanl Conference on Principles of Knowledge Representation and Reasoning .

Rao, A. and Georgeff, M. 1995. BDI agents: From theory to practice, In Proceedings of the First International Conference on Multi-Agent Systems, (San Francisco CA).

Rosenbloom, P., Laird, J., and Newell, A. 1993. The Soar Papers: Research on Integrated Intelligence. MIT Press, Cambridge, MA.

Tambe, M. and Rosenbloom, P. 1995. RESC: An approach for real-time, dynamic agent tracking. In Proceedings of IJCAI.95.

Tambe, M. 1996. Tracking dynamic team activity. Proceedings of AAAI-96.

Washington, R. 1998. Markov tracking for agent coordination. In Proceedings of the Second International Conference on Autonomous Agents (Minneapolis/St. Paul MN.