An Adaptive Approach for Reactive Actor Design

Daria E. Bergen*
The Naval Research Laboratory
bergen@enews.nrl.navy.mil

James K. Hahn†
The George Washington University
hahn@seas.gwu.edu

Peter Bock†
The George Washington University
pbock@seas.gwu.edu

Abstract

To address the complex and dynamic conditions of a virtual environment, computer animation researchers are applying methods similar to the ones used in artificial life to create reactive actors. A reactive actor is a control entity whose behavior is based on the sensory information it receives from the environment. The system presented within this paper, RAVE (Reactive Actors in Virtual Environments), demonstrates the successful use of a reinforcement learning model to automatically generate controllers for typical 2D navigational tasks. This is an improvement to existing methods because it requires no programming, can be used for a variety of tasks, and the control algorithms adapt during run-time. Collective Learning Systems (CLS) theory is integrated with a hierarchical controller to create control modules that quickly converge on optimal navigational strategies. Five different worlds are created to train and evaluate the actors. Performance metrics and results are presented for three different navigational tasks: obstacle avoidance (*avoid*), heading towards a goal object (*goto*), and moving away from a threat (*retreat*).

CR Categories: G.3 [Probability and Statistics]: Probabilistic Algorithms; I.2.6 [Artificial Intelligence]: Learning; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Animation, Virtual Reality.

Additional Keyword: Behavioral Modeling, Collective Behavior, Virtual Actors, Learning Automaton.

1.0 Introduction

Designing motion control algorithms for characters within virtual environments presents new challenges for computer animation researchers. A virtual environment differs from a classical frame based animation system mainly in its non-deterministic nature. Kinematic methods, such as key framing, are not effective due to the unpredictable movement of the user. Unlike creating a one minute motion sequence for a computer generated scene, the motion generated for a virtual environment must be valid for an unspecified length of time. To address these complex and dynamic conditions, actors should respond to events within the environment as they occur and not simply follow pre-specified scripts.

A *reactive actor* is a control entity that autonomously chooses its behavior based on information it receives from the environment and its internal state. Within the field of computer animation, there has long been an interest in the creation of autonomous actors (Magnenat-Thalmann & Thalmann, 1991; Zeltzer, 1985), and advancements in virtual environment technology makes improvement to methods for virtual actor creation and control both timely and crucial.

The objective of this work is to develop an adaptive control technique to improve the creation and run-time control of reactive actors. The system presented within this paper, RAVE (Reactive Actors in Virtual Environments), demonstrates the successful use of a reinforcement learning model to automatically generate controllers for typical 2D navigational tasks. Collective Learning Systems (CLS) theory is integrated with a hierarchical controller to create control modules that quickly converge on optimal navigational strategies. This model can also be used for adaptation, during run-time, to cope with changing environment conditions. In reactive actor design, adaptation has previously been explored only as a pre-processing step. This work demonstrates why learning, during run-time, is a useful and necessary component.

2.0 Problem Domain

Traditionally, a combination of hierarchical control and procedural methods have been used in reactive actor design (Blumberg & Galyean, 1995; Perlin & Goldberg, 1996; Tu & Terzopoulos, 1994; Zeltzer, 1985). Hierarchical approaches are chosen because they lead to reusable, extendible, and responsive control models. A generic hierarchical model is presented in Figure 1. The left column lists the associated level of control abstraction (Zeltzer, 1985), and the right enumerates the input to each subsystem. In brief, sensors receive world information and possibly task level commands from the animator. The reasoning engine interprets the sensory information and selects an appropriate task from the actor's motion repertoire. The functional control units invoke procedural control units to achieve their goals. Finally, the procedural control units update effectors which cause some attribute(s) of the actor to change.

Researchers in the fields of Artificial Life (Langton, 1994) and Adaptive Behavior (Maes, 1992) are also interested in designing autonomous beings. There are many similarities between the design of autonomous agents and the design of autonomous actors, particularly in the low-level architectural design. At a high level, however, the focus is less similar. Within the field of computer animation, the essence is the story. Animators are interested in generating characters that are expressive and have consistent personalities. While a certain level of autonomy is desired, the animator must also be able to direct the actor at various levels and during different stages of design.

Given this as the high level research goal, one can identify at least six major research areas in reactive actor design. Finding the balance between *autonomy* and *directability* is the first essential component. Questions concerning when, where, and how much autonomy is necessary and when, where, and how an animator will direct an autonomous actor remain to be answered (Blumberg & Galyean, 1995). Methods for creating *expressive motion*

* The Naval Research Laboratory, 4555 Overlook Ave. SW, Code 5707, Washington, DC 20375
† The George Washington University, Department of EE&CS, 801 22nd Street NW, Washington, DC 20052

(Perlin, 1995) are still being explored. The actor should exhibit a consistent personality and the animator should be able to influence its personality directly and indirectly. *Task decomposition* is another important area. Determining exactly how an animator breaks down a complex task into primitive modules remains to be solved. *Behavior arbitration* involves combining the primitive tasks to create complex behavior. Issues in behavior selection, smooth transitioning between tasks, and methods for avoiding "dithering" (Blumberg, 1994) must be addressed. Finally, the exact roles of *learning and evolutionary methods* within the design of the control units must be identified. Are they mutually exclusive design decisions, or can they compliment each other in the creation and control phases?

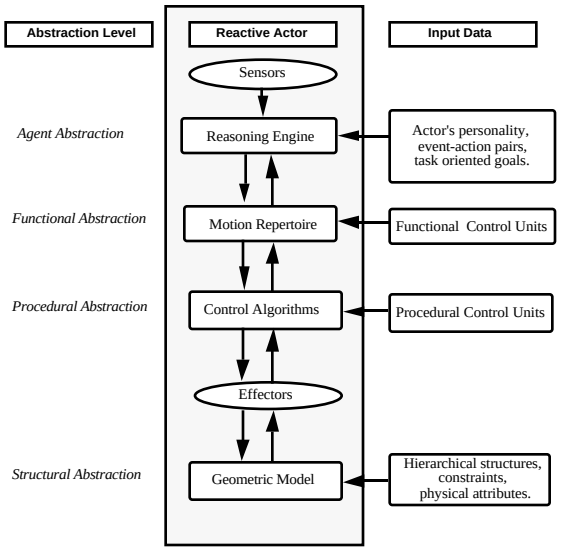


Figure 1: Hierarchical Control Model

A subset of these research problems is addressed within this work. A bottom-up approach is taken, and the individual control units within the actor's motion repertoire are the focus of this paper. Specifically, the problem of designing and controlling a motion repertoire for obstacle avoidance and 2D navigation is investigated. Issues concerning the evaluation of motion characteristics and the role of learning in control module creation are examined.

3.0 Related Work

Motion repertoires for navigational tasks have been defined procedurally with standard program languages (Reynolds, 1987; Sun & Green, 1993; Tu & Terzopoulos, 1994) and with dataflow networks (Wilhelms & Skinner, 1990). Programming expertise is required when a procedural method is used and creating algorithmic descriptions for each primitive task can be tedious and time consuming. Reynolds's seminal work in behavioral modeling (Reynolds, 1987) was initially presented as a means of reducing the tedium associated with scripting the paths of many actors. This general model has been modified slightly over time but remains the standard method for obstacle avoidance and low-level navigation.

Evolutionary techniques and standard search methods have been used to automatically generate task level modules for articulated figure motion (Gritz & Hahn, 1995; Ngo & Marks, 1993; Sims, 1994; van de Panne & Fuime, 1993). The results achieved are impressive and the motion generated with these models is believable and enjoyable to watch. However, the adaptation of the control modules has strictly been applied as a preprocessing step. Researchers have not explored issues related to run-time adaptation. Evolutionary methods are useful for globally exploring large search spaces but not for a localize search. Since animators also desire a variety in behavior, evolutionary techniques are ideal for exploring the entire controller space and finding a specific style of motion. They fail to offer assistance, however, when minor modifications to a control algorithm are needed. Alternative methods must be used when incremental improvements or run-time adaptation is desired.

The system presented within this paper uses an adaptive machine learning methodology to automatically generate functional control units for typical 2D navigational tasks. This is an enhancement to existing methods used for navigation because it requires no programming, can be used for a variety of tasks, and the control algorithms adapt during run-time. The method has been used for obstacle avoidance, movement towards a goal, and movement away from a threat.

4.0 RAVE Architecture

An overview of the RAVE architecture is presented in Figure 2. As with other reactive systems, sensory information is extracted from the world, passed to the actor, and used to determine the actor's behavior. The uniqueness of RAVE lies in the design of the motion critic sub-system and the addition of an adaptive learning module.

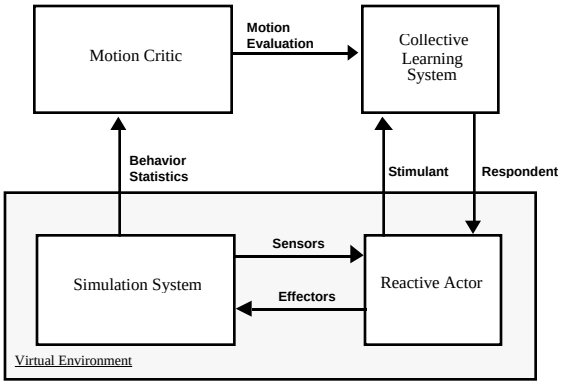


Figure 2: RAVE Architecture

4.1 Sensors and Effectors

Within the literature, two alternative approaches to synthetic vision have been explored; object (or symbolic) based (Reynolds, 1988) and image based (Horswill, 1992) vision models have both been used. RAVE uses a symbolic vision sensor. This was chosen over an image based approach for a number of reasons. One advantage is that exact information can be conveyed to the actor; algorithms for determining object type, velocity, distance, and color are not necessary. Computing symbolic features also requires little additional overhead since many object attributes are already kept in a database for rendering purposes. Finally, a symbolic approach can be easily extended. Additional features can be incorporated quickly by appending attributes to the sensory input stream.

Within this work, three vision channels provide information about the closest object in the left, middle, and right field of view (FOV); object type, distance, and direction information are provided. Parameters for the vision module

include the angle for each field of view triangle (α, β, and ψ) and the length of the vision range (v). See Figure 3.

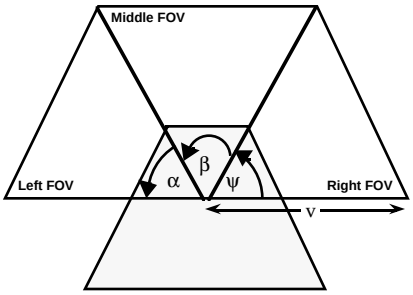


Figure 3: Vision Sensor

Moving towards a goal or away from a threat requires additional information. The bearing sensor is used to receive azimuth and distance information about a goal or threat object. The user can vary the desired range (d) and the sensing range (r). The desired range will determine how close to a goal or how far from a threat the actor should be. The sensing range determines at what range the actor can differentiate distances. For example, the bearing sensor in Figure 4 would return a value of "NW and beyond sensing range".

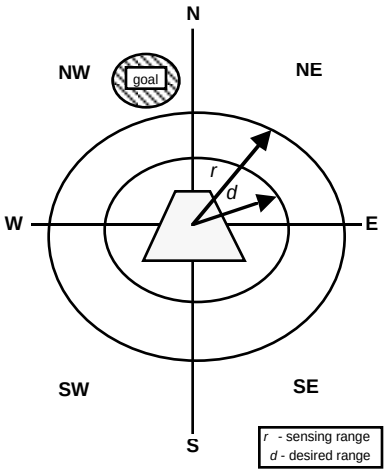


Figure 4: Bearing Sensor

An actor is represented with a position vector (**p**) and an azimuth (θ). Each actor travels at a constant speed in the direction of its heading vector (**h**). The effector modifies the heading vector by requesting a change in θ. This turns the actor to the left or to the right.

4.2 Motion Critic

The motion critic receives behavioral statistics from the simulation system and assesses the quality of the motion. The criteria used to evaluate the motion may remain constant or vary over time. The assessment is sent to the Collective Learning Systems (CLS) as an evaluation.

Computing an evaluation requires assigning a quantitative measurement to the motion sequence. This is not always straightforward task. For example, one valid measurement for obstacle avoidance would be to count the number of collisions that occur. The actor would receive a positive evaluation (reward) for a small number of collisions and a negative evaluation (punishment) for a large number of collisions. Using this as the only criterion, however, could create an actor that moves in a circular pattern. This may or may not be the behavior the designer had in mind. An evaluation that combines the number of collisions and the distance traveled would create different results. Examples of various evaluation policies are presented within §5.5. Varying the evaluation policy is one method of controlling the resulting motion.

4.3 Collective Learning System

Collective Learning System Theory (Bock, 1993) is an adaptive machine learning methodology inspired by learning automata theory (Narendra & Thathachar, 1974; Samuel, 1959; Tsetlin, 1962) and related to classical automata theory and statistical analysis. A collective learning automaton (CLA) is a finite state machine that modifies its internal structure, learning the appropriate state transitions, as a result of its interaction with the environment. An overview of a collective learning automaton is presented in Figure 5.

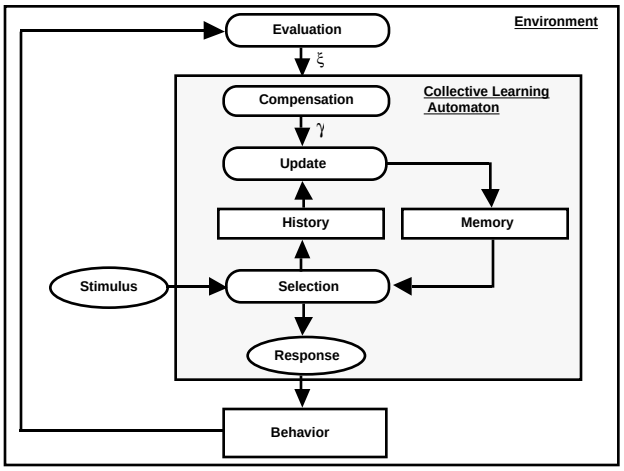


Figure 5: Collective Learning Automaton

The CLA receives evaluations from the environment and adjusts its behavior as a function of the rewards and punishments it receives. The environment does not evaluate the CLA after every interaction; a sequence of stimulus-response pairs are saved in a history structure (η) and evaluated together at the end of a stage. The length of a stage is determined by a system parameter, collection length (l). The evaluation (ξ) is received by the compensation function which provides a method for the CLA to adjusts its view of the evaluation. The compensation function converts ξ into a compensated value (γ) that is used by the update process. The update process uses γ to either increase or decrease the probability of a stimulus-response pair from happening again.

The selection process occurs at every timestep. The stimulus received from the environment is used to access the CLA's state transition matrix (Figure 6). This matrix consists of column vectors of all possible stimuli $<\phi_1, \phi_2, \ldots, \phi_n>$ and associated with each stimulus is a tuple of valid responses $<\omega_1, \omega_2, \ldots, \omega_m>$. A weight ($w_{ij}$) is stored with each stimulus-response pair. This weight is used by the selection process to determine the approach response for a given stimulus.

CLS theory was chosen for this application because it converges quickly, and updates to memory can be performed in real-time. This model also gives the animator the ability to make small modifications to the behavior of the actor; he can control the motion by modifying the evaluation function or by adjusting the learning parameters. Finally, another appealing aspect of this model is that once the controller has converged, rules which were learned can be extracted if desired.

Stimulus (Φ)

Response (Ω)	ϕ_1	ϕ_2	$\cdots$	ϕ_n
ω_1	w_{11}	w_{12}	$\cdots$	w_{1n}
ω_2	w_{21}	w_{22}	$\cdots$	w_{2n}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
ω_m	w_{m1}	w_{m2}	$\cdots$	w_{mn}

Figure 6: State Transition Matrix (STM)

4.4 Run-time Loop

At every timestep, the objects within the actor's vision field are determined and used to select a response. The stimuli-response pair is saved in a history file and processed at the end of the stage. The response is used to update the actor's position and heading. Collision detection is performed at the new location to ensure the move is valid. If the response results in a collision, forward motion is prevented, and the actor is placed at its previous location. At the end of the stage, the motion sequence is evaluated, compensated, and subsequently used to update the actor's STM. The following is pseudo-code of the run-time loop for a learning reactive actor:

```
while simulation is running do
    for k = 1 to collection length do
        draw world;
        φi = process input stimuli;
        ωj = select response(φi);
        save stimuli-response(φi,ωj,ηk);
        p' = update position(ωj);
        check for collisions;
        if (a collision occurred) p' = p;
        calculate behavior statistics;
    endfor
    ξ = evaluate motion for stage;
    γ = compensate evaluation(ξ);
    update STM (γ,η);
endwhile
```

4.5 Performance Metrics

In order to measure the performance of the control modules, a number of metrics were designed to monitor the actor's ability to learn. These metrics are similar to the evaluation functions used by the motion critic. They are scaled to the range of [0,100] with 100 representing perfect performance. The following "scores" are used to verify RAVE's ability to automatically generate three navigational tasks: obstacle avoidance (*avoid*), heading towards a goal object (*goto*), and moving away from a threat (*retreat*).

During the run of each experiment, the statistics collected depend on the navigational task being learned. For the *avoid* task, the minimize-collisions (S_{minc}) and the maximize-distance (S_{maxd}) scores are collected. For the *goto* task, the time-in-goal-region score (S_{in}) determines how long the actor stays within the desired goal region. Conversely, the time-outside-threat-region score (S_{out}) indicates how well the actor is staying away from a threat. The experiments are subdivided into contests which last for 1800 timesteps (see §6.0). The scores for each contest are computed as follows:

$$S_{minc} = 100\left[1 - \frac{C_c}{C_{cmax}}\right]$$

$$S_{maxd} = 100\left[\frac{D_c}{D_{cmax}}\right]$$

$$S_{in} = 100\left[\frac{T_{in}}{TS_c - TT}\right]$$

$$S_{out} = 100 - S_{in}$$

where C_C is the number of collisions that occurred during the contest; C_{Cmax} is the maximum number of collisions that can occur; D_C is the distance traveled during the contest; D_{Cmax} is the maximum distance an actor can travel during a contest; T_{in} is the number of timesteps the actor stays within the desired range; TS_C is the number of timesteps per contest; and TT is the travel time, the minimum time needed to travel to a goal or away from a threat region. Travel time is incorporated into the equation to account for the actors being placed at random locations at the beginning of each contest.

5.0 Technical Approach

Five different worlds were created for training the actors. Each world is 100x100 units and is bounded by walls. The four worlds used to train the actors for the *avoid* task are shown in Figure 7.

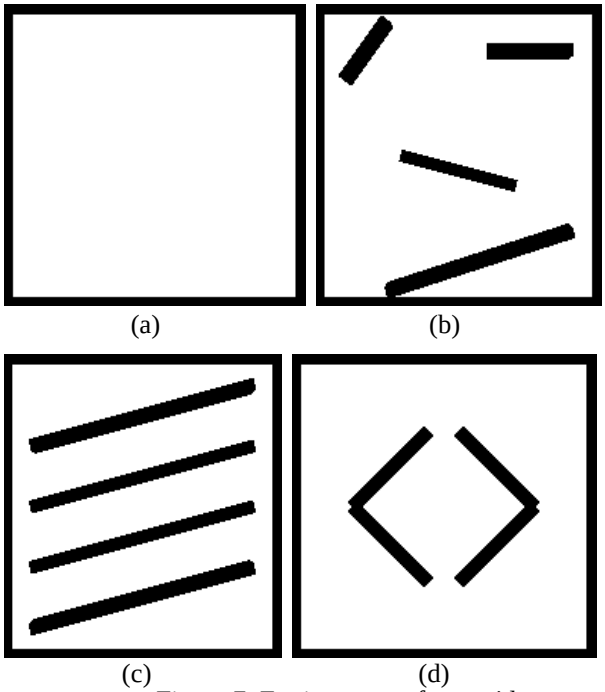


Figure 7: Environments for *avoid*
(a) Open Field, (b) Hidden Corners,
(c) Racing Lanes, (d) Courtyard

The Open Field world contains no obstacles within its boundaries. The Hidden Corners, Racing Lanes, and

Courtyard worlds have four internal wall obstacles. The angles and spacing between the walls were varied to create worlds with different characteristics. The Hidden Corners world contains wall obstacles oriented at angles of 53˚, 0˚, 14˚, and 17˚. Two dead-end passageways were created to observe how actors handled this situation. The Racing Lanes world was created to see whether a world with symmetry was easier to navigate. Each wall within this world is oriented at 14˚. Finally, the Courtyard world was created to test the actor's ability to move through openings between walls. The walls are oriented at 45˚ angles, and the openings are 7 units wide.

The *goto* and *retreat* tasks used the same world (Figure 8). Five goals or threats were placed in the world, and the target object was changed every 5 contests. The hemispheres represent possible goals or threats within the world. Only one object was active at any given time. The desired range to stay next to the goal or away from the threat is depicted as a circle surround the object.

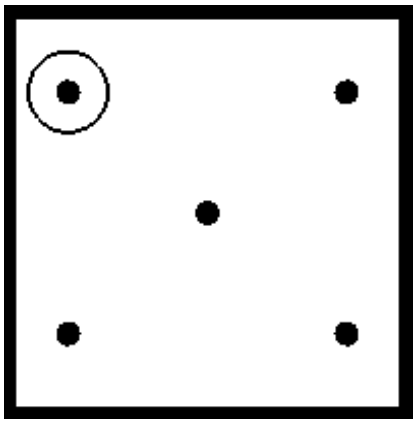


Figure 8: Environment for goto & retreat

5.1 Sensor Implementation

The actors are one unit long, and each has a total vision field of 180˚. The left, middle, and right FOVs were set to equal angles of 60˚. For the *avoid* task, the vision range was 20 units; and for the *goto* and *retreat* tasks it was 10 units in length. Initial results were obtained with the actors traveling at a constant speed of 10 units per second.

In order to determine which objects are within the actor's FOV, objects are first culled against the bounding rectangle of the actor's entire vision field. Axis-aligned bounding boxes are computed for all objects and used for an initial overlap test. Exact intersection testing is needed only if these areas overlap. Since an interactive update rate is necessary, an exact edge-to-edge intersection test with the object's polygons is not performed; geometric footprints are used to simplify the calculation. Footprints are created by projecting the bounding rectangle onto the ground plane. The edges of the footprint are then checked against the field of view triangle.

5.2 Stimulus-Response

At the beginning of each timestep a stimulant key is created to access either the vision or bearing STM. The vision sensor returns which objects, if any, are within the actor's left, middle, and right FOV. The object type and distance from the actor are used to create an index to the actor's vision STM. Objects within the world are classified into six categories; a 3-bit identifier is used as an object ID. The distance to the object is quantized into four values: very close, close, far, and very far. Distance is encoded with 2-bits. The object and distance encoding are listed in Table 1.

Object	Space (SPA)	000
	Obstacle Static (OBS)	001
	Obstacle Moving Towards (OBT)	010
	Obstacle Moving Away (OBA)	011
	Goal Static (GOS)	100
	Threat Static (THS)	101
Distance	Very Close	00
	Close	01
	Far	10
	Very Far	11

Table 1: Object and Distance Encoding

The distance range is quantized in a linear manner and the distance bits (dbits) are set as follows:

$$dbits = \begin{cases} 00 \text{ if } distance \le \frac{1}{4}v \\ 01 \text{ if } \frac{1}{4}v < distance \le \frac{1}{2}v \\ 10 \text{ if } \frac{1}{2}v < distance \le \frac{3}{4}v \\ 11 \text{ otherwise} \end{cases}$$

where *distance* is the distance from the actor to the object and v is the length of the vision range. The object ID and distance code for the left, middle, and right FOV are concatenated to form a 15-bits key. The stimulant is used to retrieve the respondent tuple from the vision STM (Figure 9).

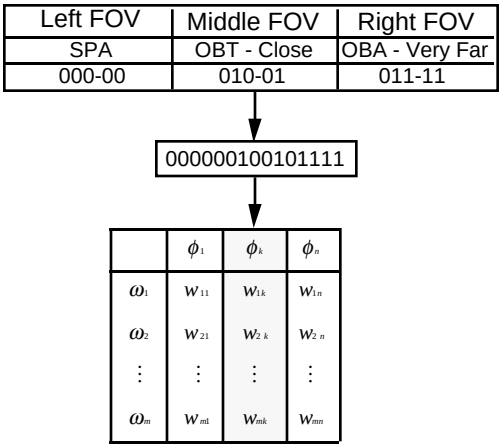


Figure 9: Sensors to STM Index

The stimulant for the bearing STM is calculated in a similar manner. The bearing sensor returns the azimuth and distance from the goal or threat object. The azimuth is quantized into eight values: N, NW, W, SW, S, SE, E, and NE. This value is encoded with a 4-bit identifier. The distance from the object is again quantized and encoded with 2-bits. The distance encoding is different for the bearing sensor:

$$dbits = \begin{cases} 00 \text{ if } distance \le d \\ 01 \text{ if } d < distance \le \dfrac{d+r}{2} \\ 10 \text{ if } \dfrac{d+r}{2} < distance \le r \\ 11 \text{ otherwise} \end{cases}$$

where *distance* is the distance from the actor to the goal or threat; *d* is the desire range; and *r* is the sensing range. The azimuth and distance code are concatenated to form a 6-bit stimulant key.

The response for both STMs consists of a heading adjustment in the range of [+45˚, -45˚]. For the result presented within this paper the response range, Φ = {+45˚, +30˚, +20˚, +10˚, 0˚, -10˚, -20˚ -30˚, -45˚}.

5.3 Evaluation and Compensation

At the end of every stage, actors receive an evaluation, ξ, in the range of [-1,1]. The actors are evaluated when a collision occurs and after every 20 timesteps. With this parameter setting, actors receive an evaluation approximately every second, assuming 20 frames per second (fps).

To train the actors for the *avoid* task, and to create behavior which minimizes the number of collisions, the evaluation ξ_{minc} is computed as follows:

$$\xi_{minc} = \begin{cases} \dfrac{TS_S}{l} \text{ if } C_S = 0 \\ -1 \text{ otherwise} \end{cases}$$

where TS_S is the number of timesteps without a collision and *l* is the collection length. As mentioned previously, one way of avoiding collisions is to travel in a circle. This is usually not the desired behavior, so an evaluation function was created to encourage straight movement. The evaluation function used to maximize distance traveled, (ξ_{maxd}), is computed as follows:

$$\xi_{maxd} = \begin{cases} \dfrac{D_S}{D_{Smax}} \text{ if } D_S \ge \dfrac{3}{4} D_{Smax} \\ -\dfrac{D_{Smax} - D_S}{D_{Smax}} \text{ otherwise} \end{cases}$$

where D_S is the distance traveled during the stage and D_{Smax} is the maximum distance the actor can travel, in a straight line, during the stage.

A number of policies can be used to compute the final evaluation (ξ). Informal experiments where the average of ξ_{minc} and ξ_{maxd} was used, lead to poor performance. Stages where the actor avoided all obstacles were not rewarded appropriately when the distance criteria was not met. Conversely, the actor was rewarded for mediocre collision avoidance when it was able to travel a great distance, while bumping into things. A dynamic evaluation function was ultimately used; the actor was trained to first avoid obstacles and then encouraged to maximize its distance traveled. The final evaluation function used for the *avoid* task is:

$$\xi = \begin{cases} \xi_{minc} \text{ if } \xi_{minc} < 1 \\ \xi_{maxd} \text{ otherwise} \end{cases}$$

For the *goto* task, in order to move towards a goal region, the actor must minimize its distance from the goal object. The evaluation function used to decrease the distance from the goal (ξ_{dd},) is:

$$\xi_{dd} = \begin{cases} \dfrac{\Delta D_{towards}}{D_{Smax}} \text{ if } D_{se} > r \\ 1 \text{ otherwise} \end{cases}$$

where $\Delta D_{towards} = D_{SS} - D_{se}$;; D_{SS} is the distance from the goal at the start of the stage; and D_{se} is the distance from the goal at the end of the stage. For this task, the actor was trained to first avoid obstacles and then to decrease its distance from the goal until it was inside the goal region. Once inside the goal region, the actor should minimize its change in distance from the goal. This evaluation ($\xi_{min\Delta d}$) encourages the actor to circle the goal:

$$\xi_{min\Delta d} = 1 - 2\left[\frac{\Delta D}{D_{Smax}}\right]$$

where $\Delta D = |D_{SS} - D_{se}.|$. The final evaluation function used for the *goto* task is:

$$\xi = \begin{cases} \xi_{minc} \text{ if } \xi_{minc} < 1 \\ \xi_{dd} \text{ if } D_{se} \ge r \\ \xi_{min\Delta d} \text{ if } \xi_{min\Delta d} < 1 \\ \xi_{maxd} \text{ otherwise} \end{cases}$$

Finally, for the *retreat* task, the objective is to train the actors to stay away from the threatening object. The evaluation for this task (ξ_{id},) encourages the actor to increase the distance between itself and the threat object.

$$\xi_{id} = \begin{cases} \dfrac{\Delta D_{away}}{D_{Smax}} \text{ if } D_{se} < r \\ 1 \text{ otherwise} \end{cases}$$

where $\Delta D_{away} = D_{se} - D_{SS}$. The dynamic evaluation function used for *retreat* trains the actor to first avoid collisions, then maximize its distance from the threat, and once outside the threat region, maximize its distance traveled. The final evaluation function used for the *goto* task is as follows:

$$\xi = \begin{cases} \xi_{minc} \text{ if } \xi_{minc} < 1 \\ \xi_{id} \text{ if } D_{se} < r \\ \xi_{maxd} \text{ otherwise} \end{cases}$$

For all initial experiments $\gamma = \xi$; evaluations were used directly in the update function. Investigating the affects of the compensation function on the "personality" of the actor is an area that will be explored in future work.

5.4 Update Function

The compensation value (γ) is applied to the weights of the stimulus-response pairs within the history structure (η). The update function increases or decreases the strength

of the stimulus-response pair in the actor's vision or bearing STM. Again, a number a policies can be used. An elementary update function simply adds γ to the initial weight:

$$w'_{ij} = \max(\gamma + w_{ij}, w_c) \text{ if } \gamma \geq 0$$

$$w'_{ij} = \min(\gamma + w_{ij}, w_f) \text{ otherwise}$$

where w_{ij} is the weight associated with stimulus i and response j; w_c is the weight ceiling; and w_f is the weight floor. This update policy works well when the environment is static or changes very little. During informal experimentation, it was found that this method is not appropriate for this application. The actor could learn a sub-optimal navigational strategies early in the match, and when it was in a region of the world where the strategy was not appropriate (i.e. a sharper turn was necessary), it would need to "unlearned" erroneous information. Discovery and extinction factors are used within the update function to adjust for situations like this. When using the discovery and extinction values, the increase or decrease in weight is proportional to the distance from the weight floor or ceiling. With this policy, sub-optimal strategies quickly lose their strength and new responses will be tried. The update policy used for this work is as follows:

$$w'_{ij} = \delta(w_c - w_{ij})(\gamma + w_{ij}) \text{ if } \gamma \geq 0$$

$$w'_{ij} = \varepsilon(w_{ij} - w_f)(\gamma + w_{ij}) \text{ otherwise}$$

where δ is the discovery factor and ε is the extinction factor.

5.5 Selection Function

The selection policy used is MTDS/RA [Maximum Thresholded Deterministic Selection with Random Arbitration] (Bock, 1993). The bounded weights within the respondent tuple are mapped to pseudo-probabilities and the respondent with the largest probability, within a threshold value, is selected. If two or more respondents fall within the selection range, a respondent within the range is randomly picked.

5.6 Behavior Arbitration

For the *goto* and *retreat* tasks, a heuristic is used to determine how to alternate between collision avoidance using the vision STM and directed movement using the bearing STM. If an object is within the actor's vision field, a pending collision is assumed, and the vision STM is used to circumvent the collision. If no pending collisions are identified, the bearing STM is used to provide the actor with information concerning the goal or threat object. Table 2 provides a summary of the various environment parameters used in this work.

6.0 Results

A total of twelve experiments were run to test the performance of RAVE. Scores (§4.5) were computed at the end of every contest and a total of 50 contests were executed for each experiment. To provide them with an opportunity to learn a variety of navigational strategies, the actors were randomly placed at a new position and heading at the beginning of every contest.

Size of World	100x100 units
Number of Actors	1 to 5
Number of Wall Obstacles	4 to 8
Number of [Goals | Threats]	5
Switch [Goal | Threat]	after 5 contests
Speed of Actor	10 units per sec
Vision Range (v)	10 or 20 units
Left FOV Angle (α)	60˚
Right FOV Angle (β)	60˚
Middle FOV Angle (ψ)	60˚
Sensing Range (r)	30 units
Desired Range (d)	10 units

Table 2: Environment Conditions

To validate the *avoid* task, actors were trained in the four worlds described in §5.0. One set of experiments was run with only one actor and another set with a total of five actors in the world. All actors were adaptive, and each had its own memory structure. Figure 10 demonstrates typical navigational strategies which were learned; the path the actor took through the environment is traced, and the S represents the start of the path. Actors were able to converge to an optimal strategy quickly, and the graphs in Figure 12 demonstrate near perfect collision avoidance occurring as early as contest number 5 (after 9000 timesteps). It took a longer period of time to learn to navigate the worlds with internal wall obstacles. The periodic decline in performance, shown in Figure 12b, represents contests where the actor was placed at a new location and needed to learn how to navigate that particular area of the world.

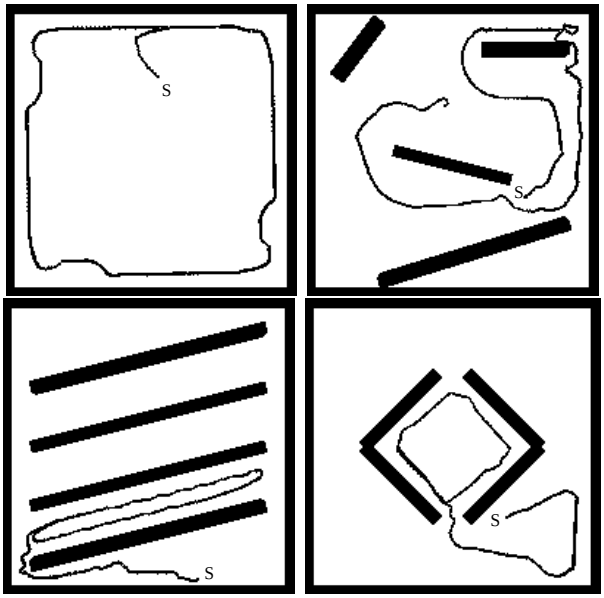


Figure 10: Obstacle Avoidance Behavior

For the *goto* task, one world was used. Again, one experiment was run with only one actor and another with five actors in total. Figure 11a provides an example of a typical goto strategy which was learned. The graphs in Figure 13 plots the performance metrics for this task. Figure 13a was an experiment performed with only one actor within the world, and Figure 13b demonstrates the results when five actors were within the world. Remaining within the desired range to the goal is more difficult when other actors are present. Collision avoidance takes precedence over staying within the desired range, and the lower scores in the S_{in} score

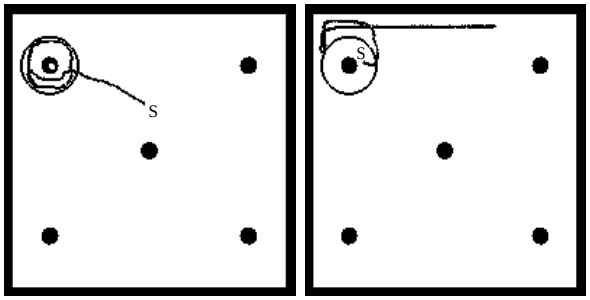
represent times when the actor needed to stay outside the region to avoid a collision.

The world used to verify the *retreat* task was similar, except that the goal objects are now treated as threats. Again, one experiment was run with only one actor and another with five actors. A screen shot of a path taken when retreating from a threat is shown in Figure 11b, and the performance metrics are presented in Figure 14. Again, the actors were trained to avoid collisions, leave the threatening area, and once outside the range of the threat, maximum their distance traveled.

Stage	20 Timestep
Contest	90 Stages
Match	50 Contests

Table 3: Experiment Parameters

Experiments were run on a SGI Indigo^2 Extreme with a 150Mhz R4000 MIPS RISC processor and 64MB of RAM. RAVE can be run interactively or in batch mode. When run interactively, with a 200x200 display window, 72 fps can be achieved while the actors are learning. A contest takes approximately 25 seconds, and a match can be completed within 21 minutes. In batch mode, a contest takes 5 seconds, and a match is finished in 4 minutes.



(a) (b)

Figure 11: Examples of (a) *goto* and (b) *retreat*

7.0 Conclusions

This adaptive methodology for reactive actor design and control can be used to aid in the building and controlling of reactive actors. The robustness of functional control units is improved when an adaptive approach is used, because the actors can modify their behavior during run-time. The usefulness of this approach was demonstrated with three different navigational tasks: *avoid*, *goto*, and *retreat*. Evaluation functions for assessing the quality of the motion and performance metrics for monitoring the actor's motions were designed.

8.0 Future Work

RAVE will be extended to include additional tasks within the motion repertoire. It is hoped that a comprehensive set of navigational tasks can be designed so that all types of navigational movement can be easily created. Additional experiments will be performed to explore the robustness of RAVE. The design of a complexity metric which assigns a numeric value to the difficulty of navigating through a specific virtual world would provide a quantitative measurement for comparing reactive models. Finally, additional experiments are needed to determine how various evaluation functions and learning parameters affect the characteristics of the resulting motion.

Acknowledgments

This work was funded in part by the Tactical Electronic Warfare Division (TEWD) of the Naval Research Laboratory and by the Office of Naval Research.

References

Blumberg, B. (1994). Action-Selection in Hamsterdam: Lessons from Ethology. In J.-A. Meyer, H. L. Roitblat, & S. Wilson W. (Eds.), From Animals to Animats: Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB94) Cambridge, MA: MIT Press.

Blumberg, B. M., & Galyean, T. A. (1995). Multi-Level Direction of Autonomous Creatures for Real-Time Virtual Environments. SIGGRAPH'95, 47-54.

Bock, P. (1993). The Emergence of Artificial Cognition: An Introduction to Collective Learning. River Edge, NJ: World Scientific.

Gritz, L., & Hahn, J. K. (1995). Genetic Programming for Articulated Figure Motion. Journal of Visualization and Computer Animation, 6, 129-142.

Horswill, I. (1992). A simple, cheap, and robust visual navigation system. In J. A. Meyer, H. L. Roitblat, & S. Wilson W. (Eds.), From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB92) (pp. 129-136). Cambridge, MA: MIT Press.

Langton, C. G. (Ed.). (1994). Artificial Life IV: Proceedings of the Workshop on Artificial Life. Cambridge, MA: MIT Press.

Maes, P. (1992). Behavior-Based Artificial Intelligence. In J. A. Meyer, H. L. Roitblat, & S. W. Wilson (Eds.), From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB92) (pp. 2-10). Cambridge, MA: MIT Press.

Magnenat-Thalmann, N., & Thalmann, D. (1991). Synthetic Actors in Computer-Generated 3D Films. New York: Springer-Verlag.

Narendra, K. S., & Thathachar, M. A. L. (1974). Learning Automata - A Survey. IEEE Transations on Systems, Man, and Cybernetics, 14, 323-334.

Ngo, J. T., & Marks, J. (1993). Spacetime Constraints Revisited. SIGGRAPH'93, 343-350.

Perlin, K. (1995). Real Time Responsive Animation with Personality. IEEE Transactions on Visualization and Computer Graphics, 1(1).

Perlin, K., & Goldberg, A. (1996). Improv: A System for Scripting Interactive Actors in Virtual Worlds. SIGGRAPH96, 205-216.

Reynolds, C. W. (1987). Flocks, Herds, and Schools: A Distributed Behavioral Model. SIGGRAPH'87, 25-34.

Reynolds, C. W. (1988). Not Bumping Into Things. In Physically Based Modeling Coursenotes Atlanta, Georgia: SIGGRAPH'88.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. IBM Journal of Research and Development, 3, 211-229.

Sims, K. (1994). Evolving Virtual Creatures. SIGGRAPH'94, 15-22.

Sun, H., & Green, M. (1993). The Use of Relations for Motion Control in an Environment With Multiple Moving Objects. Graphics Interface'93, 209-218.

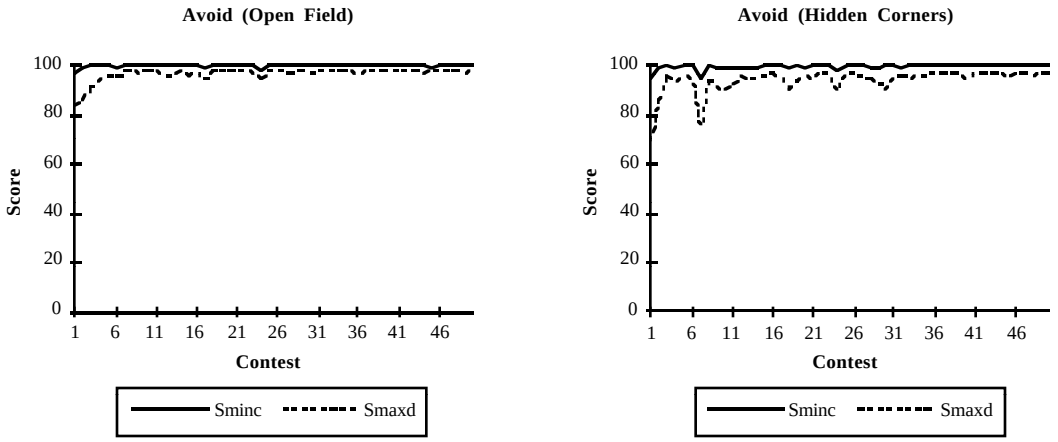
Tsetlin, M. L. (1962). On the Behavior of Finite Automata in Random Media. Automation and Remote Control, 22, 1210-1219.

Tu, X., & Terzopoulos, D. (1994). Artificial Fishes: Physics, Locomotion, Perception, Behavior. SIGGRAPH'94, 43-50.

van de Panne, M., & Fuime, E. (1993). Sensor-Actuator Networks. SIGGRAPH'93, 335-342.

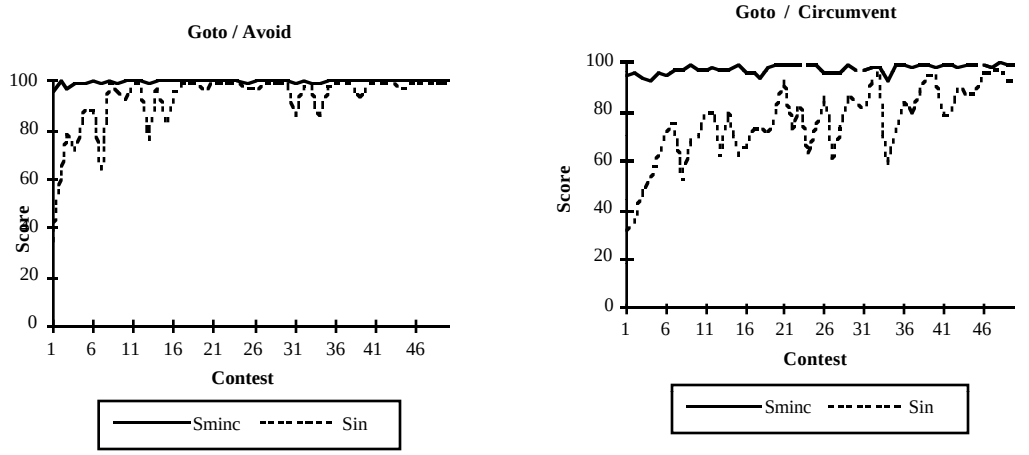
Wilhelms, J., & Skinner, R. (1990). A "Notion" for Interactive Behavioral Animation Control. IEEE Computer Graphics and Applications, May 1990, 14-22.

Zeltzer, D. (1985). Towards an integrated view of 3-D computer animation. The Visual Computer, 1(4), 249-259.



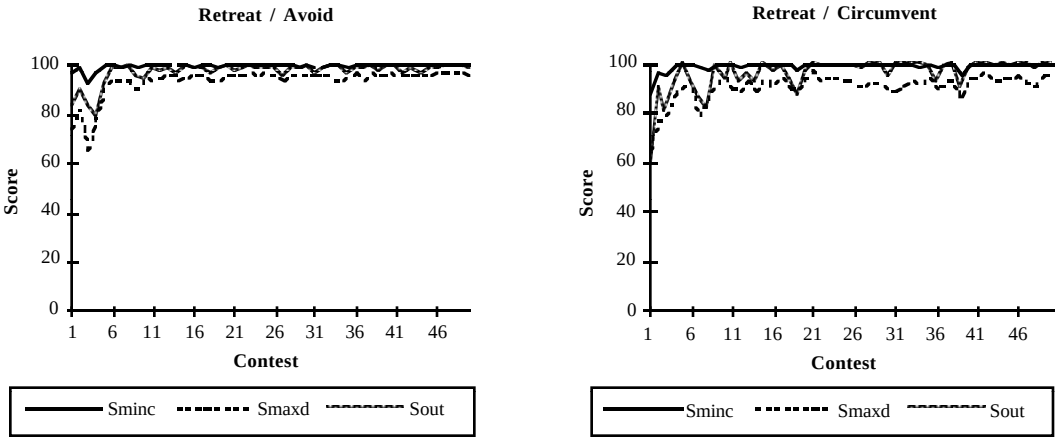
(a) (b)

Figure 12: *Avoid* Performance Metrics



(a) (b)

Figure 13: *Goto* Performance Metrics



(a) (b)

Figure 14: *Retreat* Performance Metrics