Empirical Comparison of Predictive Models for Mobile Agents

A.E. Henninger, Ph.D.
Soar Technology, Inc.
Orlando, FL. 32817
amy@soartech.com

R. Madhavan, Ph.D.
Intelligent Systems Division
National Institute of Standards and Technology
Gaithersburg, MD. 20899-8230
raj.madhavan@nist.gov

Abstract

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 three predictive models all developed for the same, well-defined modeling task. Specifically, this paper compares the performance of an extended Kalman filter based model, a neural network based model and a Newtonian based dead-reckoning model, all 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 algorithms, the integration of the models into a large-scale simulation environment, and the means by which the performance measures are generated. 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, 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 (Hill et al, 2002), Hidden Markov Models (Washington, 1998), extended Kalman filters (Madhavan and Schlenoff, 2003) and others. This paper addresses a particular application of trajectory prediction in a simulation environment and compares the effectiveness of three approaches: extended Kalman filters (EKF), neural networks, and Newtonian equations. Unique about this particular comparison is the fact that the three techniques being evaluated have been implemented over the same data set. Thus, this cross-comparison enables us to examine the efficacy of results and conclusions drawn by a number of researchers evaluating models independently.

The remainder of this section defines the trajectory estimation problem in the simulation application and describes the previous use of Newtonian equations, neural networks, and EKFs for estimating agent trajectories. Specifically, the following sections present the precise applications of the techniques and then presents the underlying theory of each of the techniques. The paper then describes the test problem used in this study and finally presents the results from the comparisons.

Newtonian Methods–Trajectory Estimation and Theory

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 to predict the trajectory of other agents.

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

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

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 Modular Semi-automated Forces (ModSAF) (Calder et al, 1993) that was used for our experiments.

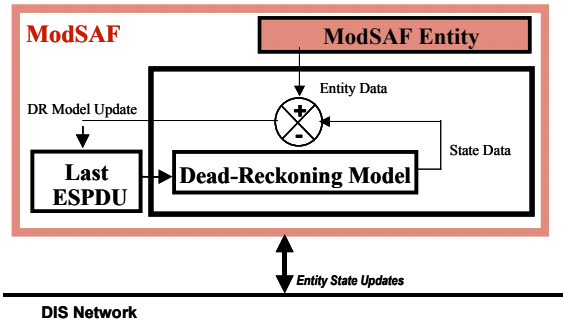

Figure 1. Dead-Reckoning Implementation in ModSAF

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.

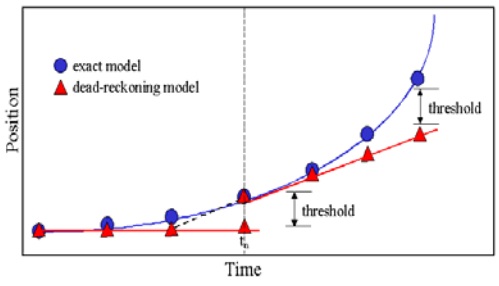

Figure 2. DIS Dead-Reckoning Process

Neural Networks - Trajectory Estimation

Hill, Kim, and Gratch (2002) report on 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 visually 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 ModSAF simulator, a system used by the military for training and research. ModSAF is elaborated in the next major section, "**Methodology**". The agent's intelligence is modeled in Soar (Newell, 1990).

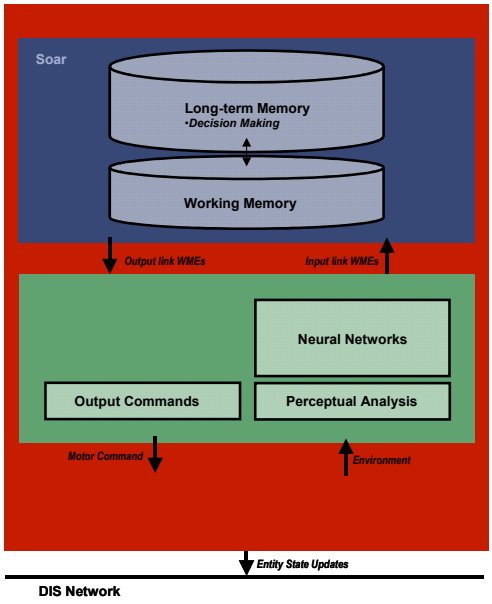

Figure 3. Visual attention for helicopter agent

Citing terrain and cultural features (e.g., roads and bridges) as complicating the trajectory, Hill et al. reject the use of simple linear projections, such as first-order Newtonian equations, and recursive state estimation

techniques, such as Kalman filters. They instead consider the use of neural network based models. 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**. This prediction, as derived from "on-roads" scenario, enables the virtual helicopter pilot to look away from the ground entity for up to 7 seconds with a reported error tolerance of 15 m.

Neural Networks – Computational Form

A variety of researchers have worked in modeling human driving skills such as acceleration, steering, and vehicle following with neural networks e.g., (Pomerlau et al 1994; Pentland and Liu, 1995; and Nechyba and Xu 1997). A neural network is a collection of simple processors or nodes interconnected with each other that learn from examples and store the acquired knowledge in their interconnections, referred to as weights. Neural networks can solve a variety of problems related to non-linear regression and discriminant analysis, data reduction, and non-linear dynamic systems. One of the practical characteristics of neural networks is that they lend themselves to parallel distributed processing using simple processing units rather than a complex CPU. This makes their execution very fast.

The multi-layer feed-forward network is one of the more typical network designs used in neural network applications. The network used in this investigation was a 4-layer feed-forward network, with seven nodes in the first layer representing each dimension of the input vector, one node in the last layer representing the output, and a two hidden layers consisting of twenty nodes in the first hidden layer and 5 nodes in the second hidden layer. This network attempts to develop a matching function between the input and output vectors by using some training algorithm. The training algorithm used in this research is a method known as back-propagation. This method is based on finding the outputs at the last layer of the network, calculating the errors between the actual and the predicted outputs, and then adjusting the network weights to minimize the error. Weight changes are implemented in a backward fashion starting from the weights converging to the output layer and proceeding backwards to the weights that converge to the hidden layer closest to the output layer. These computations are repeated such that the error is propagated back until the weights converging to the hidden layer closest to the input layer are reached.

In short, back-propagation involves a two step process. The first step, the forward pass, propagates the effects of the inputs forward through the network to reach the output layer. This step is governed by three forms of equations. For the 7-20-5-1 architecture used in this study, the first equation (2) shows the total weighted input to the j^{th} node for pattern p is given by:

$$net_j^{(1)}(p) = \sum_{k=0}^{K=7} x_k^{(0)}(p) w_{jk}^{(1)} \quad \text{for j} = 1 \text{ to } 20 \tag{2}$$

Next, the output of the jth node, $y_j^{(1)}(p)$, is given by:

$$y_j^{(1)}(p) = g(net_j^{(1)}(p)) = \frac{1}{1+e^{-net_j(p)}} \tag{3}$$

where $g()$ is the commonly used sigmoid function illustrated in Figure 4.

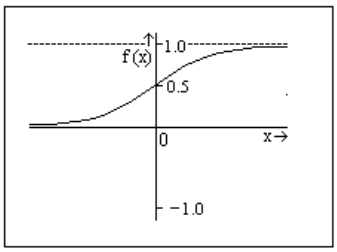

Figure 4. Example of Sigmoid Function

The net input to the i^{th} node for pattern p is similar to equation (2) and is given by:

$$net_i^{(2)} = \sum_{j=0}^{J=20} y_j^{(1)}(p) w_{ij}^{(2)} \quad \text{for i} = 1 \text{ to } 5 \tag{4}$$

and the output of the i^{th} node of hidden layer 2 due to pattern p, $y_I^{(2)}(p)$, is given by:

$$y_i^{(2)}(p) = g(net_i^{(2)}(p)) = \frac{1}{1+e^{-net_i^{(2)}(p)}} \tag{5}$$

where $g()$ is again the sigmoid function.

Finally, the net input to the l^{th} node for pattern p is also similar to equation (2) and is given by:

$$net_l^{(3)} = \sum_{i=0}^{I=5} y_i^{(2)}(p) w_{li}^{(3)} \quad for\ l = 1 \tag{6}$$

and the output of the l^{th} output node due to pattern p, $y_l(p)$, is given by:

$$y_l^{(3)}(p) = net_l^{(3)}(p) \tag{7}$$

which in this instance is not transformed, and thus is the same as the input.

Next, the error function associated with the p^{th} input/desired output {x(p),$d_l(p)$} pair, is given by:

$$E^p(w) = \frac{1}{2}\sum_{l=1}^{L=1}[d_l^{(3)}(p) - y_l^{(3)}(p)]^2 \quad (8)$$

Once the error is computed, the weights are adjusted such that E^p is minimized. This occurs in the second pass, the backward pass, by computing the negative gradient of the error function and taking the partial derivatives of this function with respect to the weights (equations 9, 10, and 11).

$$\Delta w_{li}^{(3)} = -\eta\,(\frac{\partial E}{\partial w_{li}^{(3)}}) \quad (9)$$

$$\Delta w_{ij}^{(2)} = -\eta\,(\frac{\partial E}{\partial w_{ij}^{(2)}}) \quad (10)$$

$$\Delta w_{jk}^{(1)} = -\eta\,(\frac{\partial E}{\partial w_{jk}^{(1)}}) \quad (11)$$

where η is a user defined parameter.

These equations allow errors at the output layer to be propagated backward toward the input layer in proportion to the change in activity at the previous layer. By applying the chain rule of derivation as in (Rummelhart et al 1986) equations 9, 10, and 11 reduce to:

$$\Delta w_{li}^{(3)} = \eta\delta_l^{(3)}(p)y_i^{(2)}(p) \quad (12)$$

$$l = 1 \quad i = 1\,to\,5$$

where

$$\delta_l^{(3)}(p) = g'(net_l^{(3)}(p))[d_l^{(3)}(p) - y_l^{(3)}(p)]$$

$$\Delta w_{ij}^{(2)} = \eta\delta_i^{(2)}(p)y^{(1)}{}_j(p) \quad (13)$$

$$i = 1\,to\,5 \quad j = 1\,to\,20$$

where

$$\delta_i^{(2)}(p) = g'(net_i^{(2)}(p))\sum_{l=1}^{L=1} w_{li}^{(3)}\delta_l^{(3)}(p)$$

$$\Delta w_{jk}^{(1)} = \eta\delta_j^{(1)}(p)x_k(p) \quad (14)$$

$$j = 1\,to\,20 \quad k = 1\,to\,7$$

where

$$\delta_j^{(1)}(p) = g'(net_j^{(1)}(p))\sum_{i=1}^{I=5} w_{ij}^{(2)}\delta_i^{(2)}(p)$$

Each of these weight adjustments directs the network towards a solution to the input/output mapping. That is, these weights are training the network to produce a certain output given a set of inputs. This is one of the fundamental benefits of the neural network approach. With the proper training and representation, the network will self-organize to arrive at a mapping of how the responses are formed and there is no need to acquire and represent an expert's knowledge in terms of rule sets.

Extended Kalman Filters - Trajectory Estimation

Madhavan and Schlenoff (2003) report on a system developed for use within the 4D-Real-Time Control System (RCS) (Albus et al, 2002) reference model architecture. This system was developed to generate short-term predictions of moving objects trajectory such that it will support the planning required for unmanned ground vehicles (UGVs) to move efficiently and dynamically avoid collisions in the presence of dynamic objects with unknown trajectories.

The data being recorded for the moving object includes:

- Current Location: x, y, and z locations as well as terrain type
- Dimensions
- Velocity
- Color: most predominant color, where appropriate
- Motion Pattern: based on multiple concurrent sets of position data

To test the prediction algorithms developed in this effort, Madhavan et al. implemented the algorithms in the OneSAF testbed (OTBSAF). In doing so, they use data from OTBSAF under the assumption that UGV has "perfect sensor data". In other words, the data listed above are supplied with no associated uncertainty. To compensate for this, they implemented a noise model in OTBSAF to use for sensor data.

While they don't specify results and associated errors, the authors acknowledge that, as expected, the associated uncertainty with predicted position increases as the prediction time horizon increases. Because the confidence in their estimates decreases as the planning horizon increases, they suggest the use of a threshold to determine the point at which the algorithm no longer provides adequate estimates to support safe path planning. Deriving this threshold is the subject of future research.

Extended Kalman Filters – Computational Form

Kalman's prediction theory allows the computation of the best estimate of a future system state by using the most recent estimates of system state along with the system dynamic model. With appropriate interpretation, covariance analysis inherent in the Kalman filtering techniques serves as confidence measure indicative of the uncertainty in the predicted system states. The EKF thus provides a convenient measure of prediction accuracy through the covariance matrix.

The extension of the linear Kalman filter to a non-linear system is termed "extended" Kalman filtering and it is obtained through the linearization of the non-linear state

and observation equations. In the context of this paper, the state vector **x** is comprised of the predicted position and orientation of the vehicle. In view of the availability of data at discrete instants of time, a discrete-time formulation of the continuous-time vehicle kinematics models is necessary.

To formulate a Kalman filter algorithm, process and observation (measurement) models are needed. In discrete-time, only discrete sampling instants $t_0, t_1, \ldots$ are considered. Integrating the continuous-time process model between two consecutive time steps usually derives the discrete-time process model. A general discrete-time process model can be expressed as:

$$_{k} = (\ _{k-1},\ _{k}, k) + \ _{k} \tag{15}$$

where **f**(.,.,*k*) is a discrete function that maps the previous state and control inputs to the current state, $\mathbf{x}_k$ is the state at time instant *k*, $\mathbf{u}_k$ is a known control vector, and $\mathbf{w}_k$ is the discrete process noise. The process noise is assumed to be a Gaussian-distributed random variable of zero mean and constant covariance.

Observations of the state $\mathbf{x}_k$ are made according to the observation model:

$$_{k} = (\ _{h}, k) + \ _{k} \tag{16}$$

where **h**(.,*k*) is the discrete function that maps the current state to observations. $\mathbf{v}_k$ is the measurement noise source and plays a similar role in the observation model to the role played by the process noise in the process model. It accounts for effects that are not modeled explicitly and is assumed to be a Gaussian-distributed random variable of zero mean and constant covariance.

In an autonomous vehicle navigation context, the prediction stage uses a model of the motion of the vehicle (a process model having the form described in Equation (15) to predict the vehicle position, $\hat{\ }_{(k|k-1)}$, at instant *k* given the information available until and including instant *k-1*. The state prediction function **f**(.) is defined by Equation (15) assuming zero process and control noise. The prediction of state is therefore obtained by simply substituting the previous state and current control inputs into the state transition equation with no noise. The predicted covariance of the vehicle, $_{(k|k-1)}$, is thus computed using the Jacobian of the state propagation equation linearized about the current vehicle state estimate.

Once the state and covariance predictions are available, the next step is to compute a predicted observation and a corresponding innovation for updating the predicted state. The difference between the actual observation and the predicted observation at time step *k* is termed the innovation and is written as:

$$\nu_k = \ _{k} - \hat{\ }_{(k|k-1)} \tag{17}$$

The innovation covariance is found by squaring the estimated observation error and taking expectations conditioned on the first *(k-1)* measurements

$$S_k = \nabla\ _{x_k}\ _{(k|k-1)} \nabla\ ^{T}_{\ _k} + \ _{k} \tag{18}$$

The observations that arrive are accepted only if the observation falls inside the normalized innovation validation gate given by:

$$\nu_k^T\ \ ^{-1}_{k} \nu_k \le \varepsilon_\gamma \tag{19}$$

where $_{k}$ is the innovation defined as the difference between the actual and predicted positions. The value of ε_γ can be chosen from the fact that the normalized innovation sequence is a χ^2 random variable with *m* degrees of freedom (*m* being the dimension of the observation). Once a validated observation is available, the update of the estimate equal to the weighted sum of the observation and the prediction can be computed as:

$$\hat{\ }_{(k|k)} = \hat{\ }_{(k|k-1)} + \ _{k} \nu_k \tag{20}$$

where $_{k}$ is the Kalman gain matrix determined by the relative confidence in vehicle prediction and observation and determines the influence of the innovation on the updated estimate.

The covariance update is given by:

$$_{(k|k)} = \ _{(k|k-1)} - \ _{k}\ _{k}\ ^{T}_{k} \tag{21}$$

where the Kalman gain matrix is:

$$_{k} = \ _{(k|k-1)} \nabla\ ^{T}_{\ _h}\ \ ^{-1}_{k} \tag{22}$$

Methodology

This paper compares the performance of three models based on neural-networks, extended Kalman filters, and Newtonian dead-reckoning. Like the systems described in

sections on dead-reckoning and neural networks for trajectory estimation, this experiment is conducted with a data set generated in ModSAF, a training and research system developed by the Army's Program Executive Office for Simulation, Training, and Instrumentation Command (PEO STRI). 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 5. 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.

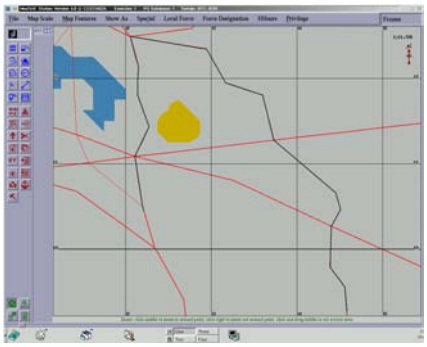

Figure 5. Route Used for Experiment

Neural Network Implementation

For this application, a feed-forward architecture developed with back-propagation training was used to develop the neural network that predicts the change in an entity's speed. The network used a sigmoid function at the hidden nodes and a linear transformation at the output nodes. The configuration of the network in each of the models may be seen in Table 1 where the inputs were normalized according to equations 24 – 40 below. Fundamentally, the inputs for the network 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

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

specific predictors are expressed in 25 – 32, and the parameters making up those inputs are explained in 33 – 40

$$S_t = S_{t-1} + \Delta S_t \quad (24)$$
where $\Delta S_t = f(Ra_{t-1}, Rb_{t-1}, Rc_{t-1}, Rp_{t-1}, Rs_{t-1}, HRab_{t-1}, HRbc_{t-1}, Hz_{t-1})$

$$Ra_t = S_t / (Da_t + M) \quad (25)$$
$$Rb_t = S_t / (Db_t + M) \quad (26)$$
$$Rc_t = S_t / (Dc_t + M) \quad (27)$$
$$Rp_t = S_t P_t / M \quad (28)$$
$$Rs_t = S_t / M \quad (29)$$
$$HRab_t = Hab_r \times Hxy_t \quad (30)$$
$$HRbc_t = Hbc_r \times Hxy_t \quad (31)$$
$$S_t = entity\ speed\ at\ t \quad (32)$$
$$Da_t = distance\ to\ previous\ waypoint \quad (33)$$
$$Db_t = distance\ to\ current\ waypoint \quad (34)$$
$$Dc_t = distance\ to\ next\ waypoint \quad (35)$$
$$M = march\ order\ speed \quad (36)$$
$$P_t = perpindicular\ distance\ to\ road \quad (37)$$
$$Hab_t = direction\ of\ road\ segment\ ab \quad (38)$$
$$Hbc_t = direction\ of\ road\ segment\ bc \quad (39)$$
$$Hxy_t = entity\ orientation \quad (40)$$

Of the 13760 simulated movement updates, 860 examples were used for training the speed network and 859 examples were used for validating the training. 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

	ΔS **Error(m/s)**
Training	0.259977±2.04558
Validation	0.206374±0.82532

Extended Kalman Filter Implementation

The kinematic vehicle model shown in Equation (16) is used to predict the positions (x,y) of the vehicle and the ground truth estimate is used for the orientation prediction

directly. The position of the vehicle is predicted until the errors exceed a defined threshold.

Once the errors are above a given threshold, an update is deemed to be performed by utilizing the observations from ModSAF. We check the validity of the observation by testing if it falls within the normalized innovation gate. A validated observation is then used to update the states of the EKF and the estimation cycle continues as before.

Results

The neural network models and extended Kalman filter models were implemented in such a way that their performance for predicting entity location could be compared with the dead-reckoning model.

An error is generated in our system according when the entity's actual location is greater than .5m away from the model's predicted location for the entity. Using these error thresholds, the neural network models required 170 updates, the extended Kalman filter model required 240 updates, and the Newtonian model required 267 updates. This information is presented in far left column of Figure 6.

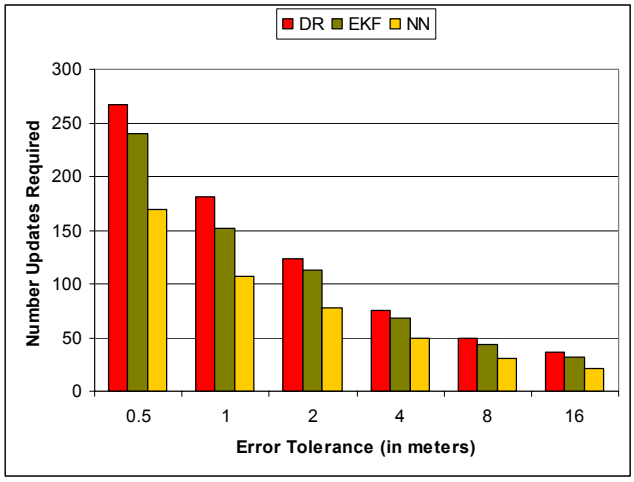

Figure 6. Number of Updates Required by NN, EKF, and DR Models

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 were measured according to an error tolerance of .5m, follow-on tests incremented these error thresholds to examine model utility over increasing error tolerance. As evidenced by the progression of columns of Figure 6, as the error tolerance increases, the predictive advantage that more sophisticated modeling techniques (i.e., NNs and EKFs) have over dead-reckoning models becomes less significant for this modeling task.

Summary and Conclusions

While a first step in the comparison between tracking models, this analysis is not complete. Still needed are processing times for models and a sense of how well they generalize to other types of terrain and roads. This analysis does confirm, generally, that the more complicated a model becomes in terms of numbers of factors and non-linear terms, the more it tends to do a better job at predicting the source model. Along those lines, one might expect processing time requirements to be inversely related. That is, as the more complicated the model gets, the more processing time it will require. Assuming this is true, 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 given the high cost of communications. 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 difference in predictive utility of the 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 NN or EKF configuration could have yielded different results. We cannot claim that these are the best configurations for this specific modeling task. Other configurations may be better. Also, combinations of approaches may be better. For example, one approach advocated in the control literature (Murray-Smith and Johansen, 1997; Narendra et al, 1995) is to employ modular models such as mixing different modeling techniques as they best apply to the problem locally. In the problem discussed in this paper, one interesting test would be to use a less computationally expensive model in the straight parts of the road database and then use another model to guide the turns, as previous work (Henninger et al, 1999) has suggested that this appears to be where the majority of updates are required.

References

Albus, J. et al., 4D-RCS Version 2.0: A Reference Model Architecture for Unmanned Vehicle Systems. Tech Report NISTIR 6910, National Institute of Standards and Technology, Gaithersburg, MD 20899.

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., Kim, Y., Gratch, J. (2002). Anticipating where to look: predicting the movements of mobile agents in complex terrain. In Proceedings of the 2002 Autonomous Agents in Multi-agent Systems (AAMAS) Conference. Bologna, IT. pp. 821 – 827.

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.

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

Madhavan, R. and Schlenoff, C. (2003). Moving Object Prediction for Off-road Autonomous navigation. In Proceedings of 2003 SPIE Conference. Orlando,FL.

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.

Nechyba, M. and Xu. (1997). Cascade Neural Networks with Node-Decoupled Extended Kalman Filtering, In the Proceedings of IEEE Int. Symp. On Computational Intelligence in Robotics and Automation, vol. 1, pp. 214-219.

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

Pentland, A. and Liu, A. (1995). Toward Augmented Control Systems. Proceedings of Intelligent Vehicles, vol. 1, page 350-55.

Pomerlau, D., Thorpe, C., Longer, D., Rosenblatt, J.K., and Sukthankar, R., (1994). AVCS Research at Carneghie Mellon University. Proceedings of Intelligent Vehicle Highway Systems America 1994 Annual Meeting, p. 257-262.

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).

Rummelhart, D., and McClelland, J. (1986). Parallel Distributed Processing. 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.