IPT-EGVE Symposium (2007)
B. Fröhlich, R. Blach, and R. van Liere (Editors)

Adaptive Classifier System-Based Dead Reckoning

Samir TORKI, Patrice TORGUET, Cédric SANZA
Institut de Recherche en Informatique de Toulouse
118 route de Narbonne, 31062 Toulouse, France
{torki | torguet | sanza}@irit.fr

Abstract

Most dead reckoning implementations are based on DIS' specifications and only use a single prediction model during the whole simulation. However, several studies manage to improve dead reckoning's performance by defining prediction model selection strategies. Nevertheless, these approaches are either too generic and based on empirical results or too specific and only have few fields of application.
This paper presents our approach that is meant to determine, among a set of extrapolation models, the one to apply in any given situation. It is based on classifier systems, adaptive evolutionary systems that are more generally involved to create artificial creatures in the field of "artificial life".
Using such systems enable us to define a general model that can generate simulation-specific rules with relatively little work. Indeed, they just require defining the parameters that have to be taken into account and the criteria to optimize (e.g. accuracy, amount of updates...). Then, the system makes a set of rules emerge through a trial/error process in order to define more efficient and finer prediction model selection strategies.

Categories and Subject Descriptors (according to ACM CCS): I.6.0 [Simulation and Modeling]: General

1. Introduction

Simulations of virtual environments are very demanding in terms of computing power. Distributed virtual environments (*DVEs*) are aimed to supply such a power through the use of networked computers. Distribution also enables several users to participate and collaborate within a same environment and tends to provide them a more enjoyable experience. This is why such distributed simulations are involved in many kinds of fields, from military simulations [DFW97] to video games [Mau00].

Nevertheless, these applications have to cope with different kinds of bottlenecks mainly due to CPU and network limitations. CPU limitations are generally bound to calculations that are badly distributed between the hosts involved in the simulation. Such issues tend to be solved through the use of *load balancing* techniques [TL01].

Network related issues are generally bound to *bandwidth limitation*, *latency* and *jitter*. At first, bandwidth limitations often require a reduction in the amount and in the size of the messages exchanged between hosts in order to avoid network congestion.

Furthermore, all *DVEs* and especially those running over the Internet also have to cope with *latency* and *jitter*. *Latency* is the delay it takes for a message to travel across the network while *jitter* can be defined as the variation of latency over time. Such delays are mainly due to routing as it takes some time for routers to process packets.

Unfortunately, distributed virtual environments involving human users are even more sensitive to latency and jitter than any other kind of distributed application. Indeed, those phenomena have a huge impact on the course and the "realism" of a simulation. For example, in the case of First Person Shooters, it has been clearly shown that higher latencies tend to deteriorate players' performance [BCL+04].

Jitter makes the drawbacks induced by latency even worse. Indeed, as updates can be received at variable rates, this generally leads to implausible and unpredictable jittery motions.

Several methods are aimed to overcome the effects of bandwidth limitation, latency and jitter. Most of them achieve that goal by reducing the amount of data transmitted through the network in order to decrease both link

and router utilization. Among those techniques, the most widely used are *multicast transmission* [Dee89], *data lossless compression* [SZ99], *filtering* [Mor96] and *dead reckoning* [CDG+93].

This paper deals with the dead reckoning algorithm. This technique consists in making hosts extrapolate entities' state in order to reduce the number of emitted updates. Section 2 is aimed to present this algorithm more precisely.

Most distributed simulations involve that algorithm, nevertheless they generally content themselves with using a single - most often linear - extrapolation model. However, depending on the nature of the simulation and of the simulated entities, different kinds of predictions may give better results by providing better accuracy and making updates decrease. Section 3 presents existing researches that attempt to determine prediction model selection strategies in order to optimize the dead reckoning process. Nevertheless, these approaches are either too generic and based on empirical results or too specific and only have few fields of application.

The system we introduce in this paper is based on artificial learning systems called *classifier systems* (CS) that are presented in section 4.2. It consists in taking advantage of the adaptive capabilities of CSs in order to provide designers a generic system that can produce simulation and entity-specific strategies. Furthermore, our approach also offers a way to relieve distributed simulation designers of a demanding and very prone to bugs task that consists in defining such rules by hand.

The results obtained using our approach are presented in section 5 as well as a comparison with the results obtained using existing strategies.

2. The dead reckoning algorithm

This algorithm was introduced in SIMNET [CDG+93] in order to reduce the effects of latency and network bandwidth limitation. It became part of the DIS (Distributed Interactive Simulation) standard [oISS95].

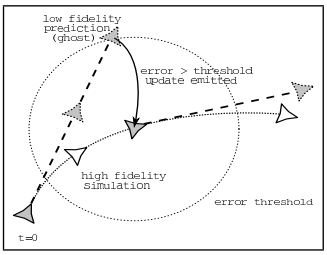
Dead reckoning consists in predicting entities' state for some time periods rather than sending *protocol data units* (PDU) at every simulation step. Indeed, two models are used to simulate every entity involved in the simulation:

- a CPU-intensive *high fidelity* model representing the exact state of the entity,
- a less CPU consuming *low fidelity* model used to predict an approximation of this state.

The *high fidelity* model is only run by the host that manages the entity. The *low fidelity* prediction is run by all the hosts involved in the simulation.

The host that simulates the entity has to run both models in order to know how remote hosts perceive this entity.

This is also necessary to be able to define the *deviation* between those two models. Indeed, the dead reckoning algorithm requires that computer to send updates only when the distance between the *ghost* (i.e. the low fidelity modeled entity) and the real entity exceeds a given threshold. As soon as a host sends or receives an update PDU, it re-synchronizes its ghost entity in order to take the latest information into account and resets the prediction model.



1: Dead reckoning

3. Related work

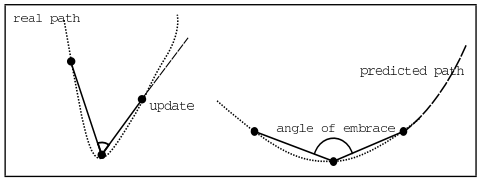
Most studies related to dead reckoning aim to increase this algorithm's performance by modifying either the extrapolation model or the way the error threshold is defined.

The approach presented in [CLC99] manages to improve the algorithm through dynamic threshold values. Indeed, according to its needs in terms of accuracy, every entity involved in the simulation can ask another one to use a specific threshold. In that way, precise positions are obtained using a low threshold while a bigger value gives less precise but less frequently updated information. To determine an entity's needs, the proposed method takes *areas of interest* [Mor96] into account.

[CLC99] also provides a method designed to select a prediction equation automatically. This approach relies on the fact that the motion of an entity can be either a *smooth*, *bounce* or *jolt* motion. Then, some rules have been defined to select a prediction scheme according to the motion class and the threshold value. These were established according to the results obtained with some generic curves such as sinusoids or saw-toothed curves.

The *position history based dead reckoning* (PHBDR) presented in [SC94] uses about the same approach to calculate extrapolation polynomials' coefficients. Indeed, it requires PDUs to contain only entities' position and orientation. Then, all the derivatives required to build the 1^{st} and 2^{nd} order polynomials are calculated from the last 2 (velocity) or 3 updates (acceleration). The selection between linear or quadratic extrapolation is performed according to the value of the *angle of embrace* formed by the last 3 update positions (figure 2). This resulted in better performance than DIS' original dead reckoning except for some situations such as circular movements. It is worth noticing that *angles*

of embrace were also involved in [ZGD04]'s *pre-reckoning* algorithm.



2: Angle of embrace

Some improvements to dead reckoning are also obtained by using different kinds of extrapolation techniques. For example [CH01] uses the same *history based dead reckoning* as [SC94] except that the Lagrange polynomial-based prediction deals with quaternions rather than positions. It is worth noticing that this approach also implements [SC94]'s *angle of embrace* as a selection criterion for the degree of the extrapolation polynomial.

The Taylor expansion-based extrapolation model introduced by [HY06] slightly differs from most dead reckoning approaches. Indeed, update packets do not actually contain the position, velocity and acceleration of the entity when the threshold was exceeded. They rather carry entity's position during the steps that preceded the transmission of an update. Then, the derivatives required by the Taylor expansion are calculated in the same way as they are in the *position history based dead reckoning*. The experiments performed with this model led to a reduction of the extrapolation error in the case of smooth trajectories obtained by writing words on a touch panel screen.

Some other works base their prediction algorithm on "artificial life" techniques. For example, [LH05] uses a bayesian network in order to predict the position of the cursor in the field of collaborative environments. In the same way, [And05]'s neural network tries to extrapolate the velocity and the force of a haptic device from their previous values.

4. Our approach

4.1. Polynomial prediction

The early experiments performed with our approach involved a polynomial prediction model. The way the polynomials are calculated is defined in a generic way, so that it can at least reproduce DIS and the position history's extrapolation formulae. Like many other dead reckoning schemes [HY06] [SC94] [oISS95], our prediction model is based on Taylor series. However, our approach differs from the others in the way the derivatives are calculated. Indeed, the extrapolation polynomial is calculated in this way:

$$x_{pred}(t) = \sum_{i=0}^{deg} \left(\frac{(t-t_{u_0})^i}{i!} \cdot \frac{d^i x}{dt}(t_{u_0}) \right) \qquad (1)$$

where u_i represents the last i^{th} update that was received at t_{u_i}, *deg* the degree of the extrapolating polynomial and *nbDer* the maximum derivative order extracted from an update packet.

Applying this formula, requires to calculate $\frac{d^i x}{dt}(t_{u_0})$s:

- if $i \leq nbDer$ and if the i^{th} derivative is provided in u_i, $\frac{d^i x}{dt}(t_{u_i})$ corresponds to the value found in u_i,
- in any other case, $\frac{d^i x}{dt}(t_{u_i})$ is calculated in a recursive manner :

$$\frac{d^i x}{dt}(t_{u_i}) = \frac{\frac{d^{i-1} x}{dt}(t_{u_i}) - \frac{d^{i-1} x}{dt}(t_{u_{i+jump}})}{t_{u_i} - t_{u_{i+jump}}} \qquad (2)$$

Such a method enables us to simulate DIS' dead reckoning by joining all the derivatives required to every transmitted update and by giving *deg* and *nbDer* the same value. *Position history based dead reckoning* can also be obtained by setting *nbDer* to 0 and *jump* to 1 or by only reporting entities' position in updates. In addition, making *jump* vary, allows to choose the appropriate combination between long term (global) information given by previous updates and short term (local) data brought by the latest ones.

4.2. Classifier systems

A classifier system (CS) is an adaptive system aimed to generate a set of '*if-then*' rules through a learning process. Classifier systems are commonly used in the field of *artificial life* [San01].Nevertheless, our approach uses them for what they are above all: optimization systems [Gol85].

This section describes Holland's *Learning Classifier System* [HR78](LCS) from which most CSs - including the αCS we use - derive from.

4.2.1. Architecture

A CS runs as a *perception→decision→action* loop. At first, it perceives information from its environment through its *sensors*. Then, its *decision unit* selects the action to trigger according to the sensor inputs. Finally, the system modifies its environment by executing that selected action through its *effectors*.

Most of a classifier system's work is achieved by its decision unit. It consists of a base of "*if condition then action*" rules called *classifiers*. These are bit strings in which:

- *condition* is represented by a sequence of 0 (false), 1 (true) and # (don't care) trits,
- *action* is a sequence of 0 and 1 bits.

This unit's function is to make useful classifiers *emerge* from a randomly initialized base. This is performed by preserving the most valuable rules and by replacing the bad ones. To distinguish good rules from bad ones, each classifier is given a strength that denotes the adequacy of a given *action*, when *condition* is met.

The classifier system's running cycle defines the way that strength is calculated. At the first step of that cycle, the information perceived by the sensors are converted into a 0,1 bit string message and stored into a message list. Then, three cycles are run in order to define the action to trigger and to update classifiers' strength: the *performance*, *credit assignment* and the *rule discovery* cycles.

4.2.2. Performance cycle

This cycle intends to choose the most appropriate rule to trigger. Firstly, the algorithm selects all the rules which *condition* part matches at least one of the bit strings contained in the message list. Then, these selected classifiers make a bid proportional to their strength in order to be triggered. The winner is designated via a *roulette wheel* selection algorithm (i.e. the probability of a classifier to win is proportional to the bid it makes). Finally, these wining classifiers "post" their *action* part as new messages into the message list. These actions can be either used to trigger a given effector or to select other rules at the following iteration.

4.2.3. Credit assignment cycle

Credit assignment cycle's goal is to update every classifier's strength so that useful rules become stronger and have greater probability to win at the end of the performance cycle. The most widely used credit assignment algorithm is the *bucket brigade algorithm* presented in this section. First of all, every bidding classifier selected has to pay the bid it made during the performance cycle. Payment is shared between the classifiers that caused that classifier's selection by sending messages during the previous iterations.

Then, the usefulness of an action is defined by giving the system a reward according to the effects it had. Bad actions generally lead to a more or less negative reward while good actions lead to positive ones and result to an increase of the rule's strength.

The credit assignment cycle also manages to make unnecessary rules become even weaker by taxing non selected or non winning classifiers. This enables the system to replace those classifiers by potentially better ones during the rule discovery cycle.

4.2.4. Rule discovery cycle

The rule discovery cycle aims to replace the least useful classifiers by potentially better ones. This cycle is based on a *genetic algorithm* which population consists of the classifiers contained in the rule base. Classifiers' strength is used as a fitness function, so that weaker rules tend to "die" and get replaced by good ones' children. Those children are obtained through different genetic operators such as *selection*, *crossover*, *mutation* and *covering* [Gol85].

4.2.5. αCS classifier systems

Our application involves αCS [San01], a variation of Holland's LCS. The most important feature introduced by those classifier systems rests on their multi-objective capabilities. Indeed, they enable designers to specify the rewards the system receives through several easy-to-define fitnesses rather than a single complicated one. Moreover, αCSs also provide a mechanism of fitness prioritization in order to give some goals more importance than others.

Such characteristics make this kind of classifier systems well suited to our approach as the next section shows.

4.3. The dead reckoning rule discovery system

Our approach aims to determine the most appropriate prediction scheme to use in any given situation without having to define all the rules by hand. Indeed, such a task is generally very demanding and prone to bugs and forgetting. To achieve such a goal, our method attempts to use classifier systems' adaptation and learning capabilities. Indeed, on every host, every "high fidelity" entity is associated to a classifier system. Such a classifier system will only require designers to define sensors, effectors and retributions and let the system determine the rules rather than having to define them by hand.

Our choice for joining a classifier system to every "high fidelity" entity comes from the fact that this enables us to define entity-specific rules. This can have a huge impact when the simulation involves different kinds of entities that can behave and move in different ways. For example, the model used for human-beings' motions may differ from the model used for planes or tanks and such differences should be reflected in the rule the system should generate.

The system can be run once the following elements have been defined:

- sensors: the parameters that have to be taken into account in order to select a prediction scheme,
- retributions: the goals that have to be fulfilled (e.g. reducing the overall error, reducing the amount of updates emitted...),
- effectors: the prediction models that can be used.

The initial rule base used by the system can be defined either from an empty or randomly initialized base or from a base resulting from a previously run simulation or even from a set of *a priori* defined rules.

Then, during one or several simulations, the system is trained in order to make interesting rules emerge. For that purpose, we had to determine when to activate the classifier system by sending it a retribution and requesting it to determine the action to trigger according to the sensors' states. Several strategies were applicable:

- triggering the system at every step: such a strategy is not

correct as the system cannot get relevant information on the effects of using such or such a prediction model,
- triggering the system periodically: at first, this strategy would require extra work from the designers as the duration of periods would have to be defined. Furthermore, the rules obtained at the end of the training process should be - to be consistent - applied periodically too. This raises the synchronization problem that often occurs in distributed application.
- triggering the system at every update: we chose such a strategy as it also enables us to follow the model used in DIS to determine the prediction scheme that remote hosts use.

On a given host, a step in a simulation using our approach runs in the way presented in figure 3.

for all simulated entities e_{ij}s **do**
- step $e_{r_{ij}}$ high fidelity model
- step $e_{g_{ij}}$ low fidelity model
- log criteria required to calculate retribution at the end of the stage

if distance($(e_{r_{ij}}$,$e_{g_{ij}})$ > *threshold*) **then**
- calculate retribution from logged data
- apply retribution to the αCS for the ending stage
- calculate sensors' state from the state of the entity and of its surrounding environment
- activate the αCS using those sensors and get the action to perform
- create an update packet containing the update information and the model to use
- send the update packet
- apply the changes on $e_{g_{ij}}$ (i.e. position update and prediction model change)

end if
end for

3: Global algorithm

5. Experiments and results

The experiments presented in this section are aimed to test and validate our approach. To reach such a goal, we decided to confront our system to different testbenches using criteria that are quite similar to those presented in [SC94] and [CLC99]. These include the use of different trajectories in order to try the system's ability to adapt to different kinds of entities. This section also refers to experiments that are meant to raise the classifier system based dead reckoning's capability to reach different goals according to designers' needs.

As our simulations involve quite "generic" motions, we decided to make the system's sensors only take the values of the *angles of embrace* (section 3) into account. However, other kinds of sensors can be defined according to simulation-specific features such as the distance to the nearest enemy or the distance to the ground.

Most researches involving angles of embrace define an arbitrary threshold to distinguish large angles from small ones and to determine the prediction model to trigger. Unlike them, we decided to define sensors in a way that enables us to generate rules according to finer ranges of angles of embrace. Indeed, the system's goal is to define the rules to trigger according to the interval the cosine of the angle of embrace belongs to, among the following bounds: -1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75 and 1.

For all the experiments presented in this section, all the effectors the system can trigger are defined in a way that enables the system to use different values for the *deg*, *nbDer* and *jump* parameters.

The following part of this section presents the results obtained using our system with different kinds of trajectories and retribution models.

5.1. Smooth trajectory punctuated with sudden velocity changes

This experiment involves an entity that follows a trajectory that mixes smooth parts (that can be assimilated with sine curves) and sudden changes in its velocity when reaching 0 along the X axis. The sensors and effectors used by the system are the same as the ones presented in the previous section.

The first experiment performed with this trajectory consists in "asking" our system to reduce as much as possible the *deviation* between the high and low fidelity models. Optimizing such a criterion should enable us to perform a better prediction and enhance dead reckoning's performance. Indeed, lower deviations should lead to fewer updates as the threshold would be exceeded less often. In the case of this experiment, the retribution given to the system at the end of every prediction stage is calculated from the average value of the deviation during the ending stage.

Early tests pointed out that this criterion could not be directly used by the system as it didn't give reliable information about the way the system behaved and about its performance. Rewarding or penalizing the system according to a more 'global' trend about the evolution of that measurement helped to solve that problem. Indeed, the results presented in this section were obtained by rewarding the system using the slope of the curve representing the cumulative value of the average deviations. The retribution given to the system at the end of every prediction stage is calculated in this way:

- At the end of the n^{th} prediction stage, the average value of deviation is calculated as follows:

$$AD_n = \frac{\sum deviation}{nbSteps} \quad (3)$$

- The cumulative total of that average is computed and stored in order to calculate the slope:

$$C_n = \sum_{i=0}^{n} AD_i \tag{4}$$

- The slope is calculated from the last m values of C_{AD_n} using a linear regression model:

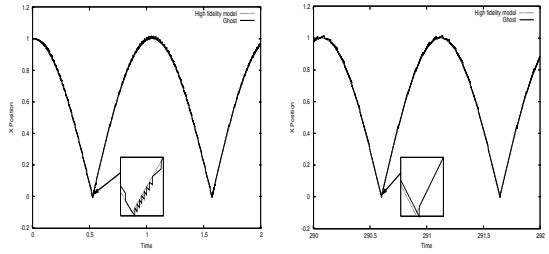
$$slope = \frac{\sum_{i=1}^{m}\left[\left(\frac{m-1}{2}-i\right)\cdot\left(C_{n-i+1}-\frac{\sum_{j=1}^{m}C_{n-j+1}}{m}\right)\right]}{\sum_{i=1}^{m}\left(\frac{m}{2}-i\right)^2} \tag{5}$$

- Finally, the retribution is defined in this way in order to minimize the slope (i.e. the smaller the slope is, the greater the retribution will be):

$$retribution = \gamma - slope \tag{6}$$

The experiments this section deals with, were performed with a randomly initialized αCS and a 0.015 threshold value for the dead reckoning algorithm.

Figure 4 presents a comparison between the trajectory of the ghost at the beginning of the learning process and its trajectory several simulation steps later. These graphs show that our system found a way to reduce the amount of updates required both near the "bouncing point" and in the smooth parts of the trajectory.



(a) At the beginning (b) During learning

4: Ghost's trajectory

During that simulation, the classifier system made the following rules emerge (all the other classifiers involved in the system are not significant as they are very weak):

aoe	-1.0	-0.75	-0.50	-0.25	0.00	0.25	0.50	0.75	1.00
jumps	1	1			8			4	
nbDeg	0	0			0			0	
der	2	1			1			1	

This rule base has some interesting features:

- At first, it does not involve all the possible ranges the angle of embrace can belong to, but only the ones the system met and should meet in the future. Such a characteristic tends to become very useful when designers want to implement those rules directly in their simulation engine. Indeed, with fewer rules the system becomes easier to code, the amount of possible bugs can also be reduced and performance increased.
- Secondly, the rules obtained by the system are consistent with [SC94] and [CLC99]'s results. Furthermore, our approach provides more precise rules as it takes finer intervals into account.

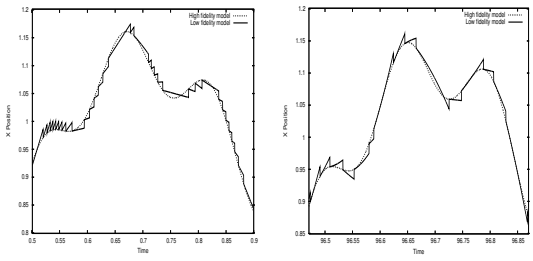
5.2. Complex trajectory

In the following experiments, the simulated entity follows a more complex trajectory that mixes *smooth*, *bounce* and *jolt* motions [CLC99].

5.2.1. Minimizing deviation

The first experiment performed with such a trajectory is aimed to try the ability of our system to adapt to different kinds of entities and movements. Indeed, the only difference between this experiment and the previous one comes from the fact that the simulated entities do not follow the same paths.

Figure 5 shows the way the ghost's trajectory evolves and shows that the system finds a way to reduce the amount of emitted updates.



(a) At the beginning (b) During learning

5: Ghost's trajectory - minimizing average deviation

Nevertheless, this experiment leads to rules that are quite different from the ones found in section 5.1:

aoe	-1.0	-0.75	-0.50	-0.25	0.00	0.25	0.50	0.75	1.00
jumps	1	4	4	4	4	4	1	8	
nbDeg	0	0	0	0	0	0	0	0	
der	2	2	1	2	1	2	1	1	

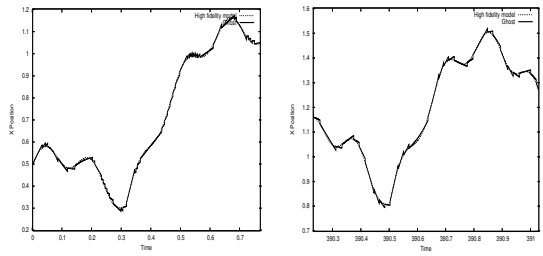
It is worth noticing that all the rules determined by the system have *nbDer* set to 0. This means that according to it, the *position history based* approach is the most appropriate one. Nevertheless, these rules do not exactly correspond to

what [SC94] proposes as the larger values of *aoe* do not necessarily require the use of a first order polynomial. However, our approach tends to determine the values *deg* and *jump* should take in a more precise manner.

5.2.2. Maximizing prediction stages' duration

This section presents the results obtained using the same system running with the same parameters except for the retribution model. Indeed, in this experiment, the system is aimed to maximize the duration of prediction stages, which should reduce the number of updates. The reward given to the system at the end of every extrapolation phase is calculated in about the same way as deviations were in the previous sections (i.e. by calculating a slope from the cumulative total of stage durations).

As figure 6 shows, requesting the system to maximize those durations leads to a decrease in updates.



(a) At the beginning (b) During learning

6: Ghost's trajectory - maximizing prediction stage duration

The stronger classifiers that emerged during that simulation are the following ones:

aoe	-1.0	-0.75	-0.50	-0.25	0.00	0.25	0.50	0.75	1.00
jumps	8	1		8	4		1	8	
nbDeg	1	0	1	0	0	1	0	0	
der	2	2	1	2	2	1	1	1	

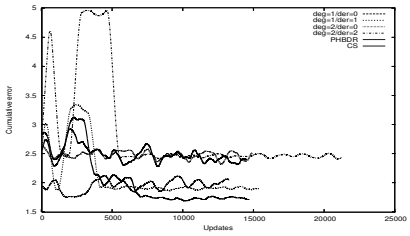
Once again, those rules differ from the ones found before. It is worth noticing that some classifiers trigger a DIS similar approach (i.e. $deg = nbDer$) in some situations and a PHBDR-like prediction when the angle of embrace belongs to some other intervals. Those results tend to show the adaptive capabilities of our system as it is able to determine interesting rules that fulfill different kinds of goals, without requiring much work from designers.

5.3. Comparison with other prediction model selection strategies

This final section is intended to compare our approach to different prediction model selection strategies. Two experiments were performed, the first one to determine the value of the average deviation during prediction phases. The second one took the stage duration criterion into account. These experiments involve 6 prediction model strategies:

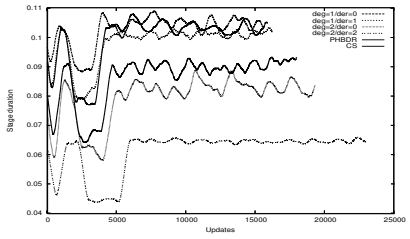
- "static" position history based dead reckoning;
- DIS' dead reckoning;
- dynamic strategies;

To get relevant results, the classifier system approach only involves four effectors corresponding to static PHBDR and DIS' strategies. These strategies are applied in a same simulation (different from the ones presented in previous sections), with exactly the same parameters. Furthermore, to ensure that all models perform in the same conditions, the ghost entity is forced to get at the real model's position periodically. This prevents us from having results biased by the gap introduced by using different kinds of extrapolations.



7: Comparison using the deviation criterion

Figure 7 shows that after 6000 updates, the classifier system performs the best, giving at least a 10% better accuracy than any other strategy. That figure also shows the correlation between the average deviation and the number of updates. Indeed, the curves related to strategies that lead to a smaller deviation require fewer updates for the same number of simulation steps.



8: Comparison using the prediction duration criterion

Figure 8 presents a comparison between the values of the prediction stage durations using those strategies. As well as it is the case with the average deviation, the longer prediction lasts, the fewer updates are sent. Indeed, it can be seen that the curves corresponding to strategies that give higher durations also require fewer updates.

This graph shows that our approach did not find rules that would overcome all the other strategies. Nevertheless, it performed quite well as its performance is equivalent to those of the better other strategies.

Anyway, those two experiments confirm the ability of our system to find performing rule sets in order to fulfill different goals. As it was the case in the previous sections, setting new effectors or changing the system's goals requires fewer work and tends to be less tedious than having to redefine a whole strategy by hand.

6. Conclusions and Future Work

In this paper, we have described a new means of defining prediction model selection strategies for the dead reckoning algorithm. The approach we presented consists in using classifier systems' adaptive capabilities. It is aimed to provide quite a generic system that can generate simulation specific rules. Furthermore, our classifier system-based dead reckoning requires less work from designers when new goals or new criteria have to be taken into account. Indeed, it relieves them of redefining all the rules by hand and of tedious debugging.

The results presented in this paper showed that our system is able to generate relevant rule sets and to adapt to different kinds of motions. This paper also presented two criteria that can be used in order to reduce the number of updates. The experiments performed in order to compare our system to existing ones, showed that the classifier system based dead reckoning is able to provide good results whatever the criterion to optimize is.

In our future work, we expect to extend our system to different kinds of extrapolation techniques. Our aim would be to enable it to provide better results by generating even more simulation-specific rules. Furthermore, we're aiming to take benefit from the αCS classifier systems multi-objective capabilities by asking the system to fulfill combinations of goals.

References

[And05] S. Andrews. A neural network-based approach to a dead reckoning predictor for haptic interfaces, 2005.

[BCL+04] Tom Beigbeder, Rory Coughlan, Corey Lusher, John Plunkett, Emmanuel Agu, and Mark Claypool. The effects of loss and latency on user performance in unreal tournament 2003, 2004.

[CDG+93] James M. Calvin, Alan Dickens, Bob Gaines, Paul Metzger, Dale Miller, and Dan Owen. The simnet virtual world architecture. In *VR*, pages 450–455, 1993.

[CH01] Yim-Pan Chui and Pheng-Ann Heng. Adaptive attitude dead-reckoning by cumulative polynomial extrapolation of quaternions. In *DS-RT '01*, page 45, Washington, DC, USA, 2001. IEEE Computer Society.

[CLC99] Wentong Cai, Francis B. S. Lee, and L. Chen. An auto-adaptive dead reckoning algorithm for distributed interactive simulation. In *PADS '99*, pages 82–89, Washington, DC, USA, 1999. IEEE Computer Society.

[Dee89] S. E. Deering. RFC 1112: Host extensions for IP multicasting, August 1989.

[DFW97] Judith S. Dahmann, Richard Fujimoto, and Richard M. Weatherly. The department of defense high level architecture. In *Winter Simulation Conference*, pages 142–149, 1997.

[Gol85] *Genetic Algorithms and Rules Learning in Dynamic System Control.*, 1985.

[HR78] John H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern-directed Inference Systems*. New York: Academic Press, 1978.

[HY06] Dai Hanawa and Tatsuhiro Yonekura. A proposal of dead reckoning protocol in distributed virtual environment based on the taylor expansion. In *CW*, pages 107–114, 2006.

[LH05] Jeff Long and Michael C. Horsch. A bayesian model to smooth telepointer jitter. In *Canadian Conference on AI*, pages 108–119, 2005.

[Mau00] Martin Mauve. How to keep a dead man from shooting. In *IDMS '00*, pages 199–204, London, UK, 2000. Springer-Verlag.

[Mor96] Katherine L. Morse. Interest management in large-scale distributed simulations. Technical Report ICS-TR-96-27, 1996.

[oISS95] Standards Comitee on Interactive Simulation (SCIS). Ieee standard for distributed interactive simulation - application protocols. Standard 1278.1-1995, IEEE Computer Society, September 1995.

[San01] Cédric Sanza. *Evolution d'entités virtuelles coopératives par système de classifieurs* . Phd thesis, Université Paul Sabatier, Toulouse, France, june 2001.

[SC94] Sandeep K. Singhal and David R. Cheriton. Using a position history-based protocol for distributed object visualization. Technical Report CS-TR-94-1505, 1994.

[SZ99] Sandeep Singhal and Michael Zyda. *Networked virtual environments: design and implementation*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

[TL01] Georgios Theodoropoulos and Brian Logan. An approach to interest management and dynamic load balancing in distributed simulation. In *Proceedings of the 2001 European Simulation Interoperability Workshop (ESIW'01)*, pages 565–571, Harrow, London, UK, June 2001.

[ZGD04] Xiaoyu Zhang, Denis Gracanin, and Thomas P. Duncan. Evaluation of a pre-reckoning algorithm for distributed virtual environments. In *ICPADS '04*, page 445, Washington, DC, USA, 2004. IEEE Computer Society.