INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

A Fuzzy-Logic Autonomous Agent, Applied as a Supervisory Controller in a Simulated Environment

Georgios Chrysanthakopoulos

A dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor Of Philosophy

University Of Washington

2000

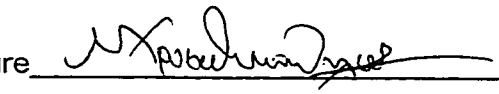
Program Authorized to Offer Degree: Electrical Engineering

UMI Number: 9995354

UMI Microform 9995354

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

In presenting this dissertation in partial fulfillment of the requirements for the Doctoral degree at the University Of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of the dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to Bell and Howell Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature 

Date 10/25/2006

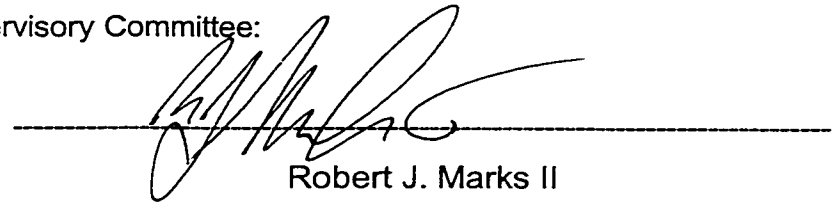
University Of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Georgios Chrysanthakopoulos

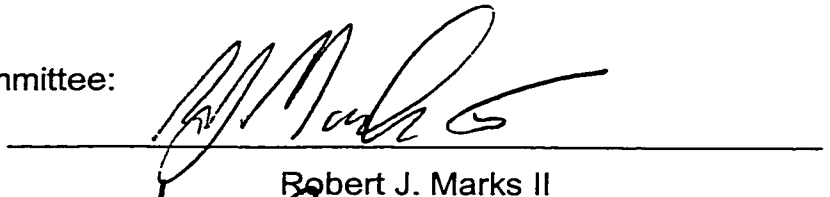
and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of Supervisory Committee:

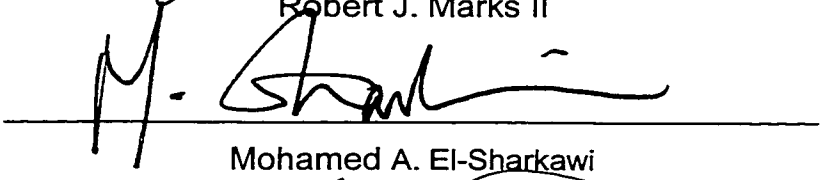


Robert J. Marks II

Reading Committee:



Robert J. Marks II



Mohamed A. El-Sharkawi

Michael Healy

Date: 10/25/2000

University Of Washington

Abstract

A Fuzzy-Logic Autonomous Agent, Applied as a Supervisory Controller in a Simulated Environment

Georgios Chrysanthakopoulos

Chairperson of the Supervisory Committee:
Professor Robert. J. Marks II
Electrical Engineering

An unsupervised learning system, implemented as an autonomous agent is presented. A simulation of a challenging path-planning problem is used to illustrate the agent design and demonstrate its problem solving ability. The agent, dubbed the ORG, employs fuzzy logic and clustering techniques to efficiently represent and retrieve knowledge and uses innovative sensor modeling and attention focus to process a large number of discrete stimuli. Simple initial rules are used to influence behavior and communicate intent to the agent. Self-reflection is utilized so the agent can learn from its environmental constraints and modify its own state. Speculation is utilized in the simulated environment, to produce new rules and fine-tune performance and internal parameters. Several resulting paths of the agent design are shown, and desirable side effects of the agent design are discussed.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Acknowledgements

The author would like to thank his advisor, Professor Marks for the invaluable support he provided all these years. He has been a great mentor and even if his sense of humor is still not completely understood, it is appreciated and with enough time the jokes will start making sense. The author would also like to thank Warren Fox for the time he dedicated in reviewing the draft documents and the rest of the team at the Applied Physics Lab for supplying a challenging problem for the ORG to tackle and great feedback during our weekly meetings. I would also like to thank Professors El-Sharkawi and Professor Healy for their feedback and guidance from the very beginning of my graduate study. Last but not least the author would like to thank his wife for providing endless motivation and encouragement (and for driving him to UW when he was too tired to take the bus).

I. Introduction

An unsupervised problem solver is presented, implemented as an autonomous agent. The agent is a software algorithm with an advanced sensory interface, learning facility, speculation abilities and self-reflection capabilities. It is applied to a path-planning problem in a simulated environment. The agent algorithm, dubbed the ORG (for organism), was designed to address specific existing agent architecture limitations. An overview of autonomous agents is given followed by the design objectives used for the ORG architecture.

A. Review of Prior Research

Agents are complete systems that integrate sensors, perception, knowledge, and learning modules to achieve broad goals. All these components operate together and have a simple communication mechanism. An agent is situated in time and space. It deals with problems in an incremental, real-time iterative fashion. Environmental constraints are exploited in agent design and environment features can aid the agent in achieving its goals. For example, if an agent is restricted to move on a two-dimensional surface, it can customize its sensor interface to use the horizon as a reference point.

Agents can exist in a cooperative society of similar agents. Since they are independent they can be assigned different tasks and concurrently tackle different aspects of a problem. Sharing information is an option.

Side effects arising from the agent's interaction with the environment and from the interaction of its internal modules are often desired and help the system achieve complex goals without complicated heuristics and explicit hard rules. Previous work by others [1,2,4,9,10,12] provides an extensive set of autonomous agent architectures. A good overview of agent architectures, limitations and the fundamental differences among traditional A.I. is found in [3,6,13].

Any autonomous learning system has to deal with a number of issues, some of which are mentioned below:

1. *The Problem of Action Selection* – What should the agent perform, from a potentially large set of possible actions to optimize the achievement of its goals? With unsupervised agents this is especially difficult since the goals themselves can be time varying or ill-defined in terms of a single iteration step
2. *The Problem of Learning from Experience* – The agent should become better at achieving goals with experience.
3. *Scaling to larger problems* – An agent should be able to handle increasingly complex problems that demand a greater amount of knowledge, and to process an increasing amount of sensory information.
4. *Attention focus (reactivity)* – In a rich environment, a large amount of sensory information needs to be processed. The agent should be able to focus its attention on a small number of relevant stimuli so it reacts in real time and makes an optimal action selection based on the few stimuli that were chosen.
5. *Modularity* – Software engineering dictates functional decomposition: components should be designed according to their function, leading to a system easier to build and maintain.

Problems one, two and three are tackled through the simplicity and inherent explanation facility of the fuzzy system used by the ORG (Sections III, V, VI). The sensor architecture of combining adjustable virtual and real sensor ranges provides a tunable solution for tackling large amounts of stimuli and addresses issues one, four and five (section IV). Finally the ORG software design coupled with a new symbol system provides a portable, modular architecture addressing issue five (section III).

B. ORG Design Features

The main ORG features are summarized below:

1. *Simple interface for adding heuristics, explanation facility* – Stimuli, rules and acquired experience is represented as a collection of software objects. Fuzzy logic methods are used to manipulate them, providing an easy methodology for adding

domain knowledge and interpreting what the agent has learned at any time during its operation.

2. *Attention focus mechanism that can deal with large number of stimuli* – A fuzzy-logic based stimulus ranking technique (Section VI.B), combined with a range dependent sensory model, enables the ORG to focus its attention to a small number of stimuli (from a large pool of candidates). Range dependent modeling uses two distance thresholds to determine if stimuli should be evaluated (Section IV.A).
3. *Reactive and long-term behavior use* – The agent combines the reaction to multiple stimuli in its immediate environment with the reaction to long term goals defined either before the ORG comes online or during its operation.
4. *Self-Reflection ability* – The ORG's internal state is represented by the same symbols used to describe external stimuli. This allows the ORG to generate a reaction to its current or future state and affect its next state transition.
5. *Platform and application independent interfaces* – The ORG is implemented in a system of software modules that abstract their functions and hide problem or platform specific details. This makes the agent re-usable and portable.
6. *Large problem scalability* – The application chosen demonstrates the scaling ability of the ORG since it requires hundreds of stimuli to be evaluated before a reactive decision is made. It also presents unique difficulties since stimulus properties change depending on their relative position to the ORG. The ORG performs well in this scenario and even in its current form meets real time constraints.

II. Application and Simulation Environment Description

To better illustrate the ORG architecture, the simulation of a challenging real world problem is described and used as a running example throughout this work. A description of the problem is given below:

A vessel carrying a sonar/hydrophone array needs to effectively search a large volume of ocean water in order to detect underwater targets. Acoustic models are used to

generate sonar performance predictions, which then are used to calculate probabilities of detection for a finite number of points in the ocean.

The ORG is generating the vessel path that meets the physical constraints of the vessel, the time constraints of the task and minimizes the probability a target went undetected. Note that if the environment is known, the ORG simulation can be run before a real vessel goes to sea or it can be run concurrently, taking into account sampled environmental values and other changing parameters.

A. Simulation Environment

A three-dimensional virtual environment represents a 16000X16000X200m water volume. A grid point is placed every 1km in the XY plane and every 15 meters in the Z plane (depth). Figure 1 is a graphical representation of the simulation environment.

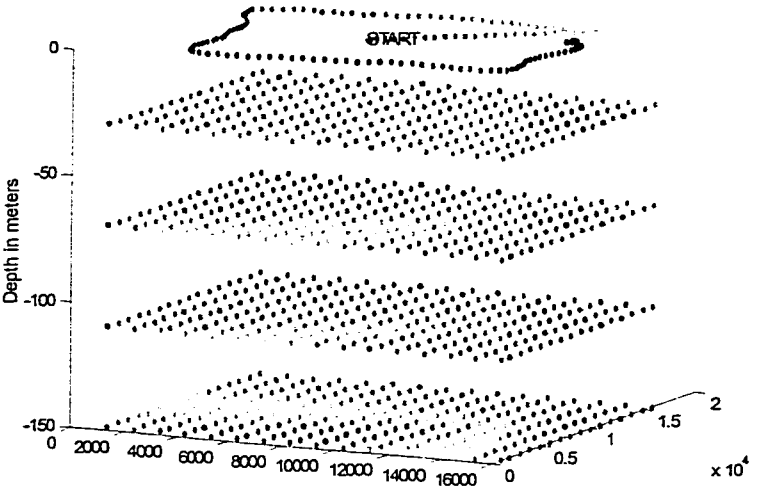


Figure 1. The ORG virtual environment.

Fig.1 illustrates the virtual targets in the simulated environment with the ORG path overlaid at the surface. The virtual targets are colored according to the sonar

performance at the targets location. The ORG trajectory is colored to illustrate incrementing iteration numbers.

Each grid point is modeled as a static external stimulus and is sensed if it falls within the agent's sensory range. Each stimulus has a number of dynamic properties that change depending on its position relative to the agent and the number and duration of prior encounters. Table 1 lists these properties.

Table 1 - List of properties in each stimulus

Property	Description	Dependencies
Signal Excess (SE)	Generated by a sonar acoustic model. High values translate to high probabilities of detection.	Re-calculated each time the org moves since SE detectability depends on the ORGs orientation and distance
Probability of target Not Detected (P_{ND})	Conditional probability calculated using Bayes theorem and a target strength distribution function.	Depends on *SE* value
Distance	Distance of a stimulus to the ORG	Depends on the ORG *position* property
Position	X,Y,Z coordinates	None
Theta	Angle between vector to the stimulus and current velocity vector	Depends on ORG direction

The signal excess property above reflects the sonar performance prediction. This value (and the associated probability of no detection) changes for each stimulus depending on the ORG's distance and orientation. This adds a challenge to the path-planning problem since the agent's perception of the environment changes as it moves. For example the same grid locations (and the stimuli representing them) will look entirely different to the ORG if it approaches them from two different angles.

B. Application Parameters

The agent assumes the role of the surface vessel searching the simulated environment, carrying a sonar/hydrophone array through a large water volume. The *task* is to form a path that minimizes the probability of a target being present and undetected (P_{ND}) for all grid points. The constraints can be summarized as follows:

1. Arbitrary velocity changes are not allowed and are constrained by physics.
2. The simulation objective must be achieved within a certain time.

III. Overview of Internal Architecture

A. Algorithm Flow

The ORG is implemented as an object oriented program with a primary iteration loop. A single iteration step is illustrated in Fig. 2. The action selection module at the bottom of Fig. 2 performs the last step in the iteration. The next iteration starts after the ORG collects environment information and passes the environment data to the sensor module at the top.

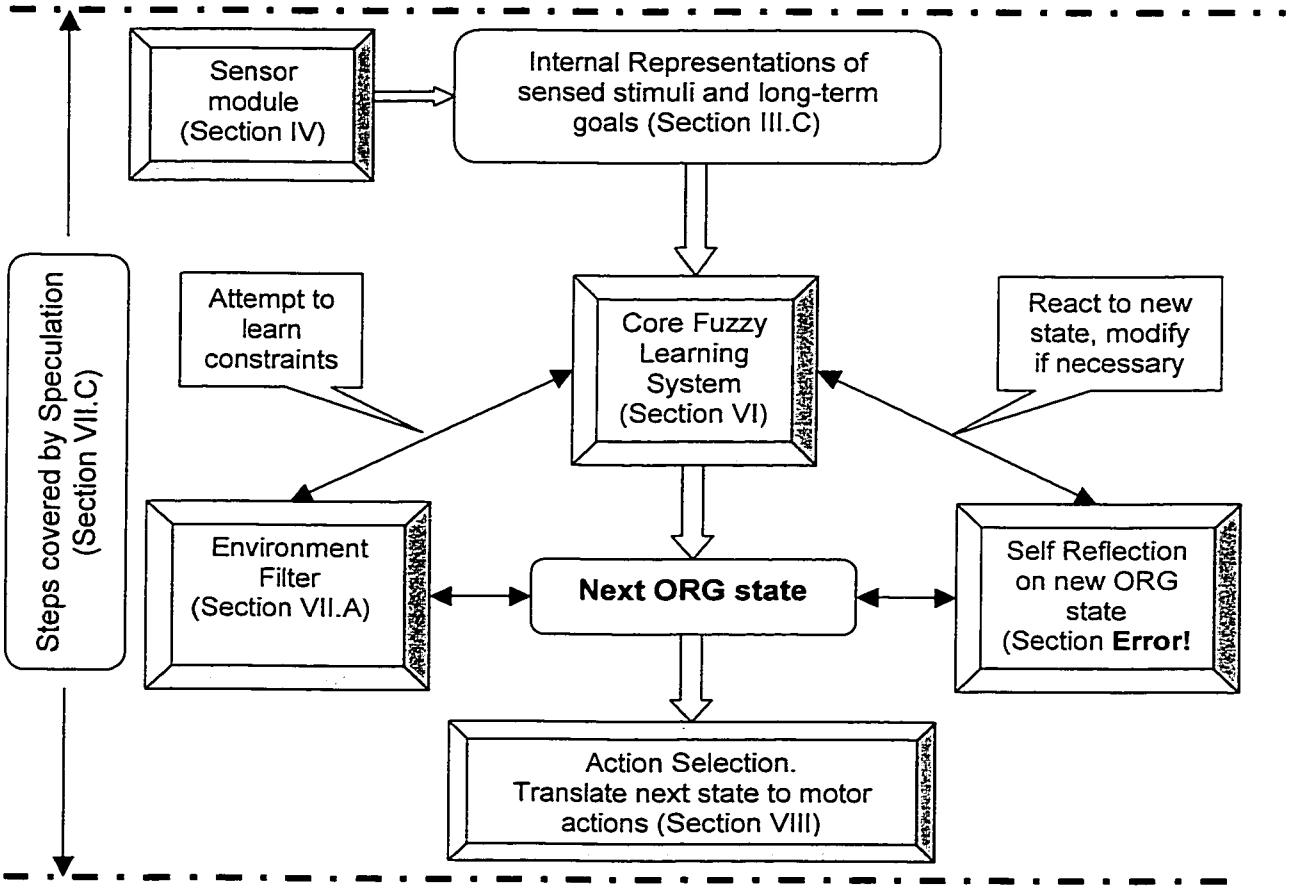


Figure 2. A single iteration step of the ORG loop.

The bi-directional arrows indicate that even if the ORG architecture is generally a feed forward loop, there exist mini-loops that feed back information. The core module is invoked multiple times from within other modules and there is no strict hierarchy among components (i.e., this is a flat network).

B. Internal Representation and the ORG Symbol System

Before the sensory module and the learning mechanism is presented, the manner by which the ORG stores information internally is explained. This section introduces the software symbols utilized to create internal representations of stimuli and the ORG itself.

The *physical symbol system hypothesis* [11] proposes that a physical symbol system has the necessary and sufficient means for general intelligence. Newell differentiates between the *symbols* (the abstraction of physical phenomena) and *tokens* (the physical instantiations). A *symbol system* has the following [3]:

- *Memory* – to store the symbol information
- *Symbols* – to provide a pattern to match or index remote token information
- *Operators* – to manipulate symbols
- *Interpretation* – to derive operations from the symbols
- *Capacities* for:
  1. sufficient memory,
  2. composability (the operators may make any symbol structure),
  3. interpretability (the symbol structures be able to encode any meaningful arrangement of operations).

A versatile symbol system and symbolic architecture that minimizes memory utilization and allows for efficient lookup of knowledge is proposed here. The ORG defines data structures to represent stimuli, fuzzy rules and its own internal state. The data objects and the methods used to manipulate them create the ORG symbol system. The code implementation is the fixed symbolic architecture

In [10] the authors present a software system for defining multi-agent missions. The work in [10] presents a software environment using object-oriented techniques to define agent behavior and situate the agents in an environment. Rules and module

interconnection are defined in computer code a priori and then compiled to produce the robotic agent. The ORG instead chooses a more flexible representation system based on a combination of fuzzy logic and software data objects. Fuzzy rules and a fuzzy logic inference already provide well established mathematical means for combining multiple human readable rules and dealing with partial or inaccurate sensory information (The fuzzy inference and defuzzification procedures employed by the ORG are described in App A., section A). Software engineering is utilized to make the agent modular, portable and easier to maintain.

The ORG software architecture defines a set of data structure objects used as primitives for information exchange and knowledge acquisition. The primitives are used as flexible containers capable of representing a wide range of objects with varying numbers of properties. Figure 3 shows the object hierarchy. Below is a sample 'C' language declaration of the object types (the ORG code is implemented in MATLAB 5 m-file language but C equivalent code is used here for clarity):

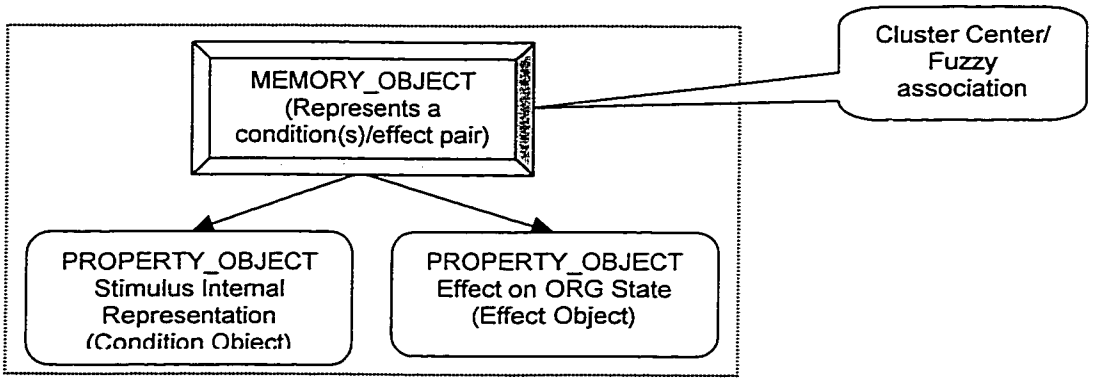


Figure 3. Symbol hierarchies

```
typedef struct _PROPERTY_OBJECT {
      DWORD ObjectType;
      BYTE        Index [MAX_PROPERTIES];
      BYTE        PropertyValues [MAX_PROPERTIES][MAX_SIZE];
} PROPERTY_OBJECT;

typedef struct _MEMORY_OBJECT {
      PROPERTY_OBJECT ConditionObject;
      PROPERTY_OBJECT EffectObject;
      BYTE        MergeCount;
      BYTE        LastIterationUpdated;
} MEMORY_OBJECT;
```

The `PROPERTY_OBJECT.PropertyValues` matrix contains the values for each property vector (row N, for property N). Each property can optionally have an associated time derivative, which is treated as a separate property and assigned an index number of N+1 in `PROPERTY_OBJECT.Index`. (N is the column index assigned to the original property, N+1 is the column index of its first derivative). Derivative properties used for the application described are dP_{ND}/dt, *dDistance/dt* (speed) *and dPosition/dt* (velocity). A property is considered *active* if `PROPERTY_OBJECT.Index[N]` is set to one, were N is the assigned property number. The properties active in the internal representation correspond to the physical properties present in the stimulus. The same symbol (`PROPERTY_OBJECT` in this case) can thus be used to describe different types of stimuli, with different physical properties.

C. Representation of Stimuli and the ORG State

A stimulus is represented using a `PROPERTY_OBJECT`. The internal ORG state is also represented using the same object but its `ObjectType` field is different (see Table 2) and has a different set of active properties. This allows the core module (Section VI) to filter learned associations intended for self-reflection (Section **Error! Reference source not found.**) and not use them to generate reactions to stimuli.

Table 2. Internal representations of objects

PROPERTY_OBJECT.ObjectType (Bit field)	Description
STIMULUS_EXTERNAL	Object is external to the ORG
STIMULUS_INTERNAL	Object represents an internal ORG state
STIMULUS_REACTIVE	Object is real (actively sensed)
STIMULUS_LONGTERM	Object compatible with long-term rules (learned condition/effect pairs). Its a representation of a stimulus previously encountered or of a desirable ORG state

Sample "C" code for defining the internal ORG state is given below:

```
ORG.ObjectType = STIMULUS_INTERNAL | STIMULUS_REACTIVE;
```

An external stimulus (the virtual targets in each grid point) is defined as:

```
Stimulus.ObjectType = STIMULUS_EXTERNAL | STIMULUS_REACTIVE;
```

IV. Sensor Module

Information about the environment gathered from various sensors flows first through the ORG sensory module. Separate functions within the module process raw information and create standard data structures (Section III.B) to represent discrete stimuli. A stimulus can appear discrete under a low-resolution sensor and appear as multiple

interconnected objects under a more advanced sensor. In the simulation discussed here, all external objects are discrete with a fixed number of observable properties (Table 1).

Whenever the ORG encounters a stimulus it creates an **internal representation** of that object using the PROPERTY_OBJECT structure. Due to learned experience or prior knowledge in the form of initial rules, the ORG might have *a reaction* to that representation. The reaction (described in Section VI.B) is the effect the stimulus will have on the next ORG state. That effect is also modeled using a `PROPERTY_OBJECT` and associated with the internal representation of the stimulus using a `MEMORY_OBJECT`. The combined reaction to all stimuli forms the *state transition* from the current ORG state to the next.

A. Stimulus filtering

The *distance* property is compared to a threshold to determine if the object is sensed in the current iteration and if its internal representation will be made available to the ORG. For this simulation the *distance* property has an infinite range, which means that no matter how far away an object is its distance can be determined. As the ORG encounters stimuli it creates data objects to represent them internally. In the simulation presented here, each stimulus is a virtual target placed at a fixed location on a grid. When the object falls within range, its internal representation becomes part of an internal map of the environment. Not all properties active in the representation of the stimuli, (P_{ND} is one example), have a physical equivalent. They are, instead, virtual properties that are only present in the data object used for the internal representation of the stimulus. This is an application dependent behavior and, in the current simulation, is needed to keep track of the cumulative P_{ND} associated with all virtual targets the ORG encounters.

The ORG employs two sensor thresholds to distinguish between the actual sensor limitations ($T_{physical\_range}$) and when to make the internal representation of a stimulus "available" ($T_{virtual\_range}$). Internal representation properties are updated only for stimuli that are within a close range $T_{physical\_range}$. In a real environment (vs. the simulation discussed here) $T_{physical\_range}$ is reduced to the physical limit of the particular

sensor, or even smaller (here it is set to 6km). This sensor modeling is summarized in Table 3 and utilizes *spatial locality to select a small number of relevant stimuli.*

Table 3 Explanation of range dependent sensor behavior.

Stimulus/Internal Representation distance (d) to ORG	Sensor behavior
$d < T_{physical\_range}$	Stimulus properties determined, internal representation properties can be determined and modified
$T_{physical\_range} < d < T_{virtual\_range}$	Stimulus properties determined, internal representation available but can not be modified
$d > T_{virtual\_range}$	Stimulus can not be sensed, internal representation is not made available

The ORG is capable of creating internal representation of stimuli from two sources:

1. The environment - with sensors providing the data describing each object.
2. A model - with a pre-initialized environment filled with virtual objects, as in a simulation.

In the simulation discussed here, if the human designer has model-supplied data or prior knowledge of the area being searched, he/she can pre-initialize a virtual environment with objects so that the ORG can sense and create internal representations. Then the ORG merges both the virtual environment with the real world representations to make decisions. This approach works for the specific application and might not be possible with other applications due to the *frame problem* discussed in [3] (section on architecture issues). The frame problem arises when the agent cannot easily associate an internal representation of an external object either because the object is moving or because the sensor information is not sufficient to distinguish it from others. In the simulation the static virtual targets are placed on a regular grid pattern, allowing the ORG to associate internal representations of stimuli created at some point in the past, with currently

observable virtual targets . In [9], multiple agents pursue each other but the frame problem is also addressed in the simulation by assigning unique color to each agent.

V. Instincts

Instincts are abstract rules that influence ORG behavior throughout the simulation. In this application instincts instruct the ORG how to react to external stimuli and long-term goals. The human designer forms instincts by specifying a MEMORY_OBJECT and then invoking the Core module in order for the ORG to learn the new association. The learning function is described in the following section. Once an instinct becomes part of the ORG, it can be modified during the simulation, just like other knowledge. The MEMORY_OBJECT representing the instinct translates to a fuzzy linguistic rule. The antecedents refer to properties in the internal representation of a stimulus, the consequents describe the effect that sensing this stimulus will have on the ORG state. One of the instincts used in this simulation is described below:

> *IF* P_{ND} is VERY LARGE **and** Induced change in ORG direction (dTHETA/dt) due to this stimulus is VERY SMALL **and** distance to stimulus is SMALL *THEN* attempt LARGE DECREASE in distance to stimulus with the *intent* of LARGE DECREASE in P_{ND} in the location of the stimulus

Instincts can provide a performance indicator and the ***notion of intent*** if they specify the derivative of an internal ORG property and the desirable change direction (negative or positive). The intent property must be part of the Effect portion of the instinct and must not map to a motor function. This is how the ORG discovers intent properties. In this context, "intent" is a vague effect property that helps the ORG determine the motivation behind the instinct.

The use of instincts highlights one desirable feature of lazy learning methods (see [7] pp. 245]): vague rules representing prior knowledge of the problem domain are

utilized to sustain performance when available training instances are scarce or not available. An example of an instinct is given below in MATLAB5 script language.

```
%
% Define condition property object
% Active properties we look for in the stimulus are
% 1) its current probability of detection
% 2) the deviation it would cause to our current velocity
% 3) its distance from the ORG
% The variables are defined as follows:
% PROB  = probability of target present (P_TP)
% THETA = angle between the ORG velocity vector and the
%         direction vector to the stimulus (THETA+1 is the first
%         time derivative)
% DIST  = Euclidean distance to the stimulus
% FUZZY_SET_xxxx = Fuzzy set support (center) number. It
%                  characterizes the fuzzy set to use. This can
%                  be used if a crisp number is not desired.
% pI is a variable of type PROPERTY_OBJECT used to specify the
% condition portion of the instinct
% pO is a variable of the same type, used to define the effect
% portion of the instinct. The instinct is then defined by
% variable m, of type MEMORY_OBJECT.

pI = PROPERTY_OBJECT;

pI.Index(1,[PROB THETA+1 DIST ])  = [ 1 1 1];
pI.Type = bitor (STIMULUS_EXTERNAL,STIM_RT);

pI.Values(PROB) = FUZZY_SET_LARGE_POSITIVE;
pI.Values(THETA+1) = FUZZY_SET_ZERO;
pI.Values(DIST) = ORG.Constraints.NearSensorRange;
%
% Define the effect property object.
% Only one motor action is defined: Decrease distance
% One intent property is defined: Decrease probability target is
% present
%

pO = PROPERTY_OBJECT;
pO.Index(1,DIST+1 PROB+1]) = [ 1 1];
pO.Values(DIST+1,1) = FUZZY_SET_LARGE_NEGATIVE*
   SIM.ORG.PropertyObject.RangeHi (DIST+1);
pO.Values(PROB+1,1) = SET_LARGE_NEGATIVE;
```

```
% Create rule association in the form of a MEMORY_OBJECT

m = MEMORY_OBJECT;

m.P = pI;
m.Effect = pO;

% Call the CORE_LEARN function to cluster the memory object to
% the ORG's existing knowledge

Core(CORE_LEARN,m);
```

VI. The Core Fuzzy System

The core fuzzy system is a variable-input/variable-output fuzzy engine that uses the data structures defined above to determine the relevant information content to be used from each learned fuzzy association. A fuzzy association is a rule comprising of an input and output pair. The rule is in human readable form since it can be translated to a linguistic rule in the form described in section V. A clustering algorithm is used to compress information and merge similar pairs with each other. ORG modules invoke the core fuzzy module (central block in figure 1) in the following modes of operation:

1. CORE_LEARN – The caller module supplies a `MEMORY_OBJECT` with a condition and effect object. The core learns this new association through fuzzy clustering
2. CORE_REACT – The caller module supplies a `MEMORY_OBJECT` with only a condition object (the effect is left empty). In response the core generates a reaction to this condition using all relevant existing associations. The reaction is returned in the form of an effect `PROPERTY_OBJECT`. If there are no relevant associations in the ORG's memory, an empty (null) reaction is generated and the ORG does not modify its behavior due to this condition object.
3. CORE_RECALL - The caller module supplies a `MEMORY_OBJECT` with only a condition object (the effect is left empty). The core will attempt to find all similar condition objects and return them in a list, using a fuzzy membership indicator (discussed in section B below) to rank them in descending order

Note that the Core module could potentially use a different algorithm for learning and later recall condition/effect pairs. Fuzzy logic is used because it provides an explanation facility. Learned rules are in human readable form and for each input presented to the core fuzzy engine, the human designer can trace which rules, and to what extent they contributed to the output (references [14,15] are highly recommended for readers not familiar with fuzzy logic).

A. Learning new associations (CORE_LEARN)

This function is used to add a new MEMORY_OBJECT to the existing rule base. It uses the clustering procedure described in Appendix A to add a new condition/effect pair (MEMORY_OBJECT). New clusters are created or old ones absorb the new MEMORY_OBJECT as needed in order to fit the finite memory implementation of the ORG. Learning is described in more detail in section VII.

B. Reacting to a stimulus

The sensor module (top block in Figure 1) creates a condition object for each stimulus within $T_{virtual\_range}$, invoking the core (module B) for a fuzzy reaction to this condition. A fuzzy reaction is the output produced by evaluating the input through Eq.2 in App. A for each rule in the fuzzy rule base. The core then returns a fuzzy output object (App. A, Eq. 3) which is combined to form a condition/effect pair in the form of a MEMORY_OBJECT. Each pair is then ranked in descending order based on the total fuzzy membership value its condition generated when compared to all existing rules. That value is called the ***fuzzy rule base membership indicator*** and is calculated using standard fuzzy logic inference techniques (App. A, Eq. 2).

High membership indicators indicate that the particular stimulus matched well to the already learned associations (such as instincts) in the core. The effect objects of the highest ranked N pairs are then averaged together and used as the *combined reaction* to the environment for this iteration. N is a tunable parameter in the ORG state and is usually a small percentage of the number of stimuli within $T_{virtual\_range}$. Using the fuzzy membership indicator of how relevant existing rules are to the current stimulus,

combined with range dependent filtering, described in section IV.A, comprises *the ORG attention focusing mechanism.*

C. Long-Term Goals

Long-term goals motivate the ORG to escape local minima and provide the means for human objectives to be incorporated into decision-making. A long-term goal is defined by:

1. A `PROPERTY_OBJECT` describing the long-term stimulus. This can describe an object, physical or virtual, with any number of active properties. It can also be a desired future internal ORG state
2. A fuzzy association between a condition and an effect (a `MEMORY_OBJECT`) that tells the ORG how to react to the object in 1).

During ORG initialization (before the iteration loop starts) a list of all known long-term objectives is created and used during the iteration loop to generate a cumulative long-term reaction. The aggregation of ORG reactions to each long-term stimulus is done in the same fashion as with the sensory stimuli, but with no ranking. This means all long-term objectives are considered.

1. Dynamic Long Term Goal Generation Utilizing Sensed Stimuli

The highest ranked internal representation of a stimulus with a distance *d* to the ORG, which satisfies the inequality

$$T_{physical\_range} < d <= T_{virtual\_range}$$

is added to the long-term list. The process of adding internal representations of desirable stimuli, that satisfy the above inequality, to a volatile list for future evaluation, is the ORG's dynamic long-term generation facility.. In this simulation, "desirable" are stimuli with a high P_t property, due to the instincts used. The list of stimuli is used to keep the ORG reacting to some input in situations when no desirable stimulus is within sensor range. This reduces the chance the ORG stagnates (no longer moves or moves in circles). Objects that have come within close range in the past will stop being considered in the long-term list because the ORG can affect their P_{ND} values making

them less relevant to one of the current instincts. This eliminates the need of keeping track when a goal generated with this means has been "achieved".

D. Clustering algorithm

A variant of the sequential fuzzy c-means algorithm [5], with no predetermined number of clusters, is used to classify rules based on their condition (input) PROPERTY_OBJECTs. Each cluster is the association of an input PROPERTY_OBJECT (for example, a stimulus) to an output PROPERTY_OBJECT (the effect the stimulus has on the ORG). The input object is *condition* portion and the output object is the *effect* object. The fuzzy membership of the input condition to the condition object of the cluster is used as the clustering threshold. Clusters are represented by MEMORY_OBJECTs and are created dynamically based on a global expansion threshold T_e (defined in App. A, Eq. 1). A maximum number of clusters can optionally be specified to force any new associations to merge with an existing cluster. The clustering method (described in App. A) is a data driven approach with certain thresholds determining the maximum number of clusters after instances are classified.

VII. Learning

The ORG can learn by:

1. Observing the changes imposed by the environmental filter and learning environmental constraints,
2. Speculation and experimentation.

A. Environment Filter

As mentioned earlier (section IV.A), the combined reaction to external stimuli is used to generate an effect on the ORG's current state. The effect object contains the state transition matrix for the ORG's motor functions. Motor functions are described by properties only active in the ORG internal state. The state transition matrix is comprised of derivative properties that express the transition of each property from its current state to the next (note that state changes only between iterations). Before the state transition matrix is applied to the current ORG state, the simulation code ensures that the transition will not violate physical laws and constraints of the vessel the ORG is emulating. To guarantee this, each state transition is filtered through an *environment filter module.*

The filter module emulates the natural world. It uses physical models and modifies the state transition matrix to fit the constraints of the environment and of the vessel the ORG is controlling. The modified state transition is called the *filtered state transition* and is compared with the original state transition. If they differ, a condition/effect association is created in the form of a `MEMORY_OBJECT` with the intended state transition associated with the filtered state transition. By repeatedly learning similar cause and effect relationships the ORG can use the learned fuzzy associations to predict environmental constraints and self-reflect (see following section) on its intended state transition.

B. Self Reflection

The ORG makes decisions based on the following input:

1. Sensory Information from its current environment (Section IV.A),
2. Long-term objectives (Section VI.C.),
3. Its own internal state.

Each state transition generated by the core fuzzy module should adhere to physical constraints. The ORG addresses this by learning prior modifications to its state transition, imposed by the environment filter, and then utilizes that knowledge each time it attempts to modify its state. That process is called self-reflection since the ORG generates a reaction to the internal representation of its proposed next state. That reaction is then applied to modify the proposed state change before it is translated to motor functions (Figure 4).

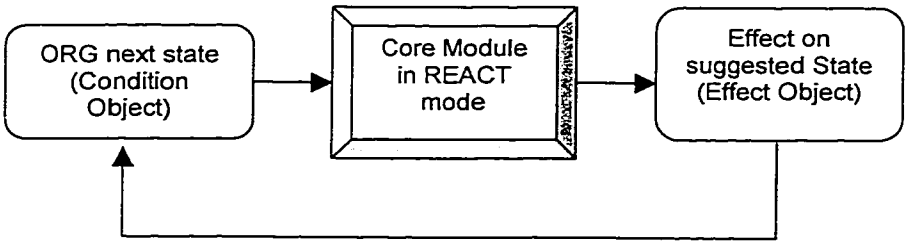


Figure 4. The process of Self-Reflection.

The self-reflection feedback loop is run only once per iteration.

C. Speculation And Experimentation Facility

Speculation is part of the learning process and is used to evaluate new rules and determine appropriate motor function mappings for reactions generated by the core module. The ORG starts the speculation procedure by creating one or more alternate versions of itself through experimentation. It then runs the alternate version(s) for N iteration steps, reacting to a snapshot of the environment and performing N "soft" state-transitions, predicting the sensory input at each step. At the end of the speculation run, the ORG compares the performance of the alternative ORG(s) with the original ORG (which also run for N time steps in parallel) and chooses the one that performed better, according to the instincts of the original ORG (Figure 5). The simulation environment

makes speculation feasible since its possible to predict sensory input as the ORG moves in the simulation space.

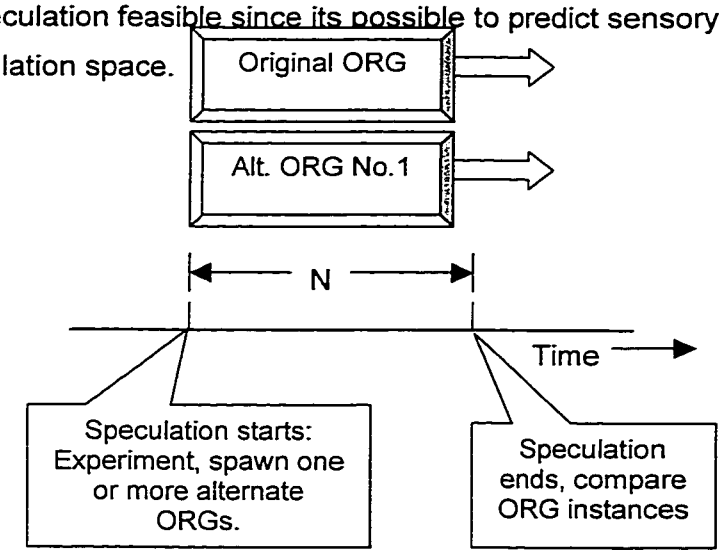


Figure 5. The speculation process

This strategy differs from those presented in [2] since it compares the alternative system as a whole, not just the new rules created by experimentation. Emergent properties that arise during the N iteration steps can then be observed and utilized in determining the usefulness of the experiment.

D. Experimentation Strategy

The ORG experiments in the beginning of speculation by creating new condition/effect associations or by slightly modifying existing knowledge. Each new fuzzy association (represented by a `MEMORY_OBJECT`) is then presented to the core fuzzy module, operating in the CORE_REACT mode. If a similar association exists in the core, a high fuzzy membership indicator will be returned and the new fuzzy association will be discarded. If the new fuzzy rule is kept however, the ORG is cloned (see Fig. 5) creating an exact replica of itself. The clone then adds the new rule to its fuzzy rule base (see App A. section B for the process of clustering a new rule) and the clone is evaluated as a whole for N iteration steps.

VIII. Action selection

The sensor module invokes the core module to generate a reaction using all the learned associations and instincts within the fuzzy rule base. The reaction is an effect data object. The properties present in the effect might not map exactly to ORG motor functions since they were generated by using a number of vague rules (instincts) and acquired knowledge. Thus, an application specific function is

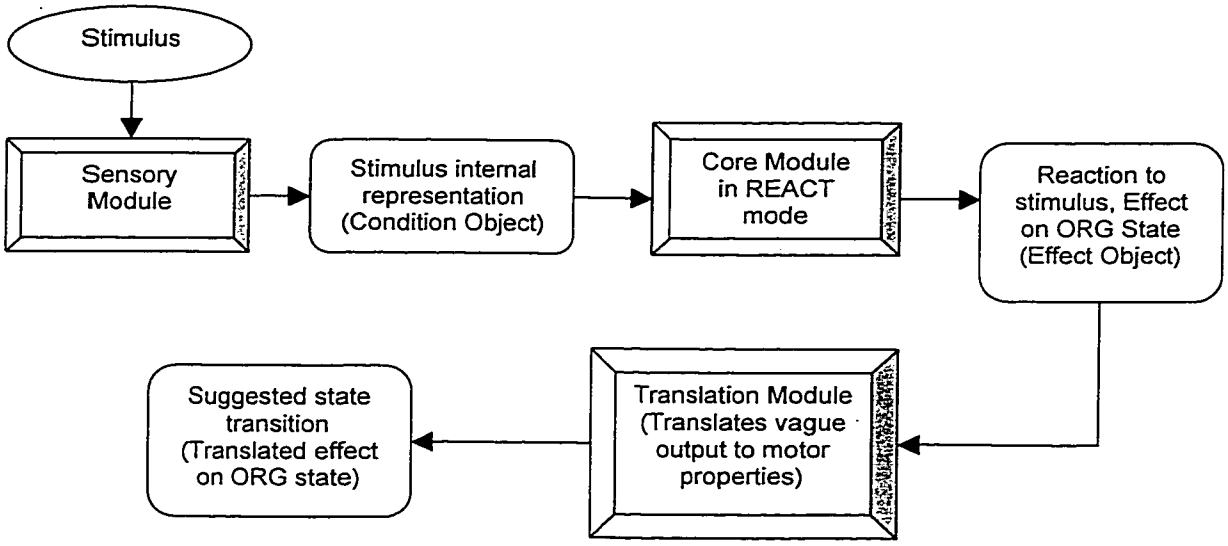


Figure 6. Translation of a stimulus reaction, to an ORG state transition

called to perform a translation (translation module in figure 6). If the agent is not given the motor function to action mapping, it can learn how to create that mapping by experimentation. The instincts provide the means to determine which coordinated motor action is required, given a vague effect property.

While experimenting, the ORG can determine which motor function resulted in the desired effect described by the instinct. For example, the instinct in section V calls for the ORG to decrease distance to a stimulus. If the ORG creates a velocity vector that brings it closer to the stimulus (during experimentation), this motor action will then be associated with the effect property in the instinct.

IX. Discussion of Results

The ORG produced several paths through the simulated environment that met the constraints. Different environment sizes, starting positions and initial instincts were tried. Also different features and capabilities of the ORG were modified to observe the effect they had on the ORG path.

A. Side-effects resulting in emergent behavior

By utilizing different instincts and by modifying the values for $T_{physical\_range}$ and $T_{virtual\_range}$, interesting side effects emerged. The side effects described in the following sections emerged because of the interaction of the ORG with the specific simulation environment described in section II.

Three different instincts were used (in descending order of complexity) in the simulations described in the following sections:

1. *IF* P_{ND} is VERY LARGE *THEN* attempt LARGE DECREASE in distance to stimulus with the *intent* of LARGE DECREASE in P_{ND} in the location of the stimulus
2. *IF* P_{ND} is VERY LARGE **and** Induced change in ORG direction (dTHETA/dt) due to this stimulus is VERY SMALL T*HEN* attempt LARGE DECREASE in distance to stimulus with the *intent* of LARGE DECREASE in P_{ND} in the location of the stimulus
3. *IF* P_{ND} is VERY LARGE **and** Induced change in ORG direction (dTHETA/dt) due to this stimulus is VERY SMALL **and** distance to stimulus is SMALL *THEN* attempt LARGE DECREASE in distance to stimulus with the *intent* of LARGE DECREASE in P_{ND} in the location of the stimulus

1. Motivation to always keep moving towards un-visited areas

In the discussion of the sensory interface we mentioned that the virtual properties of only the stimuli within $T_{physical\_range}$ are changed. In this simulation this translates to the

probability values being reduced since they came within close range of the ORG. Those stimuli thus become "un-attractive" (dashed objects in Fig. 7) and cease to influence the ORG, effectively pushing the agent towards any un-affected objects that lie in the area between $T_{physical\_range}$ and $T_{virtual\_range}$ (objects with solid gray or black colors). Due to inertia a further bias exists that pushes the ORG towards the objects that lie in its current direction of movement (solid black objects). This behavior resulted with any of the three instincts due to one common condition found in all of them: *decrease distance to stimulus* based on probability.

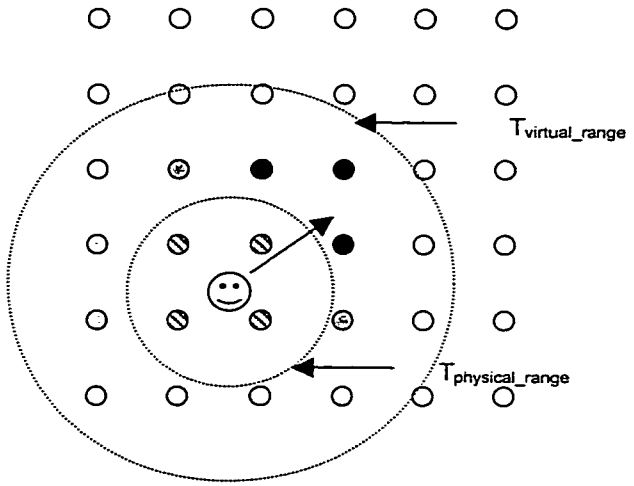


Figure 7. Tendency to cover new area

As Fig. 7 illustrates the ORG has a bias towards stimuli that lie in its current path where distance satisfies the inequality is $T_{physical\_range} < d < T_{virtual\_range}$.

2. Rapid probability reduction versus exhaustive, longer path

A side effect of the attention-focusing scheme employed by the agent was that the Org moved fast through the entire area (thus reducing rapidly the total probability a target

was present and un-detected). However a very small portion of the search area was not covered. This is a typical result present in all simulation runs and is illustrated in Fig. 8, where only the upper right corner still has a large average probability for that depth column.

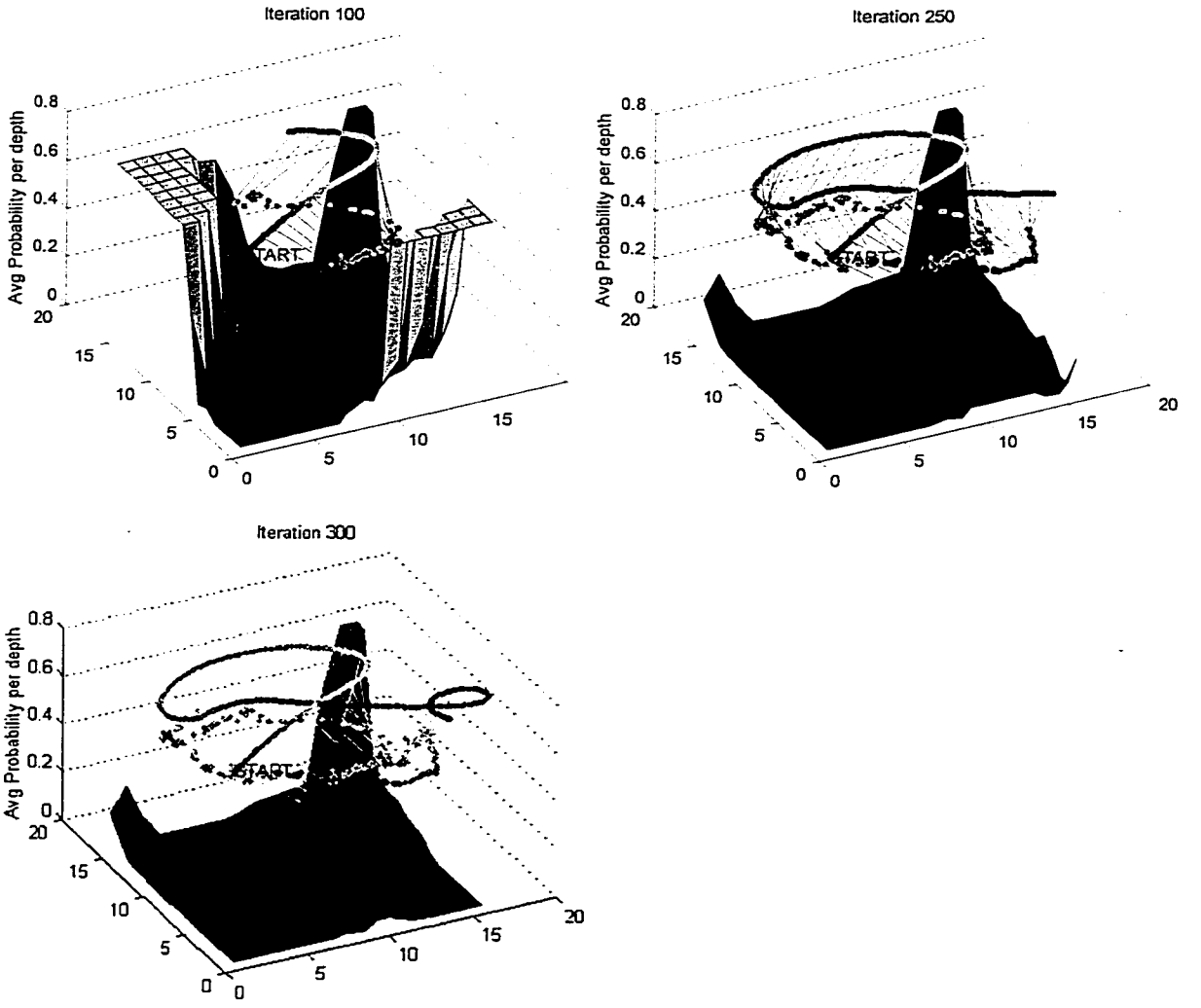


Figure 8. Org run with instinct 1 and with $T_{physical\_range}$ = 5000m, $T_{virtual\_range}$ = 8000m.

The position of the ORG during each iteration step is represented by a point in the path overlaid on top of each mesh in Figure 5. Each iteration corresponds to 30 seconds of real time, with a constant ORG speed of 10 knots/hour. The ORG starts on the bottom

left corner of the first snapshot and then follows a diagonal path through the search area turning left when it reaches the upper right corner. The path is color-coded to reflect the time progression. The three dimensional colored mesh under the path represents the average probability of target present (P_T) per depth column. The thin lines connect points in the ORG path to the centroid of the 10 most attractive stimuli at each iteration step. The last plot in in Fig. 8 illustrates the diminished returns after iteration 250: the ORG has already reduced the overall probability and is focusing in the lower right corner (red loop in the path) in an effort of diminished returns.

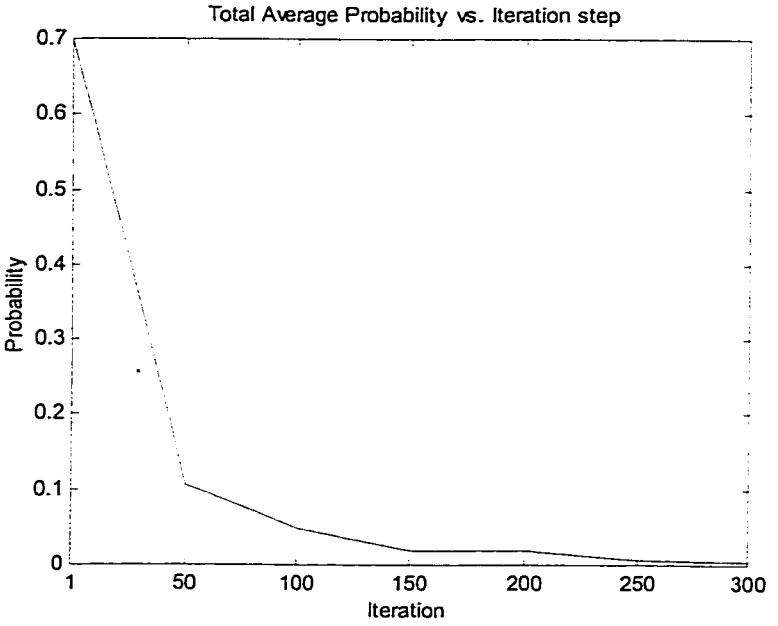


Figure 9. Rapid probability reduction.

Fig. 9 is plot of the total probability reduction vs. iteration for the path in Fig. 8. In the first 50 iterations the Org had reduced the overall probability by more than 80% of its initial value.

3. Dependency on $T_{virtual\_range}$ and the number of stimuli chosen per ranking

The ORG employs two sensor thresholds (Section IV) to distinguish between the actual sensor limitations ($T_{physical\_range}$) and when the internal representation of a stimulus

should become available ($T_{virtual\_range}$). If the $T_{virtual\_range}$ is expanded to a number close to the size of the simulated environment, the ORG sees all grid points from any position and hundreds of stimuli get processed during each iteration step. This is desirable if the human designer wants the ORG to take advantage of prior knowledge of the entire environment, while at the same time reacting to stimuli within close range.

Instinct 1 gives no preference to distance or angle from the current ORG path, evaluating stimuli only based on their current probability. This creates erratic paths since stimuli far apart (and ranked high enough to be chosen with the top N stimuli) generate conflicting reactions from the ORG. Even when instinct 2 is used, unsatisfactory results are obtained (Fig 10.). Note that for the path in Fig. 10 each iteration step represents 90 seconds (previous paths used 30 seconds) so this path is especially time consuming..

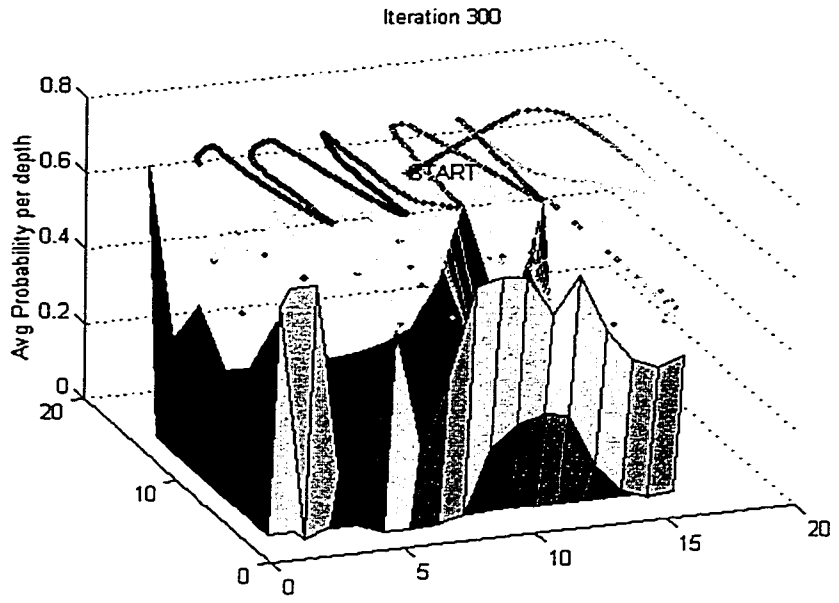


Figure 10. Results when instinct 2 was used.

Fig 10. is from an ORG simulation with $T_{virtual\_range}$=14km and instinct 2 utilized.

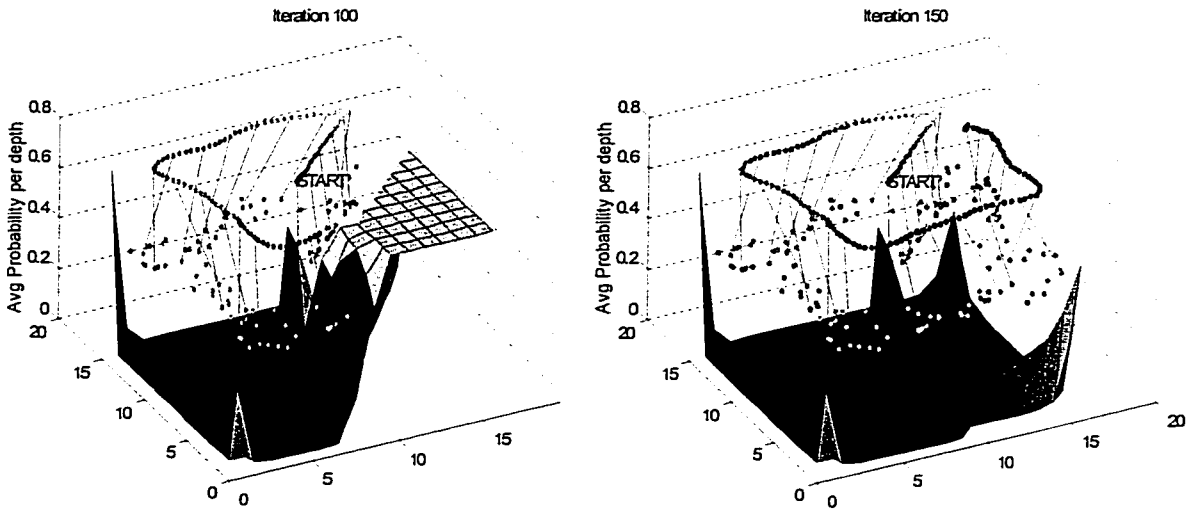


Figure 11. Resulting path when instinct 3 used.

Fig. 11 is the resulting ORG path when instinct 3 was utilized and $T_{virtual\_range}$=14km, $T_{physical\_range}$=3km. With all other parameters remaining the same the resulting path was much shorter and smoother .

The attention focusing mechanism depends highly on the fuzzy membership indicator computed for each stimulus in the sensor module. Utilizing an instinct with more input condition properties (instinct 3) makes the fuzzy membership indicator sensitive to more information per stimulus. Stimuli that looked "desirable" with instinct 1, now ranked very differently, helping the ORG focus better. These results strongly suggest that the complexity of the instinct must increase with the amount of stimuli being processed.

B. Effects of Learning on the ORG path

In the previous section we presented the ORG behavior that emerged out of the simple instincts and the ORG architecture. Environment constraint learning and speculation were not enabled. In this section we will discuss the resulting ORG paths when learning is utilized during the simulation.

When learning was enabled the attempted ORG state transitions adhered more to the physical constraints, reducing the changes imposed by the environment filter. The

information contained in the ORG core memory module reflected which type of fuzzy associations the ORG had learned at the end of the simulation. Since the ORG memory has a finite size, when the memory limit was reached, the clustering algorithm would either replace fuzzy associations entirely or modify existing ones. Fig. 12 shows the number of clusters in the ORG core memory, updated or replaced during the simulation. If a cluster was replaced, this shows when the replacement took place. Note that cluster 1 is occupied by the instinct, and it's the only one that didn't get replaced by the fuzzy associations produced from learning.

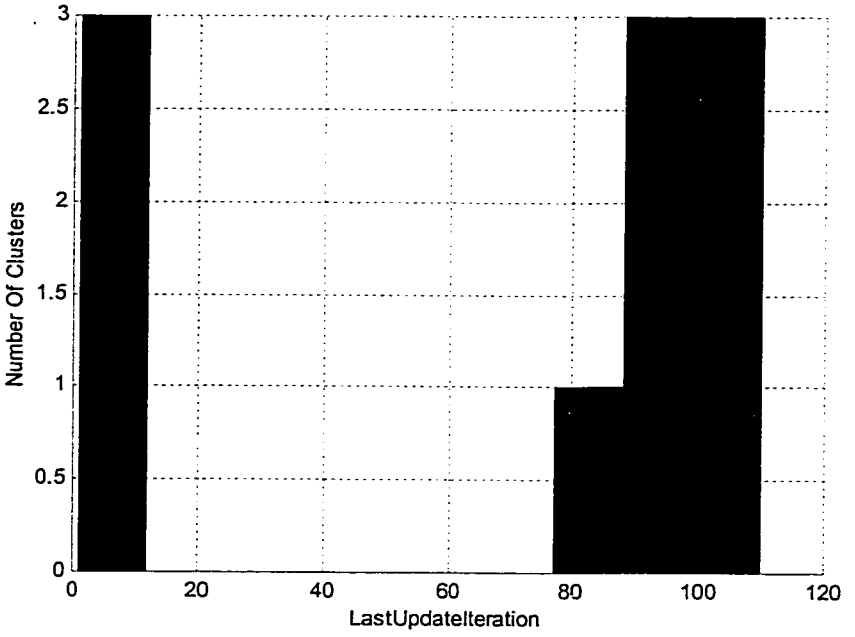


Figure 12. Clustering results with learning enabled.

Fig 12 shows the iterations during which the core clusters were last updated for the simulation in Fig 15. Fig 13 and Fig.14 contrast the ORG path between simulations with learning external constraints enabled and one with learning disabled. All other simulation parameters were identical. Utilizing learning to alter the next state transition produced a smoother, shorter path as illustrated by Fig 14. The probability reduction was nearly identical between the two runs. Each ORG path missed stimuli on one of the corners

primarily due to the very short physical range (3Km). Fig.15 is the simulation result when the physical range was set to 6Km and physical constraint learning was enabled. The entire space was covered and the total average probability was reduced to near zero.

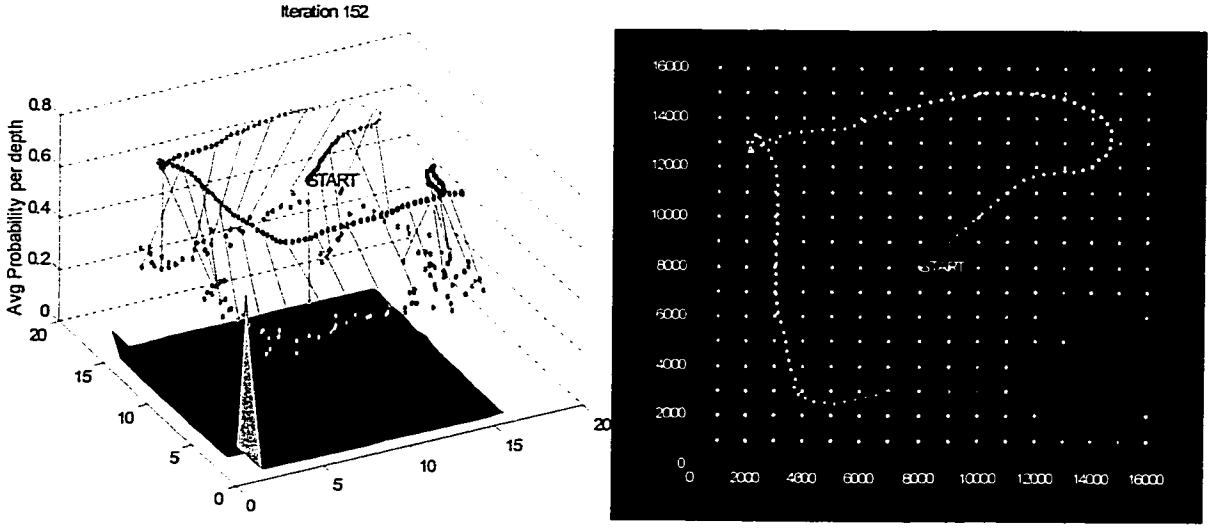


Figure 13. Results with learning disabled.

Fig. 13 is the resulting ORG path with $T_{physical\_range}$ =3 Km, $T_{virtual\_range}$ = 16Km and with instinct 3 utilized. Learning from constraints disabled.

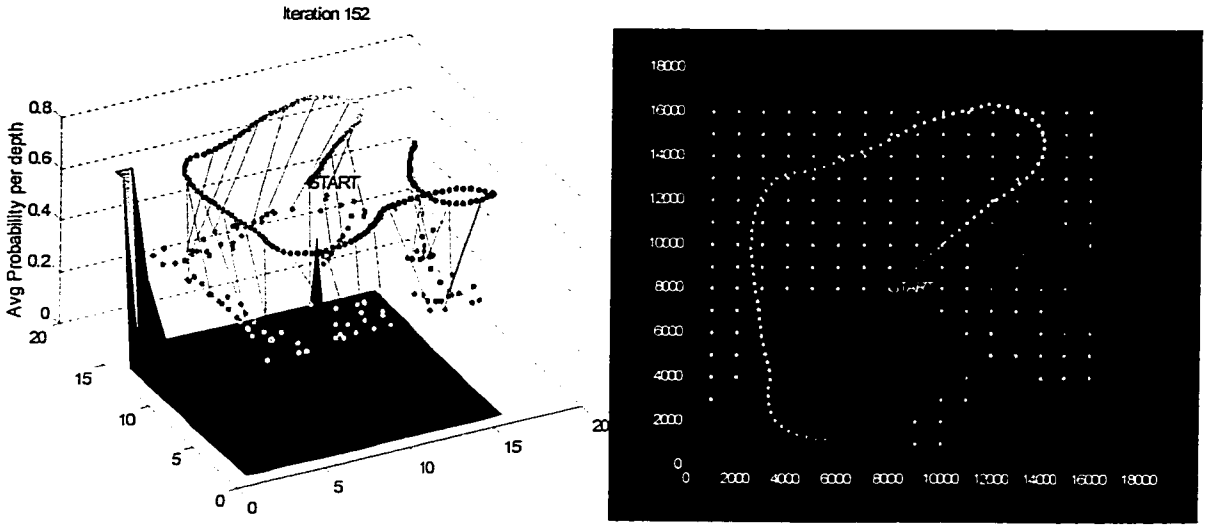


Figure 14. Results with learning enabled.

The ORG path with same simulation parameters as Fig. 13 but with learning from external constraints enabled produces a smoother and shorter path (Fig 14.)

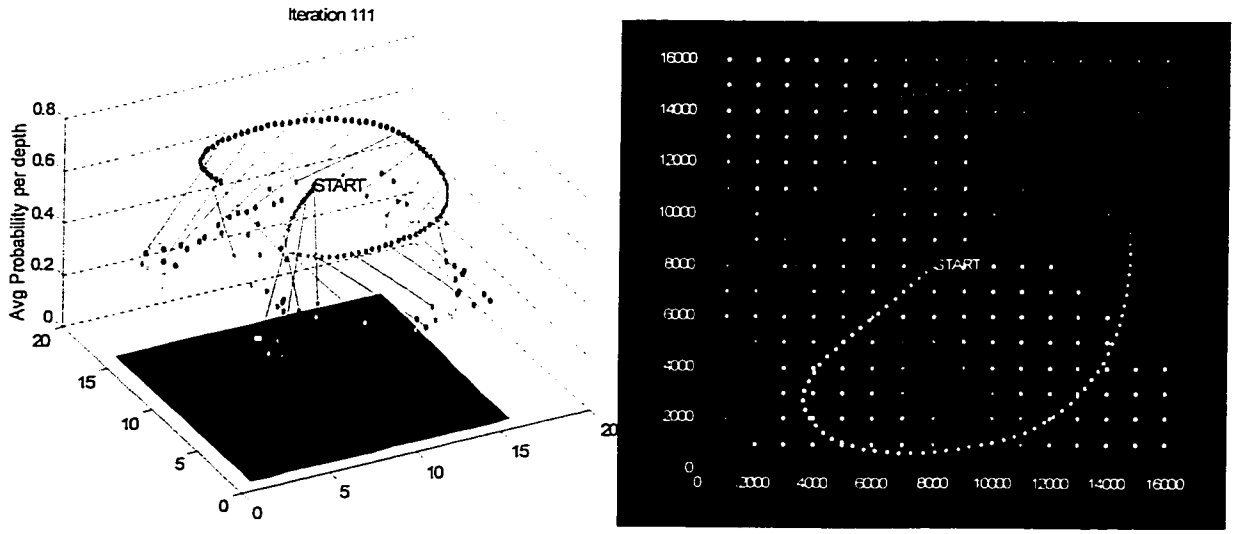


Figure 15. Learning enabled, larger sensor range.

Fig 15. is another illustration of the posistive effects of learning. For this ORG path $T_{physical\_range}$ =6 Km, $T_{virtual\_range}$ = 16Km and instinct 3 was used.

C. Robustness

The outcome of many simulation runs revealed that the Org produced desirable paths independent of its starting position and orientation. The simulation run of Fig. 16 has the same parameters as that of Fig. 15 except the Org starting position. The Org starts in position (1000,1000,0) instead of (8000,800,0). The path is different form compared to the path in Fig. 15 but the two paths took approximately the same time (103 iterations vs. 111 in Fig 15) and the final overall probability was near zero in both cases.

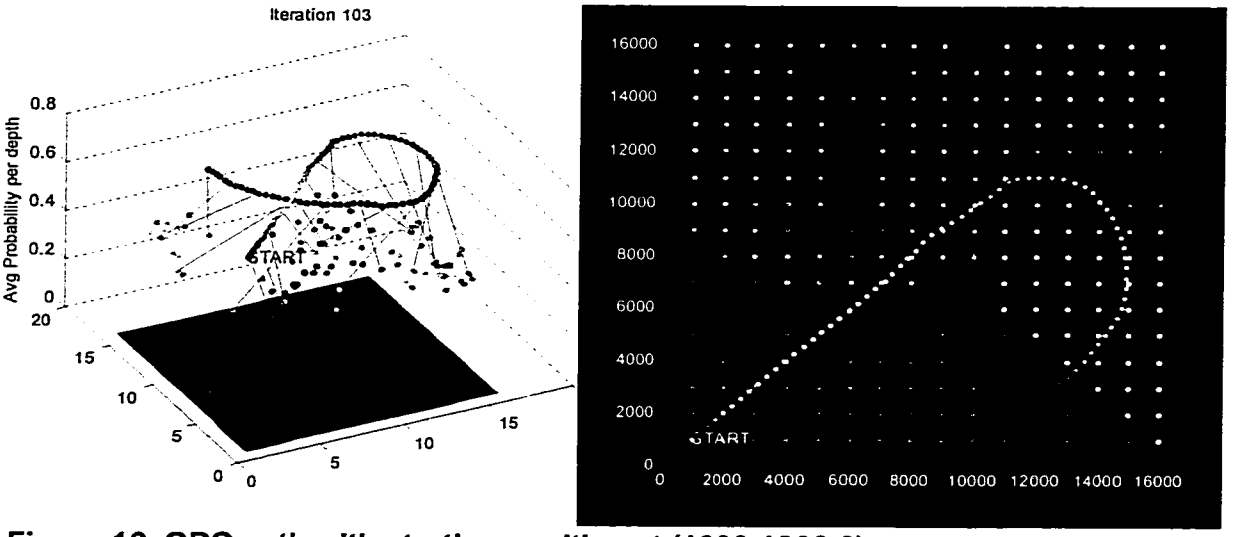


Figure 16. ORG path with starting position at (1000,1000,0).

X. Conclusion

An unsupervised problem solver implemented as an autonomous agent has been proposed in this paper. Utilizing data objects for flexible knowledge representation and fuzzy logic for storing and retrieving information, the agent provides a new physical symbol system. Abstract initial rules shape the agents behavior and provide the means to discover global intent behind the human supplied heuristics.

By implementing the core learning system using fuzzy logic, the issues of action selection, knowledge representation and attention focusing are addressed with simple

and proven means. Fuzzy membership indicators are used to optimize learning and focus the agent's reaction when dealing with large numbers of external stimuli. Sensor modeling based on two different range thresholds merges internal representations of stimuli with the actual sensor data allowing for hybrid model/real world environments. By observing the modifications imposed by the environment on the agents suggested state transitions, the ORG can learn physical constraints and adapt its behavior to comply with the constraints. Speculation and experimentation is used to expand the ORG's abilities and fine-tune its performance. Experimentation allows the ORG to gradually map vague fuzzy reactions to motor functions.

Several simulation results are presented that illustrate typical ORG behavior. Side effects of the instincts and ORG architecture are discussed in detail demonstrating that complex and desirable (for the specific application) behavior emerges out of simple heuristics and a flexible architecture. The effect of external constraints learning on the simulation results is discussed, demonstrating the subtle and positive influence of learning by self-reflection.

List Of References

1. El-Nasr and M Skubic, M., "A Fuzzy Emotional Agent For Decision Making in a Mobile Robot", in *Proc. IEEE WCCI,* Anchorage, Alaska, 1998
2. Wiesmeyer, M. and Laird, J., "Attentional Modeling of Object Identification and Search", *The Soar Papers*, Rosenbloom, P., Laird, J. and Newell, A., ed., Cambridge, Massachusetts, MIT Press, 1992
3. University Of Michigan, Electrical Engineering Department, "A Survey of Cognitive and Agent Architectures", URL: http://ai.eecs.umich.edu/cogarch0/
4. Brooks, R. A., "A robust layered control system for a mobile robot", *IEEE Journal of Robotics and Automation, RA-2*, April 1986.
5. Chintalapudi, Krishna K and Moshe Kam, "A Noise Resistant Fuzzy C Means Algorithm For Clustering", *In Proc. IEEE WCCI,* Anchorage, Alaska, 1998.
6. Maes, Pattie, "Modeling Adaptive Autonomous Agents", *Artificial Life An Overview*, Christopher G. Langton, ed., Cambridge, Massachusetts, MIT Press, 1997
7. Mitchell, Tom M. *Machine Learning,* WCB/McGraw-Hill 1997
8. VanLandingham, Hugh. and Chrysanthakopoulos, George, "Data Driven Fuzzy Logic Systems For System Modeling", in *Proc. IEEE Systems, Man And Cybernetics,* p.841-4 vol.1, Vancouver, B.C, Canada, 1995
9. Chrysanthakopoulos, George and Marks, Robert J, "Simulated Autonomous Agents Utilized Instincts and Behavior Learning: Orgs in Orgland", in *Proc IEEE International Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence,* p.727-34, Anchorage, Alaska, 1998
10. MacKenzie, Douglas C. Cameron, Jonathan M. Arkin, Ronald C., "Tactical mobile robot mission specification and execution", in *Proc The International Society for Optical Engineers,* vol.3838, p.150-63, 1999
11. Newell, A. *Unified Theories of Cognition,* Cambridge, Massachusetts, Harvard University Press, 1995
12. Manolov,O.; Stanev, P., Noikov, S., Janglova, D., Uher, L., Vagner, S., "Intelligent autonomous agent motion control by fuzzy logic in unknown environment", in *Proc*

Seventh International Conference on Artificial Intelligence and Information-Control Systems of Robots '97, p.127-32, Smolenice Castle, Slovakia, 1997

13. Bryson, J., "Cross-paradigm analysis of autonomous agent architecture", *Journal of Experimental and Theoretical Artificial Intelligence*, vol.12, no.2, p.165-89, April-May 2000
14. Zadeh, L.A., "Outline of a New Approach to the Analysis of Complex Systems and Decision Processes," *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3, pp. 28-44, 1973.
15. Klir, G.J. and Folger, T.A., *Fuzzy Sets, Uncertainty, and Information*, Englewood Cliffs, N.J., Prentice-Hall, 1988.

Appendix A

The following sections describe in detail the procedure used to cluster new associations of condition/effect objects. The term "cluster centroid" and MEMORY_OBJECT is used interchangeably.

A. Step 1: Initialize thresholds

A global *expansion* threshold T_e (Eq. 1) is used to determine when a new input condition object should become its own cluster. This controls how easily new clusters are created. The threshold is usually initialized to a value close to 1 (e.g. 0.9). A high value makes cluster creation easier since even slightly dissimilar objects will generate a new cluster. The expansion threshold is adjusted each time a new cluster is created or a cluster is replaced. The adjustment is done as follows:

$$T_e = T_e - \left(\alpha T_e e^{-\frac{1}{N}} \right) \quad (1)$$

where T_e is the expansion threshold, N =max(1, number of existing clusters) and α is the expansion decay. This is set to a small number in the order of 10^{-2}.

A maximum number of clusters can also be optionally set. This number, *MaxClusters*, determines when new clusters can not be added. Instead old clusters get replaced when a new input requires its own cluster. Clusters get replaced using the *relevance* value, R_j for cluster *j*:

$$R_j = (1 - N) + Iteration_{Cluster(j)} / Iteration_{Global} + MergeCount_{Cluster(j)} + E + O$$

where N is the number of active input properties, *Cluster.Iteration* is the iteration this cluster was last modified at and *Cluster.MergeCount* is the number of elements this cluster has. The Boolean variable *E is* set to one if the particular MEMORY_OBJECT has an Effect PROPERTY_OBJECT, zero otherwise. The Boolean variable *O* is set to one if Cluster.Cnd.Type == InputCnd.Type. The smaller the R_j, the more eligible cluster *j*

becomes for replacement. Justification for this equation was influenced by the task at hand and comes from the following reasoning:

> If a cluster has been used recently it is considered relevant to the current environment. Thus the last update iteration is used. If new associations have been merged several times this cluster, its considered "heavier" and it has a larger "inertia". Thus it becomes much harder for new knowledge to affect it unless its very similar to it. This justifies the MergeCount variable.

B. Step 2: Cluster New Memory Object M

When an input `MEMORY_OBJECT`, referred to as M_{input}, is presented to the CORE_LEARN function, the clustering algorithm uses the input condition value matrix ($M_{input}.Cnd.Values$) to calculate a membership function to each existing cluster condition object.

The input matrix $M_{input}.Cnd.Values$ is represented as matrix U. Matrix U is of size MxK, where

- M is the maximum number of properties that can be concurrently sensed in a single stimulus
- K is the number of elements in the largest vector property.

The cluster matrix *Cluster(j).Cnd.Values* is represented as $V_j \in R^{Nc}$. Using fuzzy inference the membership value of U_{input} for each cluster ***j*** is calculated as follows:

$$m_j = p_j \prod_{k=1}^{N} \left(\prod_{l=1}^{R} \exp\left(-\frac{u_k(l) - v_k(l)}{r_k(l)} \right) \right) \quad (2)$$

Where

- $r(j)_k$ is the range for element *l* of property vector *k*

- N is the number of properties common between the Cluster condition and the input condition
- M is the number of properties in the Cluster condition. Always M >= N
- p_j is the Partial Information multiplier described below

Note that the cluster condition property vector v_k is used to define the support for the membership function for each input property *k*.
The crisp output value matrix is generated using:

$$O = \sum_{j=1}^{N} m_j Effect_j \Big/ \sum_{j=1}^{N} m_j \quad (3)$$

$Effect_j$ is the matrix formed from the active *N* property row vectors, from the `EffectObject.PropertyValues` array of cluster j .

Since the clustering algorithm has to account for variable length input vectors, only the same properties active in both the input condition and the cluster condition are compared. When objects of different information content are compared to each other, $M \neq N$, a fuzzy value is calculated to adjust the membership:

$$p_j = \exp\left(-\frac{M-N}{M}\right).$$

The above equation assigns equal weight to all properties. Optionally the weight can be adjusted by determining the relevance of each observed property.

C. Cluster Expansion/Creation

The decision to merge the new input with an existing cluster or create a new cluster is done as follows:

IF $m_{min} < T_{expansion}$ AND $N <$ *MaxClusters* THEN

Create new cluster

ELSE IF $m_{min} < T_{expansion}$ AND $N >= MaxClusters$

Replace cluster j, $m_{min} = m_j$ with new *MEMORY_OBJECT* M_{input}.

ELSE

Add new MEMORY_OBJECT M_{input}, to *cluster j*, where $m_{min} = m_j$.

When adding matrix M_{input} to cluster j, *Cluster (j).Cnd* and *Cluster(j).Effect* property value matrices are adjusted to reflect the information content of the new member. The adjustment formula for value matrix U is given below:

$U_{i,new} = U_{j, old} - U_{j, old} * m + U_{j, input} * m$

where m_j is the combined membership value of M_{input} to cluster j.

Appendix B

A detailed description of the sonar performance maps and the probability formulation used by the simulation is provided here.

Sonar performance prediction maps

The Org requires sonar performance predictions in order for it to assess how well it can detect targets as it moves through the environment. For this work, CASS (Comprehensive Acoustic System Simulation) was used to generate signal excess (SE) predictions. CASS takes as input descriptions of the environment (sound speed profile, bathymetry, bottom composition, wind speed, etc.) and the sonar system (transmit pulse, beam pattern, sonar depth, etc.), and calculates a signal to interference ratio (SIR). The signal portion is the echo level returned from a hypothetical target, and the interference is a power sum of ambient noise and reverberation levels. Signal excess is then the number of dB the SIR is above the detection threshold (in this case, DT = 12 dB). For this part of the calculation, a target strength of 0 dB is assumed.

By hypothesizing target locations at an array of depths and ranges, a so-called "signal excess map" can be generated. An example is shown in Fig. 17. For the environment under consideration, mild bathymetry and sound speed inhomogeneities have been introduced throughout the area in order to produce position-dependent performance for the Org. CASS runs were performed for platform locations every 4 km in the XY plane, with a SE map being generated for look directions spaced every 90 degrees in azimuth.

The following approximation was used to convert signal excess to probability of detection. Assuming a uniform probability of observing a given target class at a given aspect, a generic target strength probability density function was generated. Then, the approximation is made that SE>0 produces probability of detection equal to 1 and SE<0 produces probability of detection equal to 0. The target strength probability density function is added to the SE value of a range-depth cell in the SE map to produce a distribution of "true" SE, and this function is integrated upwards from SE=0 to produce an

estimated overall probability of detection. This is obviously a rather crude estimate for probability of detection, but since the behavior of the Org was the main item of interest for this study, it was desirable to keep these computations as simple as possible.

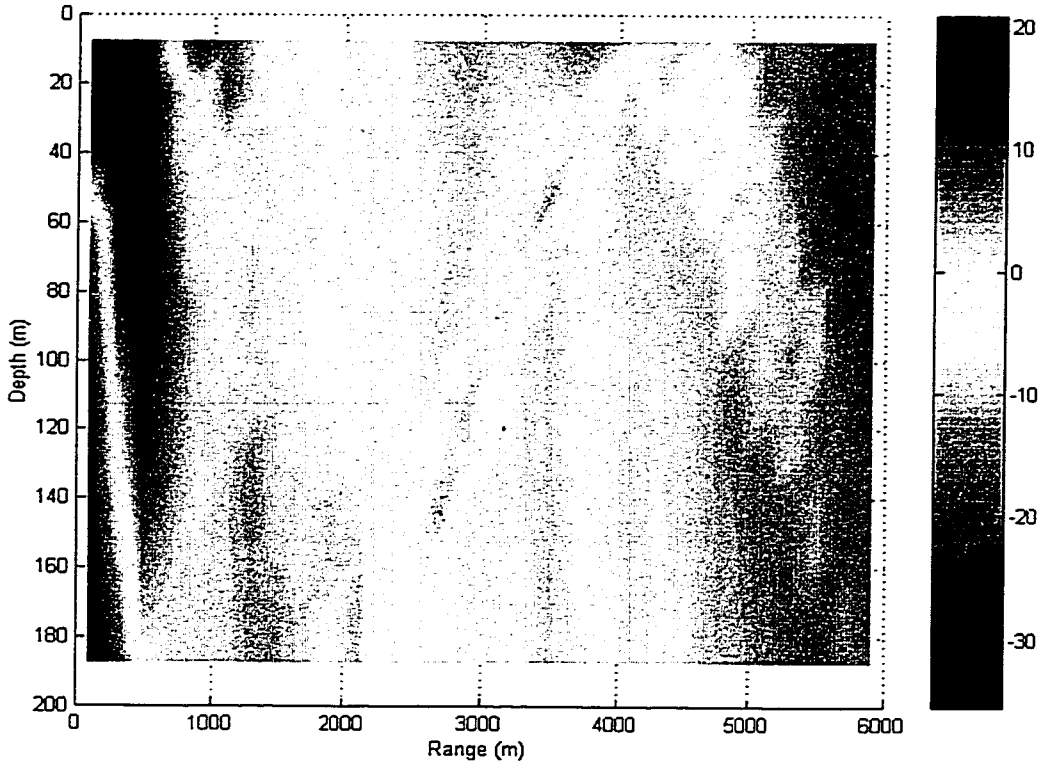


Figure 17. Sample sonar performance map for a single look direction. Color is in dB.

The value of signal excess (and the associated probability of detection) changes for each stimulus depending on the Org's distance and orientation to the stimulus under consideration. This adds a challenge to the path-planning problem since the agent's perception of the environment changes as it moves. For example the same grid locations (and the stimuli representing them) will look entirely different to the Org if it approaches them from two different angles.

Probability formulation

The formulation of the Org property that describes the cumulative detection probability is the following. Consider establishing a 3-D grid of reference points in the ocean and assign an *a priori* probability of a target being present at each point, $P_T(x,y,z)$. For the purposes of pre-engagement path planning, we assume that as the Org makes its way through the search area, no detections will be made. After the first ping, we can form the *a posteriori* probability of a target being present in a given cell given that no detection has been made using Bayes' rule:

$$P_{T \mid ND} = (P_{ND \mid T} * P_T) / (P_{ND \mid T} * P_T + P_{ND \mid NT} * P_{NT}),$$

where *T* denotes "target present," *ND* denotes "no detection," and *NT* denotes "no target present."

Table 2 illustrates how the probabilities are calculated, expressed as the probability of making a target present/not present decision conditioned on the actual state of the cell being interrogated. P_d is the probability of detection calculated from the relevant SE map, and P_{fa} is the probability of false alarm as specified by the operating point. P_{fa} is usually specified as a very small number (e.g., 10^{-5}). The *a posteriori* probability of a target being present in a cell is then:

$$P_{T \mid ND} = ((1 - P_d) * P_T) / ((1 - P_d) * P_T + (1 - P_{fa}) * (1 - P_T)).$$

This quantity then provides the *a priori* probability of target presence (P_T) for the next ping. One of the Org's overall goals is to minimize this quantity at all points in the search area.

Table 4 Decision of probability calculation based on target presence

		Decision	
		Target Present	No Target Present
Cell Status	Target Present	P_d	1 - P_d
	No Target Present	P_{fa}	1 – P_{fa}

Appendix C

The ORG is implemented in the MathWorks Inc. MATLAB scripting language Version. 5. The code greatly resembles 'C' and is included here as a reference and explanation tool for the ORG. The code exists in several modules and is presented here under sections named after m-file module it appears in. There is no particular ordering of the sections.

ORG Software Implementation (m-file code)

StartOrg.m

```
InitLevel = 0;
%
% starts the simulation
% first initialize global data and ORG settings
%

if InitLevel < 1
   clear global;
end;

globdecl;
initdata(InitLevel);

if InitLevel < 1

   %
   % if a variable "rst" is 0, then we initialize all structures
   %

   SIM.DebugLevel = 2;
   instinct;

end;


if InitLevel == -1

   figure(SIM.Visual.ProbabilityFigNum);
   clf;

   h=title('Click once for starting position, then again for
direction');
   set(h,'Color',[1 1 1]);
   set(gcf,'Color',[ 0.1 0.1 0.1]);
```

```
    set(gca,'Color',[ 0 0 0]);
    set(gca,'XColor',[ 1 1 1]);
    set(gca,'YColor',[ 1 1 1]);
    set(gca,'ZColor',[ 1 1 1]);

    axis ([0 SIM.Env.Grid(1,1) 0 SIM.Env.Grid(2,1)]);
    axis normal;

    [x,y] = ginput(2);

    ORG.Current.S.Values(POS,1:3) =[1000*floor(x(1)) 1000*floor(y(1)) 0];

    dx = x(2)*1000 - ORG.Current.S.Values(POS,1);
    dy = y(2)*1000 - ORG.Current.S.Values(POS,2);

    [th,mag] = cart2pol(dx,dy);

    %
    % restrict velocity vector to maximum magnitude
    %

    mag = SIM.ORG.PropertyObject.RangeLo(DIST+1,1);

    % convert back to cartesian
    [x y] = pol2cart(th,mag);

    if sign(x) ~= sign(dx)
       x = -x
    end;

    if sign(y) ~= sign(dy)
       y = -y
    end;

    ORG.Current.S.Values(POS+1,1:2) = [x y];

end;

if InitLevel < 1

    %
    % pre initialize every virtual target
    %

    envmake;

end;

orgsim;
```

Module Initdata.m

```
function initData(InitLevel)

% GLOBAL DEFINITIONS USED IN ALL M-FILES FOR THIS PROJECT

globdecl;

if InitLevel < 1

   dbstop if error;
   dbstop if warning;

   globdecl;

   SIM = [];
   ORG = [];

   %
   % Load non isomoprhic enviroment simulation data
   %

   load seNonIsoEnv.mat;

   %
   % pre loaded static enviroments are independent of sonar depth or
   % other parameters
   % they are just 3d arrays that match exactly our grid
   %

   %load seNSRidge.mat;

   %
   % load isomorphic enviroment, only rnage and depth are supplied
   %

   %load se.mat


   %
   % load data describing SE data format for a isomorphic enviroment
   % row 1: Range data points, interval, starting range
   % row 2: Depth data points, interval, starting range
   % row 3: Sonar Depth data points, interval, starting range
   % for non isomorphic enviroments
   % row 4: Orientation of Receive sonar Azimuth data point, interval,
   % starting range
   % row 5: X Grid data points, interval , starting point
   % row 6: Y grid same as above
```

```
% for non isomorphic Enviroment
% There are 448 lines and 395 numbers per line.  The first number in
% each line is the simulation version number and will always be 0 for
% this file.  The second number is x location (2, 6, 10, or 14 km),
% the third is y location (2, 6, 10, or 14 km), the fourth number is
% source depth (10, 30, 50, 70, 90, 110, 130, or 150 m), and the
% fifth number is orientation of the receive beam azimuth
% (0 deg [north], 90 deg [west], 180 deg [south], or 270 deg [east]).
% The next 390 numbers are SE for a 30 x 13 grid of hypothetical
% target locations in range and depth.
%


SIM.Env.SE = SEFormat;

clear SEFormat;

load TsTable.mat
SIM.Env.TsTable =  TsTable;

clear TsTable;

SIM.Env.Size = [16000 16000 195];
SIM.Env.SeaLevel = 0;
SIM.Env.AprioriProb = 0.7;

%
% grid points represent the Virtual Targets placed in the simulation
% Grid(1,1) = number of X elements
% grid(2,1) = number of Y elements
% grid(3,1) = number of discrete depths

SIM.Env.Grid = [16 1000 0 1;16 1000 0 1; 4 40 10 2 ];
% grid type cartesian

SIM.Visual.Flags.Text = 1;
SIM.Visual.Flags.DrawStimuli = 0;

SIM.Visual.Replay.DrawSeMaps = 0;
SIM.Visual.Replay.DrawAvgProb = 1;
SIM.Visual.Replay.OverlayPath =1;

SIM.Visual.EnvFigNum= 1;
SIM.Visual.PdStatusFigNum= 2;
SIM.Visual.SeMapFigNum= 3;
SIM.Visual.ProbabilityFigNum = 4;
SIM.Visual.CoreMemoryFigNum = 5;
SIM.Visual.ProbVsIterationFigNum = 6;

SIM.Visual.ObjectText.Color = [ 0 1 1];
```

```
SIM.Visual.ObjectText.FontSize = 9;

PLOT_VIRTUAL_TARGET = 1;
PLOT_ORG =2;

CORE_LEARN = 1;
CORE_REACT = 2;
CORE_MEMORIZE = 3;
CORE_RECALL = 4;

SIM.ORG.Constants.Speed = 4.44; % 10 knots in meters/sec, BUGBUG;
SIM.ORG.Constants.Elevation = SIM.Env.SeaLevel;

SIM.ORG.StateTrajectory = [];
SIM.ORG.LtCheckInterval = 1;

%
% simulation parameters
%

SIM.SpeculationAltUsed = 0;

% seconds per iteration; This is set so high so our sim moves fast

SIM.TimeScale = 90;
SIM.MaxDebugLevel = 4;
SIM.DebugLevel = 1;
SIM.MaxIterations = 302;
SIM.ProbSaveInterval = 25;

SIM.Env.ElementArray = [];

%
% NOTE!!!!!!!!
% the flags below determine the behavior of the ORG since the
% select the mode of operation for longterm stimuli and for reactive
% input
%

%
% use pre-loaded list of long term goals
%

SIM.ORG.Flags.StaticLtGoals = 0;

%
% completely shut off the reactive sensory interface
% only long term goals are considered
%
```

```
SIM.ORG.Flags.DisableReactiveInput = 0;

%
% generate reaction to next state transition, modify it
%

SIM.ORG.Flags.SelfReflect = 1;

%
% enable/disable speculation
%

SIM.ORG.Flags.Speculate = 1;

%
% allow new rules appliedto external stimuli to be generated
% during speculation
%

SIM.ORG.Flags.SpeculateNewRules = 1;

%
% attempt to learn from changes imposed on suggested state
% transitions by the enviroment filter
%

SIM.ORG.Flags.LearnFromEnvironmentalEffects = 1;

%
% use this matrix to specify any false target locations. These
% locations will have the property RxSS set adn their probability
% will always remain high
%

SIM.Env.FalseTargetIndex = [];
%[ 5 5 1; 1 3 1; 10 5 1];

%
% this value determines how many stimuli we consider in each teration
% to generate our cumulative reactive feedback. Changing it has major
% implications to the org path
%

SIM.ORG.MaxRtWinners = 10;

%
% If the org senses any less stimuli than the number before
% it will use any memorized stimuli to generation a reaction
%
```

```
    SIM.ORG.MinRtStimuli = 2;

    %
    % set the physical contraints for this ORG
    % sensor range is dependent on direction. We have directional bias
    % due to the hydrophone array
    %

    SIM.ORG.Constraints.SensorRange(1:180) = 16000;%max(SIM.Env.Size);
    SIM.ORG.Constraints.SensorDirectionalGain(1:10) = [-9.3 -5.2 -2.8...
       -1.3 -0.2 0.5 0.9 1.2 1.5 1.7];

    SIM.ORG.Constraints.StimulusUpdateThreshold = 3000;

    %
    % number of iterations to speculate ahead
    % the speculation duraton must be long enough that new objects come
    % within sensor range
    %

    SIM.ORG.SpeculationDuration =  ceil(min(SIM.Env.Grid(1:2,2)) /
       (SIM.ORG.Constants.Speed*SIM.TimeScale));
    SIM.ORG.SpeculationInterval = 5;


end;

SIM.Env.ElementIndex = zeros(60,60,6);

SIM.Current.ProbMap = [];
SIM.Current.Iteration = 1;
SIM.StateTrajectory = [];

if InitLevel < 1

    %%%%%%%%%%%% Properties. The variable correpsonds to a row index in
    % an array initial properties observable in self
    % Each property has a derivative, which is the next column index
    % following the property
    %

    t=1;

    SIR = t; t=t+2;
    SIM.ORG.PropertyDescription(SIR).S = 'SIR';
    SIM.ORG.PropertyDescription(SIR+1).S = 'delta SIR';

    PROB = t; t=t+2;
    SIM.ORG.PropertyDescription(PROB).S = 'Probability';
    SIM.ORG.PropertyDescription(PROB).S = 'Delta Probability';
```

```
SPECULATE = t; t=t+2;
SIM.ORG.PropertyDescription(SPECULATE).S = 'SPECULATE';
SIM.ORG.PropertyDescription(SPECULATE+1).S = 'delta SPECULATE';


POS = t; t=t+2;
SIM.ORG.PropertyDescription(POS).S = 'Position';
SIM.ORG.PropertyDescription(POS+1).S = 'delta Position';


DIST = t; t=t+2;
SIM.ORG.PropertyDescription(DIST).S = 'Distance';
SIM.ORG.PropertyDescription(DIST).S = 'delta Distance';

THETA = t; t=t+2;
SIM.ORG.PropertyDescription(THETA).S = 'THETA';
SIM.ORG.PropertyDescription(THETA+1).S = 'delta THETA';


ITER = t; t=t+2;
SIM.ORG.PropertyDescription(ITER).S = 'Iteration';
SIM.ORG.PropertyDescription(ITER+1).S = 'Iteration delta';

RxSS = t; t=t+2;
SIM.ORG.PropertyDescription(RxSS).S = 'Receive Signal Strength';
SIM.ORG.PropertyDescription(RxSS+1).S = ...
   'Receiver Signal Strength delta';

NUM_OBJECTS = t; t=t+2;
SIM.ORG.PropertyDescription(NUM_OBJECTS).S = ...
   'Number of Stimuli considered';
SIM.ORG.PropertyDescription(NUM_OBJECTS+1).S = ...
   'delta on NUM_OBJECTS not valid';

SNR_DEPTH = t; t=t+2;
SIM.ORG.PropertyDescription(SNR_DEPTH).S = 'SONAR_DEPTH';

MAX_PROP_IDX = t;

SIM.Visual.ColorMap(SIR,:,:) = hsv(512);
SIM.Visual.ColorMap(PROB,:,:) = jet(512);

%
% types of membership functions
%

MEMBERSHIP_FUNCTION_GAUSSIAN = 1;
MEMBERSHIP_FUNCTION_TRIANGULAR = 2;
MEMBERSHIP_FUNCTION_STEP = 3;
```

```
SIM.Env.NumProperties = t;
SIM.Env.MaxPropertyVectorSize = 4;

%
% values corresponding to sets
% Some of them (like use currect and RANDOM are actually flags that
% will promt the fuzzy sytem to replace them with real values...
%

SET_LARGE_NEG = -1;
SET_SMALL_NEG = -0.25;
SET_MEDIUM_NEG = -0.5;
SET_ZERO = 0.01;
SET_SMALL_POS = 0.25;
SET_MEDIUM_POS = 0.5;
SET_LARGE_POS = 1;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% behavior objects are abstractions of groups of actions
% Their generality allows for describing complex sequences
% without explicitly stating the state transitions.
% those are to be discovered of aided by instinct...
% all element are actually degree of change values..
%

%
% Knowledge representation, memory
% Each knowledge element can be pointed to by other elements.
%


%
% object that hold properties
% Index is a row with ones corresponding to the properties that
% are enabled in this object.
% Size is a row corresponding to a property
% Values is a matrix, with row == Property,
% Column = Property vector comp.
%

STIMULUS_EXTERNAL = 1;
STIMULUS_INTERNAL = 2;
STIM_LT = 4;
STIM_RT = 8;

po.Type = bitor(STIMULUS_EXTERNAL,STIM_RT) ;
po.Index = zeros(1,MAX_PROP_IDX);
```

```
po.Values = zeros(t,3);

%
% save a limited version of the property object for general use
%

PROPERTY_OBJECT = po;

%
% further define more fields only referenced in one place
%

po.Size(1,[SIR SIR+1 PROB PROB+1]) = [1 1 1 1];

po.Size(1,[DIST DIST+1]) = [1 1];
po.Size(1,[POS POS+1 ]) = [ 3 2 ];
po.Size(1,[ITER ITER+1]) = [1 1];
po.Size(1,[SNR_DEPTH RxSS RxSS+1]) = [1 1 1];
po.Size(1,THETA) = 1;
po.Size(1,SPECULATE) = 1;
po.Size(1,[NUM_OBJECTS NUM_OBJECTS+1]) = [1 1];

%
% define property ranges
%

po.RangeLo(1,1:max(max(po.Size))) = 0;
po.RangeHi(1,1:max(max(po.Size))) = 0;

%
% lower bound
%

po.RangeLo(SNR_DEPTH,1) = SIM.Env.SE.Format(3,3) - ...
   SIM.Env.SE.Format(3,1)*SIM.Env.SE.Format(3,2);

po.RangeLo(PROB,1) = [0 ];
po.RangeLo(PROB+1,1) = 0 ;

po.RangeLo(RxSS,1) = [0];

po.RangeLo(NUM_OBJECTS,1) = 1;
po.RangeLo(NUM_OBJECTS+1,1) = 0;

po.RangeLo(THETA,1:po.Size(THETA)) = 0;
po.RangeLo(POS,1:po.Size(POS)) = 0;
po.RangeLo(POS+1,1:po.Size(POS+1)) = 0;
po.RangeLo(DIST,1:po.Size(DIST)) = 0;
po.RangeLo(SPECULATE,1) = 0;
```

```
s =SIM.ORG.Constants.Speed*SIM.TimeScale;

po.RangeLo(DIST+1,...
   1:po.Size(DIST+1)) = s;

if SIM.Env.SE.Flags.Preloaded3dEnviroment

   po.RangeLo(SIR,1) = min(min(min(SIM.Env.SE.Data(:,:,:))));

else

   po.RangeLo(SIR,1) = ...
      min(min(SIM.Env.SE.Data(:, ...
         SIM.Env.SE.NumInputs+1:size(SIM.Env.SE.Data,2) )));

end;

po.RangeLo(ITER,1) = 0;
po.RangeLo(ITER+1,1) = 0;

%
% high bound
%

po.RangeHi(SNR_DEPTH,1) = SIM.Env.SE.Format(3,3);

po.RangeHi(RxSS,1) = [1];
po.RangeHi(SPECULATE,1) = SET_LARGE_POS;

po.RangeHi(NUM_OBJECTS,1) = 15;
po.RangeHi(NUM_OBJECTS+1,1) = 5;


po.RangeHi(PROB,1) = 1 ;
po.RangeHi(PROB+1,1) = 1 ;

po.RangeHi(POS,1:po.Size(POS)) = SIM.Env.Size(1:po.Size(POS));
po.RangeHi(POS+1,1:po.Size(POS+1)) = SIM.Env.Size(1:po.Size(POS+1));

po.RangeHi(DIST,1:po.Size(DIST)) = ...
   max(SIM.ORG.Constraints.SensorRange);
po.RangeHi(DIST+1,1:po.Size(DIST+1)) = s;
po.RangeHi(THETA,1:po.Size(THETA)) = 180;

if SIM.Env.SE.Flags.Preloaded3dEnviroment

   po.RangeHi(SIR,1) = max(max(max(SIM.Env.SE.Data(:,:,:))));

else
```

```
        po.RangeHi(SIR,1) = max(max(SIM.Env.SE.Data(:, ...
            SIM.Env.SE.NumInputs+1:size(SIM.Env.SE.Data,2) )));

    end;

    po.RangeHi(ITER,1) = SIM.MaxIterations;
    po.RangeHi(ITER+1,1) = SET_LARGE_POS;

    po.MbrFunction = ones(1,t)*MEMBERSHIP_FUNCTION_GAUSSIAN;
    po.Sigma = ones(1,MAX_PROP_IDX)*20;
    po.Sigma(PROB) = 5;

    SIM.ORG.PropertyObject = po;

    MEMORY_OBJECT.P = [];
    MEMORY_OBJECT.Effect = [];
    MEMORY_OBJECT.MergeCount = 0;
    MEMORY_OBJECT.Iter = 0;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %
    % structure describing the virtual targets at each grid point
    %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    VIRTUAL_TARGET = PROPERTY_OBJECT;
    VIRTUAL_TARGET.Index([DIST POS PROB ITER THETA+1 SIR ]) = 1;

    VIRTUAL_TARGET.Values(PROB) = 0;

    FALSE_TARGET = PROPERTY_OBJECT;
    FALSE_TARGET.Index([DIST POS PROB ITER SIR RxSS]) = 1;

    SIM.Env.ElementType = VIRTUAL_TARGET;


end;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% structure describing the organism
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%
% ORG.Current.Cnd is the current state of this ORG.
% ORG.Current.Effect is the transition vector on all active properties
% of the org.
% By operating on ORG.Current.Cnd with ORG.Current.Effect, you get the
```

```
% new ORG.Current.Cnd
%

ORG.Num = 1;

if InitLevel < 1

   ORG.StateTrajectory = [];
   ORG.Core.Memory = [];
   ORG.Core.Constraints.Memory(1).MergeThreshold = 1;
   ORG.Core.Constraints.Memory(1).MaxSize = 10;
   ORG.Core.Constraints.Memory(1).Size = 0;

   %
   % long term memory
   %

   ORG.Core.Constraints.Memory(2).MaxSize = 5;
   ORG.Core.Constraints.Memory(2).Size = 0;

   ORG.Core.Constraints.Memory(2).MergeThreshold = 1;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% insert longterm goals if any
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   if SIM.ORG.Flags.StaticLtGoals == 0

      SIM.ORG.NumStaticLtGoals = 0;

   else

      i=1;
      pi = MEMORY_OBJECT;
      pi.P = PROPERTY_OBJECT;
      pi.P.Type = bitor(STIMULUS_EXTERNAL,STIM_LT);
      pi.P.Index(1,[POS PROB DIST ])  = [ 1 1 1 ] ;

      %
      % load path from file. Its grid must much the current grid..
      % replace filename below with appropriate file
      %

      load ladderPath3legs;

      SIM.ORG.NumStaticLtGoals = size(ltArray,1);
```

```
    %
    % adjust long term memory size to accomodate the static long term
    % goal list
    %

    ORG.Core.Constraints.Memory(2).MaxSize = ...
       ORG.Core.Constraints.Memory(2).MaxSize + ...
       SIM.ORG.NumStaticLtGoals;

    for i=1:size(ltArray,1)

       pi.P.Values(PROB,1) = 1;
       pi.P.Values(POS,1:3) = ltArray(i,1:3);

       core(CORE_LEARN,pi);

    end;

    clear ltArray;

end;

ORG.Core.PropCorrMatrix=zeros(MAX_PROP_IDX,MAX_PROP_IDX);

ORG.Current.S = PROPERTY_OBJECT;
ORG.Current.Cause = [];
ORG.Current.RtCause = [];
ORG.Current.LtCause = [];
ORG.Current.Effect = [];
ORG.Current.LtEffect = [];
ORG.StateTrajectory = [];

ORG.Current.S.Index([SNR_DEPTH SPECULATE POS PROB PROB+1 ...
   POS+1 NUM_OBJECTS]) =[1 1 1 1 1 1 1];
ORG.Current.S.Type = STIMULUS_INTERNAL;

%
% define a set of properties that modifying them affects other
% properties
%

ORG.Core.ActiveParameters = [NUM_OBJECTS];
ORG.Core.MotorProperties = [POS+1];

%
% Starting position
%

ORG.Current.S.Values(SNR_DEPTH,1) = [SIM.Env.SE.Format(3,3)];
```

```
    ORG.Current.S.Values(POS,1:3) = [1000 1000 0];
    ORG.Current.S.Values(POS+1,1:3) = [ 0 0 0];

    ORG.Current.S.Values(NUM_OBJECTS) = SIM.ORG.MaxRtWinners;

end;
```

Globdecl.m

```
%
% define global variables
%

global FT_PER_NM;
global SIM;
global ORG;

global STIMULUS_EXTERNAL;
global STIMULUS_INTERNAL;
global STIM_RT;
global STIM_LT;

global PROPERTY_OBJECT;
global MEMORY_OBJECT;

global MAX_LINK_WEIGHT;

global POS ;
global DIST;
global PROB ;
global SIR;
global ITER;
global RxSS;
global NUM_OBJECTS;
global THETA;
global SNR_DEPTH;
global SPECULATE;

global MAX_PROP_IDX;
global MEMBERSHIP_FUNCTION_GAUSSIAN ;
global MEMBERSHIP_FUNCTION_TRIANGULAR ;
global MEMBERSHIP_FUNCTION_STEP ;

global SET_LARGE_NEG ;
global SET_MEDIUM_NEG;
global SET_SMALL_NEG ;
global SET_ZERO ;
global SET_SMALL_POS;
global SET_MEDIUM_POS;
```

```
global SET_LARGE_POS;

global PLOT_VIRTUAL_TARGET;
global PLOT_ORG;

global CORE_LEARN;
global CORE_REACT;
global CORE_RECALL;
```

Instinct.m

```
function instinct
% INSTINCTS initializes an ORG with its instincts

globdecl;

%
% NOTE: The actual value we store in the antecedent of the rule, is the
% support of the membership function. That support must live within the
% input range of this property Since we arelady have a bucnh of SET_XX
% defined we multiply the values of those (between -1 1) with the range
% of the proprty, to get a real value in that range
%

pI =  PROPERTY_OBJECT;

pI.Index(1,[PROB THETA+1 DIST])  = [ 1 1 1];
pI.Type = bitor(STIMULUS_EXTERNAL,STIM_RT);

pI.Values(PROB) = SET_LARGE_POS;
pI.Values(THETA+1) = SET_ZERO;
pI.Values(DIST) = SIM.ORG.Constraints.StimulusUpdateThreshold;

pO=  PROPERTY_OBJECT;
pO.Index(1,[DIST+1]) = [ 1 ];
pO.Values(DIST+1,1) = SET_LARGE_NEG;
pO.Values(PROB+1,1) = SET_LARGE_NEG;

m = MEMORY_OBJECT;

m.P = pI;
m.Effect = pO;
core(CORE_LEARN,m);


return;
```

Envmake.m

```
function envmake;

% ENVMAKE Creates a 3d or 2d matrix filled with virtual objects
% It then visualizes this environment. The parameters for the
% environment are in the SIM global structure
%

%
% initialize the size of the environment element matrix
%

globdecl;

for k=1:size(SIM.Env.FalseTargetIndex,1)

   x = SIM.Env.FalseTargetIndex(k,1);
   y = SIM.Env.FalseTargetIndex(k,2);
   z = SIM.Env.FalseTargetIndex(k,3);

   S = generatestimulus(x,y,z);

   %
   % false targets have an extra property active: Return Signal Strength
   %

   S.Values(RxSS,1) = 1;
   S.Index(RxSS) = 1;

   SIM.Env.ElementArray(x,y,z) = S;

end;


for x=1:SIM.Env.Grid(1,1)
   for y=1:SIM.Env.Grid(2,1)
      for z=1:SIM.Env.Grid(3,1)

         %
         % create evenly distributed point acros the volume of water
         % that defines the search space. Each cell center is used to
         % calculate the detection coverage this particular areas has
         % received
         %

         S = generatestimulus(x,y,z);

      end;
```

```
   end;

end;
```

GenerateStimulus.m

```
function [Stimulus] = generatestimulus(x,y,z)

globdecl;

Update = 1;

if (SIM.Env.ElementIndex(x,y,z) == 0)

   % initalize this element since we have not seen it before during this
   % simulation
   SIM.Env.ElementArray(x,y,z) = SIM.Env.ElementType;
   SIM.Env.ElementArray(x,y,z).Values(POS,1:3) = ...
      [x*SIM.Env.Grid(1,2)+SIM.Env.Grid(1,3) ...
         y*SIM.Env.Grid(2,2)+SIM.Env.Grid(2,3) ...
         -z*SIM.Env.Grid(3,2)+SIM.Env.Grid(3,3)];

   SIM.Env.ElementArray(x,y,z).Values(PROB) = SIM.Env.AprioriProb;
   SIM.Env.ElementIndex(x,y,z) = 1;

end;


%
% boundary check. Return no stimuli
%

if SIM.Env.ElementArray(x,y,z).Values(POS,1) > SIM.Env.Size(1,1)
   Stimulus = [];
   return;
end;

if SIM.Env.ElementArray(x,y,z).Values(POS,2) > SIM.Env.Size(1,2)
   Stimulus = [];
   return;
end;


SIM.Env.ElementArray(x,y,z).Values(ITER+1) = SET_LARGE_POS -
(SIM.Env.ElementArray(x,y,z).Values(ITER)/SIM.Current.Iteration);
```

```
rpos = SIM.Env.ElementArray(x,y,z).Values(POS,1:3)-
ORG.Current.S.Values(POS,1:3);

SIM.Env.ElementArray(x,y,z).Values(DIST) = ...
   sqrt(sum(rpos.^2,2));

if SIM.Env.ElementArray(x,y,z).Values(DIST) >
SIM.ORG.Constraints.StimulusUpdateThreshold
   Update = 0;
end;

%
% calculate angle from ORG to virtual target.
%

spos = SIM.Env.ElementArray(x,y,z).Values(POS,1:2) - ...
   ORG.Current.S.Values(POS,1:2);

% convert sonar position to polar

[sTh,sR] = cart2pol(spos(1,1),spos(1,2));

sThDeg = (sTh *180)/pi;
dThDeg = abs(ORG.Current.S.Values(THETA,1) - sThDeg);

if dThDeg > 180
   dThDeg = 360 - dThDeg;
end;

% dont care if angle deviation is positive or negative (to our left or
% right)
SIM.Env.ElementArray(x,y,z).Values(THETA+1,1) = dThDeg;

if Update
   SIM.Env.ElementIndex(x,y,z) = SIM.Current.Iteration;
   SIM.Env.ElementArray(x,y,z).Values(ITER) = SIM.Current.Iteration;
end;

%
% calculate SIR value for this Stimulus and at the current sonar depth,
% bearing
%

range =
min(SIM.Env.SE.Format(1,1),max(1,round((SIM.Env.ElementArray(x,y,z).Valu
es(DIST) - SIM.Env.SE.Format(1,3))/...
   SIM.Env.SE.Format(1,2))));
```

```
depth =
min(SIM.Env.SE.Format(2,1),max(1,abs(round(SIM.Env.ElementArray(x,y,z).V
alues(POS,3)/SIM.Env.SE.Format(2,2)) )));

snr_depth =
min(SIM.Env.SE.Format(3,1),1+abs(round((ORG.Current.S.Values(SNR_DEPTH)-
SIM.Env.SE.Format(3,3))/...
   SIM.Env.SE.Format(3,2)) ));


if SIM.Env.SE.Flags.Preloaded3dEnviroment

   %
   % pre loaded environments are independent of sonar depth or other
   % parameters
   % they are just 3d arrays that match exactly our grid
   %

   SIM.Env.ElementArray(x,y,z).Values(SIR) = ...
      SIM.Env.SE.Data(x,y,z);

elseif SIM.Env.SE.Flags.IsomorphicEnviroment

   SIM.Env.ElementArray(x,y,z).Values(SIR) = ...
      SIM.Env.SE.Data(snr_depth,SIM.Env.SE.NumInputs+(depth-
1)*SIM.Env.SE.Format(1,1) + range);

else

   %
   % For non isomorphic enviroments we also have to find the closest x,y
   % location
   % we have data for and also find the bearing to that point
   % first interpolate on closest x,y data point
   %

   se_x = min(SIM.Env.SE.Format(5,1),...
      1+abs(round((ORG.Current.S.Values(POS,1)-
SIM.Env.SE.Format(5,3))/...
      SIM.Env.SE.Format(5,2)) ));

   se_y = min(SIM.Env.SE.Format(6,1),...
      1+abs(round((ORG.Current.S.Values(POS,2)-
SIM.Env.SE.Format(6,3))/...
      SIM.Env.SE.Format(6,2)) ));

   %
   % now find bearing from the ORG to this virtual target
   %
```

```
   se_theta =
min(SIM.Env.SE.Format(4,1),1+abs(round((SIM.Env.ElementArray(x,y,z).Valu
es(THETA+1,1)-SIM.Env.SE.Format(4,3))/...
      SIM.Env.SE.Format(4,2)) ));

   %
   % now calculate the row index were the combinations of x,y,snr_depth,
   % bearing exists
   % Data is shorted first by X, then by Y, then by Snr Depth, then by
   % bearing
   %

   sz = size(SIM.Env.SE.Data,1)/SIM.Env.SE.Format(5,1);
   rowIndex = 1+ sz * (se_x-1);

   % further refine find y location
   sz = sz/SIM.Env.SE.Format(6,1);
   rowIndex = rowIndex + sz*(se_y-1);

   % snr_depth is calculated

   rowIndex = rowIndex + (snr_depth-1);

   % now bearing

   rowIndex = rowIndex + (se_theta-1);

   SIM.Env.ElementArray(x,y,z).Values(SIR) = ...
      SIM.Env.SE.Data(rowIndex,SIM.Env.SE.NumInputs+(depth-1) ...
      *SIM.Env.SE.Format(1,1) + range);

end;


%
% adjust the SIR value using direction gain
%

th = SIM.Env.ElementArray(x,y,z).Values(THETA+1,1);
if th > 90
   th = 180-th;
end;

if th < 0
   th
end;

index = floor(th/10)+1;
index2 = round(th/10)+1;
if index2 == th/10
```

```
   gainAdjust = SIM.ORG.Constraints.SensorDirectionalGain(index2);

else

   gainAdjust = (SIM.ORG.Constraints.SensorDirectionalGain(index2) + ...
      SIM.ORG.Constraints.SensorDirectionalGain(index))/2;

end;

SIM.Env.ElementArray(x,y,z).Values(SIR) =
SIM.Env.ElementArray(x,y,z).Values(SIR)+...
   gainAdjust;



%
% calculate Pd value if we are currently pinging.
% we use the already tabulated 1-F(ts) values fo the target strength/SE
% relationship
% This code represents an advanced sensor which uses information to
% calculate sensed values. The result is "derivative" sensory
% data
%

if Update

   Pt = SIM.Env.ElementArray(x,y,z).Values(PROB) ;

   ts = -SIM.Env.ElementArray(x,y,z).Values(SIR);

   if ts < SIM.Env.TsTable(1,1)

      Pd = 0.99;

   elseif ts > SIM.Env.TsTable(size(SIM.Env.TsTable,1),1)

      Pd = 0;

   else

      idx = find(abs(SIM.Env.TsTable(:,1) - ts) <=0.25);
      Pd = SIM.Env.TsTable(idx(1,1),2);

   end;

   if SIM.Env.ElementArray(x,y,z).Index(RxSS)

      %
      % return signal registered for this location. Take that into
```

```
        % account when calculating probability
        % BUBUG for now set Pd = 0 if RxSS active
        %

        Pd = 0.0;

    end;

    if SIM.Current.Iteration > 1

        %
        % use bayesian conditional probability..
        %

        Pfa = 0.00001;

        D = (1-Pd)*Pt + (1-Pfa)*(1-Pt);
        if D == 0
           dbgbreak;
        end;

        newProb =  ((1-Pd)*Pt) / D;

        SIM.Env.ElementArray(x,y,z).Values(PROB+1) = ...
           SIM.Env.ElementArray(x,y,z).Values(PROB) - ...
           newProb;

        SIM.Env.ElementArray(x,y,z).Values(PROB) = newProb;

    end;

end;

SIM.Env.ElementArray(x,y,z).Index(POS) = 1;
SIM.Env.ElementArray(x,y,z).Index(POS+1) = 0;

Stimulus = SIM.Env.ElementArray(x,y,z);
return;
```

Orgsim.m

```
function orgsim
%
% 3-d dimension navigation using autonomous Agents with instincts and
% adaptive unsupervised learning and speculation facilities
%

%
% define and initialize global vars
%

globdecl;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%

%
% This is the central loop that takes the organism
% through each iteration of its life.
% At each iteration actions and sensory input is recorded in a history
% matrix also its action are recorded
%


while SIM.Current.Iteration < SIM.MaxIterations

   %
   % speculate on alternate state transitions by creating random
   % rules and applying them for a limited number of iterations
   %

   speculation;

   %
   %  record state for each iteration
   %

   c.S = ORG.Current.S;
   c.Cause = ORG.Current.Cause;
   c.LtCause = ORG.Current.LtCause;
   c.RtCause = ORG.Current.RtCause;
   c.Effect = ORG.Current.Effect;
   c.LtEffect = ORG.Current.LtEffect;


   ORG.StateTrajectory(SIM.Current.Iteration) = c;
   SIM.StateTrajectory(SIM.Current.Iteration) = SIM.Current;

   %
```

```
% take an enviroment snapshot
%

if (mod(SIM.Current.Iteration, SIM.ProbSaveInterval) == 0) | ...
   (SIM.Current.Iteration ==2)

   prob = 0; i= 0;
   for x=1:SIM.Env.Grid(1,1)
      for y=1:SIM.Env.Grid(2,1)
         for z=1:SIM.Env.Grid(3,1)
            if  SIM.Env.ElementIndex(x,y,z) > 0

               SIM.StateTrajectory( ...
               SIM.Current.Iteration).ProbMap(x,y,z)= ...
               SIM.Env.ElementArray(x,y,z).Values(PROB);

               prob = prob + ...
                  SIM.StateTrajectory(...
                  SIM.Current.Iteration).ProbMap(x,y,z);

               i=i+1;

            end;

         end;
      end;
   end;

   prob = prob/i;

else

   prob = ORG.Current.S.Values(PROB);

end;

%
% certain properties and property information must be updated
% per iteration
%

SIM.ORG.PropertyObject.RangeHi(ITER) = SIM.Current.Iteration;
ORG.Current.S.Values(PROB+1) = prob-ORG.Current.S.Values(PROB);
ORG.Current.S.Values(PROB) = prob;

%
% generate the total reactive output from all the current stimuli
%

sense([]);
```

```
    longterm(ORG.Current.Effect,0);

    nextState = decision;
    dbgprint(1,sprintf('x = %d, y = %d dx = %d, dy= %d', ...
       nextState.Values(POS,1),nextState.Values(POS,2),...
       nextState.Values(POS+1,1),nextState.Values(POS+1,2)));

    ORG.Current.S = nextState;

    SIM.Current.Iteration = SIM.Current.Iteration+1;

    %
    % print current status
    %

    dbgprint(1,sprintf('I = %d,  snr depth %d ', ...
       SIM.Current.Iteration,ORG.Current.S.Values(SNR_DEPTH,1)));


end;
```

Sense.m

```
function sense(Stim)


%
% This is the sensory module for an ORG
%

globdecl;

numSensed = 0;
considerMemorizedStimuli = 0;

%
% each member of the stimuli array are objects we sense in this
% iteration
%

rtStimuli = MEMORY_OBJECT;
TotalEffect = PROPERTY_OBJECT;
TotalCnd = PROPERTY_OBJECT;

if SIM.Visual.Flags.DrawStimuli == 1

   figure(SIM.Visual.StimuliFigNum);
   clf;
```

```
    colormap hsv;
    grid;
    set(gcf,'Color',[ 0 0 0]);
    set(gca,'Color',[ 0 0 0]);
    set(gca,'XColor',[ 1 1 1]);
    set(gca,'YColor',[ 1 1 1]);
    set(gca,'ZColor',[ 1 1 1]);

end;

%
% only look at the volume cells within our sensor range
% find the "center cell" we are closer to. Essentially this is
% converting our coordinates
% so we look only at the grid points closer to us..
%

cellNum(1,1) =
max(floor(ORG.Current.S.Values(POS,1)/SIM.Env.Grid(1,2)),1);
cellNum(1,2) =
max(floor(ORG.Current.S.Values(POS,2)/SIM.Env.Grid(2,2)),1);

r =
max(SIM.ORG.Constraints.SensorRange)/min(SIM.Env.Grid(1,2),SIM.Env.Grid(
2,2));

sz = SIM.Env.Size(1,:) ./SIM.Env.Grid(:,2)';

for x=max(1,(cellNum(1,1)-r)):min(sz(1,1),cellNum(1,1)+r)
   for y=max(1,(cellNum(1,2)-r)):min(sz(1,2),cellNum(1,2)+r)

      tProb = 0;
      for z=1:SIM.Env.Grid(3,1)

         iter = SIM.Env.ElementIndex(x,y,z);

         if isempty(Stim)
            S = generatestimulus(x,y,z);
         else
            S = Stim;
         end;

         if ~isempty(S)

            idx = min(180,floor(S.Values(THETA+1,1))+1);

            numSensed = numSensed + 1;
            rtStimuli(numSensed) = MEMORY_OBJECT;
            rtStimuli(numSensed).P = S;
```

```
        rtStimuli(numSensed).Effect = [];

        if (z == 1) & (S.Values(DIST) > ...
           SIM.ORG.Constraints.SensorRange(idx))

           %
           % out of range. Restore object to its initial state
           % since its not considered sensed yet
           %

           numSensed = numSensed - 1;

           SIM.Env.ElementIndex(x,y,z) = iter;
           SIM.Env.ElementArray(x,y,z).Values(ITER) = iter;
           SIM.Env.ElementArray(x,y,z).Values(ITER+1) = ...
              (SIM.Current.Iteration - iter)/SIM.Current.Iteration;

           break;

        end;

        [rtStimuli(numSensed).Effect, mvIndex(numSensed)] = ...
           core(CORE_REACT,rtStimuli(numSensed));

        if ~isempty(Stim)
           break;
        end;

        %
        % track change in probability
        % BUGBUG: this is application specific. What property to
        % track depends on the ORG instincts and their intent.
        % The ORG PROB property is the cumulative probability from
        % all we just sensed..
        %

        ORG.Current.S.Values(PROB) = ORG.Current.S.Values(PROB)+ ...
           rtStimuli(numSensed).P.Values(PROB);

        ORG.Current.S.Values(PROB+1)=ORG.Current.S.Values(PROB+1)...
           + rtStimuli(numSensed).P.Values(PROB+1);


     else

        break;

     end;

end;
```

```
      if ~isempty(Stim)
         break;
      end;

   end;


end;

ORG.Current.S.Values(PROB) = ORG.Current.S.Values(PROB)/numSensed;
ORG.Current.S.Values(PROB+1) = ORG.Current.S.Values(PROB+1)/numSensed;


if SIM.ORG.Flags.DisableReactiveInput

   TotalEffect = PROPERTY_OBJECT;
   TotalCnd = PROPERTY_OBJECT;

elseif numSensed > 0

   %
   % sort the top performer based on the membership values
   % use only those to determine long term effect
   %

   [y,idx] = sort(mvIndex);

   %
   % convert sorted list to descending order since winners are with
   % highest membership values
   % Choose only top 20% of all stimuli, Unless otherwise specified
   %

   idx = fliplr(idx);
   sz = ORG.Current.S.Values(NUM_OBJECTS);

   if sz <= SIM.ORG.MinRtStimuli
      considerMemorizedStimuli = 1;
   end;

   TotalCnd.Index =  TotalCnd.Index | rtStimuli(1).P.Index;
   cndIx = find(TotalCnd.Index);

   for i=1:sz

      k  = idx(i);
      rtStimuli(k).Effect = applytoorg(rtStimuli(k));

      TotalEffect.Index = TotalEffect.Index | rtStimuli(k).Effect.Index;
```

```
        ix = find(rtStimuli(k).Effect.Index == 1);
        TotalEffect.Values(ix,:) = TotalEffect.Values(ix,:) + ...
           rtStimuli(k).Effect.Values(ix,:);

        TotalCnd.Values(cndIx,:) = TotalCnd.Values(cndIx,:) + ...
           rtStimuli(k).P.Values(cndIx,:);

    end;

    %
    % add most desireable short term stimulus to the CORE short term
    % memory
    % only add the condition portion. If we add Cnd + Effect it will be
    % treated as a learned rules
    %

    learnObject = MEMORY_OBJECT;
    learnObject.P = rtStimuli(idx(1)).P;

    %bugbug
    %core(CORE_LEARN,learnObject);

    %
    % average effect from all stimuli
    %

    idx = find(TotalEffect.Index == 1);
    TotalEffect.Values(idx,:) = TotalEffect.Values(idx,:)/sz;

    TotalCnd.Values(cndIx,:) = TotalCnd.Values(cndIx,:)/sz;


else

    considerMemorizedStimuli = 1;

end;

%BUGBUG
considerMemorizedStimuli = 0;

if considerMemorizedStimuli

    %
    % we did not sense anything within range.
    % So look back at the last few most attractive stimuli we had
    % encountered in the past. As a recall template uses the standard
    % stimulus
    % BUGBUG instead we should use our ORG.PropCorrMatrix to figure what
    % properties we need to have active in our template
```

```
    %

    template = MEMORY_OBJECT;
    template.P = SIM.Env.ElementType;

    recallList = core(CORE_RECALL,template);
    cndIdx = find(template.P.Index);

    idx = find(TotalEffect.Index == 1);

    if numSensed > 0

       %
       % average memorized stimulus reaction, with reactive stimulus
       % reaction
       %

       if ~isempty(recallList) & ~isempty(recallList(1).AverageEffect)

          TotalEffect.Values(idx,:) = ( TotalEffect.Values(idx,:)+ ...
             recallList(1).AverageEffect(idx,:) )/2;

          idx = find(TotalCnd.Index == 1);

          TotalCnd.Values(idx,:) = ( TotalCnd.Values(idx,:) + ...
             recallList(1).AverageCnd.Values(idx,:) )/2;

       end;

    else

       if ~isempty(recallList)
          TotalEffect = recallList(1).AverageEffect;
          TotalCnd = recallList(1).AverageCnd;
       end;

    end;


end;

%
% Store effect and cumulative averaged cause
%

ORG.Current.RtCause = TotalCnd;
ORG.Current.Cause = TotalCnd;
ORG.Current.Effect = TotalEffect;

return;
```

GenerateStimulus.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Stimulus] = generatestimulus(x,y,z)

globdecl;

Update = 1;

if (SIM.Env.ElementIndex(x,y,z) == 0)

   % initalize this element since we have not seen it
   % before during this simulation

   SIM.Env.ElementArray(x,y,z) = SIM.Env.ElementType;
   SIM.Env.ElementArray(x,y,z).Values(POS,1:3) = ...
      [x*SIM.Env.Grid(1,2)+SIM.Env.Grid(1,3) ...
         y*SIM.Env.Grid(2,2)+SIM.Env.Grid(2,3) ...
         -z*SIM.Env.Grid(3,2)+SIM.Env.Grid(3,3)];

   SIM.Env.ElementArray(x,y,z).Values(PROB) = SIM.Env.AprioriProb;
   SIM.Env.ElementIndex(x,y,z) = 1;

end;


%
% boundary check
%

if SIM.Env.ElementArray(x,y,z).Values(POS,1) > SIM.Env.Size(1,1)
   Stimulus = [];
   return;
end;

if SIM.Env.ElementArray(x,y,z).Values(POS,2) > SIM.Env.Size(1,2)
   Stimulus = [];
   return;
end;


SIM.Env.ElementArray(x,y,z).Values(ITER+1) = ...
   SET_LARGE_POS - ...
(SIM.Env.ElementArray(x,y,z).Values(ITER)/SIM.Current.Iteration);

rpos = SIM.Env.ElementArray(x,y,z).Values(POS,1:3)-...
   ORG.Current.S.Values(POS,1:3);
```

```
SIM.Env.ElementArray(x,y,z).Values(DIST) = ...
   sqrt(sum(rpos.^2,2));

if SIM.Env.ElementArray(x,y,z).Values(DIST) > ...
      SIM.ORG.Constraints.StimulusUpdateThreshold

   Update = 0;

end;

%
% calculate angle from ORG to virtual target.
%

spos = SIM.Env.ElementArray(x,y,z).Values(POS,1:2) - ...
   ORG.Current.S.Values(POS,1:2);

% convert sonar position to polar

[sTh,sR] = cart2pol(spos(1,1),spos(1,2));

sThDeg = (sTh *180)/pi;
dThDeg = abs(ORG.Current.S.Values(THETA,1) - sThDeg);

if dThDeg > 180
   dThDeg = 360 - dThDeg;
end;

% dont care if angle deviation is positive or negative
% (to our left or right)

SIM.Env.ElementArray(x,y,z).Values(THETA+1,1) = dThDeg;

if Update
   SIM.Env.ElementIndex(x,y,z) = SIM.Current.Iteration;
   SIM.Env.ElementArray(x,y,z).Values(ITER) = ...
      SIM.Current.Iteration;
end;

%
% calculate SIR value for this Stimulus and at the current
% sonar depth, bearing
%

range = min(SIM.Env.SE.Format(1,1),...
   max(1,round((SIM.Env.ElementArray(x,y,z).Values(DIST) - ...
   SIM.Env.SE.Format(1,3))/...
   SIM.Env.SE.Format(1,2))));
```

```
depth = min(SIM.Env.SE.Format(2,1),...
   max(1,abs(round(SIM.Env.ElementArray(x,y,z).Values(POS,3)/...
   SIM.Env.SE.Format(2,2)) )));

snr_depth = min(SIM.Env.SE.Format(3,1),1+...
   abs(round((ORG.Current.S.Values(SNR_DEPTH)-...
   SIM.Env.SE.Format(3,3))/SIM.Env.SE.Format(3,2)) ));


if SIM.Env.SE.Flags.Preloaded3dEnviroment

   %
   % pre loaded enviroments are independent of sonar depth
   % or other parameters they are just 3d arrays that match
   % exactly our grid
   %

   SIM.Env.ElementArray(x,y,z).Values(SIR) = ...
      SIM.Env.SE.Data(x,y,z);

elseif SIM.Env.SE.Flags.IsomorphicEnviroment

   SIM.Env.ElementArray(x,y,z).Values(SIR) = ...
      SIM.Env.SE.Data(snr_depth,...
      SIM.Env.SE.NumInputs+(depth-1)*...
      SIM.Env.SE.Format(1,1) + range);

else

   %
   % For non isomorphic enviroments we also have to find
   % the closest x,y location we have data for and also
   % find the bearing to that point first interpolate on
   % closest x,y data point
   %

   se_x = min(SIM.Env.SE.Format(5,1),...
      1+abs(round((ORG.Current.S.Values(POS,1)-...
      SIM.Env.SE.Format(5,3))/...
      SIM.Env.SE.Format(5,2)) ));

   se_y = min(SIM.Env.SE.Format(6,1),...
      1+abs(round((ORG.Current.S.Values(POS,2)-...
      SIM.Env.SE.Format(6,3))/...
      SIM.Env.SE.Format(6,2)) ));

   %
   % now find bearing from the ORG to this virtual target
   %
   se_theta = min(SIM.Env.SE.Format(4,1),...
```

```
      1+abs(round((SIM.Env.ElementArray(x,y,z).Values(THETA+1,1)-...
      SIM.Env.SE.Format(4,3))/...
      SIM.Env.SE.Format(4,2)) ));

   %
   % now calculate the row index were the combinations of x,y,
   % snr_depth, bearing exists. Data is shorted first by X,
   % then by Y, then by Snr Depth, then by bearing
   %

   sz = size(SIM.Env.SE.Data,1)/SIM.Env.SE.Format(5,1);
   rowIndex = 1+ sz * (se_x-1);

   % further refine find y location
   sz = sz/SIM.Env.SE.Format(6,1);
   rowIndex = rowIndex + sz*(se_y-1);

   % snr_depth is calculated

   rowIndex = rowIndex + (snr_depth-1);

   % now bearing

   rowIndex = rowIndex + (se_theta-1);

   SIM.Env.ElementArray(x,y,z).Values(SIR) = ...
      SIM.Env.SE.Data(rowIndex,SIM.Env.SE.NumInputs+...
      (depth-1)*SIM.Env.SE.Format(1,1) + range);

end;


%
% adjust the SIR value using direction gain
%

th = SIM.Env.ElementArray(x,y,z).Values(THETA+1,1);
if th > 90
   th = 180-th;
end;

if th < 0
   th
end;

index = floor(th/10)+1;
index2 = round(th/10)+1;
if index2 == th/10

   gainAdjust = SIM.ORG.Constraints.SensorDirectionalGain(index2);
```

```
else

   gainAdjust = ...
      (SIM.ORG.Constraints.SensorDirectionalGain(index2) + ...
      SIM.ORG.Constraints.SensorDirectionalGain(index))/2;

end;

SIM.Env.ElementArray(x,y,z).Values(SIR) = ...
   SIM.Env.ElementArray(x,y,z).Values(SIR)+...
   gainAdjust;



%
% calculate Pd value if we are currently pinging.
% we use the already tabulated 1-F(ts) values for the
% target strength/SE relationship
% This code represents an advanced sensor which uses
% information to calculate sensed values.
% The result is "derivative" sensory data
%

if Update

   Pt = SIM.Env.ElementArray(x,y,z).Values(PROB) ;

   ts = -SIM.Env.ElementArray(x,y,z).Values(SIR);

   if ts < SIM.Env.TsTable(1,1)

      Pd = 0.99;

   elseif ts > SIM.Env.TsTable(size(SIM.Env.TsTable,1),1)

      Pd = 0;

   else

      idx = find(abs(SIM.Env.TsTable(:,1) - ts) <=0.25);
      Pd = SIM.Env.TsTable(idx(1,1),2);

   end;

   if SIM.Env.ElementArray(x,y,z).Index(RxSS)

      %
      % return signal registered for this location.
      % Take that into account when calculating probability
```

```
        % BUBUG for now set Pd = 0 if RxSS active
        %

        Pd = 0.0;

    end;

    if SIM.Current.Iteration > 1

        %
        % use bayesian conditional probability..
        %

        Pfa = 0.00001;

        D = (1-Pd)*Pt + (1-Pfa)*(1-Pt);
        if D == 0
           dbgbreak;
        end;

        newProb =  ((1-Pd)*Pt) / D;

        SIM.Env.ElementArray(x,y,z).Values(PROB+1) =...
           SIM.Env.ElementArray(x,y,z).Values(PROB) - ...
           newProb;

        SIM.Env.ElementArray(x,y,z).Values(PROB) = newProb;

    end;

end;

SIM.Env.ElementArray(x,y,z).Index(POS) = 1;
SIM.Env.ElementArray(x,y,z).Index(POS+1) = 0;

Stimulus = SIM.Env.ElementArray(x,y,z);
return;
```

Core.m

```
function [RetrievedObjects, mV] = core(Function,In)

%
% define global variables
%

globdecl;
MBR_IDX = 1;
RPV_IDX = 2;
```

```
RT_ROW_IDX = 1;
LT_ROW_IDX = 2;

rowIdx = 0;

if bitand(In.P.Type, STIM_RT)

   %
   % short memory clustering
   %

   rowIdx = RT_ROW_IDX;

else

   rowIdx = LT_ROW_IDX;

end;


mV = 0;
if (Function == CORE_LEARN) | (Function == CORE_REACT)

   RetrievedObjects = [];

   %
   % find the closest match to this stimulus
   %

   if Function == CORE_LEARN
      [cIdx,mValues, RetrievedObjects] = bestMatch(In, 0);
   else
      [cIdx,mValues, RetrievedObjects] = bestMatch(In, 1);
   end;

   In.Iter = max(size(ORG.StateTrajectory,2),1);


      %
      % there is no memory, just add this
      %

   if ORG.Core.Constraints.Memory(rowIdx).Size == 0

      ORG.Core.Memory(rowIdx,1) = In;
      ORG.Core.Constraints.Memory(rowIdx).Size = 1;

   else

      if (Function == CORE_REACT)
```

```
    if ~isempty(mValues)
       mV = mValues(MBR_IDX);
    end;

    return;

end;

if ORG.Core.Constraints.Memory(rowIdx).Size == 0

    %
    % this can only mean our memory is empty.
    %

    ORG.Core.Memory(rowIdx,1) = In;
    ORG.Core.Constraints.Memory(rowIdx).Size = 1;
    return;

end;

if isempty(mValues)

    return;

end;

mV = mValues(MBR_IDX);

if (mV < ORG.Core.Constraints.Memory(rowIdx).MergeThreshold)

    if ORG.Core.Constraints.Memory(rowIdx).Size < ...
          ORG.Core.Constraints.Memory(rowIdx).MaxSize

       %
       % we have not reached memory capacity so we can add new
       % objects freely
       %

       ORG.Core.Constraints.Memory(rowIdx).Size = ...
          ORG.Core.Constraints.Memory(rowIdx).Size+1;

       ORG.Core.Memory(rowIdx,...
          ORG.Core.Constraints.Memory(rowIdx).Size) = In;
       ORG.Core.Memory(rowIdx,...
          ORG.Core.Constraints.Memory(rowIdx).Size).Iter=...
          max(1,size(ORG.StateTrajectory,2));

    else
       %
```

```
        % replace existing cluster
        %

        if isempty(In.Effect) == ...
              isempty(ORG.Core.Memory(rowIdx,cIdx(RPV_IDX)))

           ORG.Core.Memory(rowIdx,cIdx(RPV_IDX)) = In;
           ORG.Core.Memory(rowIdx,cIdx(RPV_IDX)).Iter = ...
              size(ORG.StateTrajectory,2);
        end;

     end;

     %
     % update our clustering threshold
     %

     ORG.Core.Constraints.Memory(rowIdx).MergeThreshold = ...
        ORG.Core.Constraints.Memory(rowIdx).MergeThreshold - ...
        ORG.Core.Constraints.Memory(rowIdx).MergeThreshold* ...
        exp(-1/ORG.Core.Constraints.Memory(rowIdx).Size)*0.005;

  else

     %
     % merge this object with the most similar one. Only compatible
     % objects can be merged. Stimuli sensed through longterm
     % module
     % are not compatible with stimuli sensed by reactive module
     % Internal stimuli are not compatible with external stimuli
     %

     matchIdx = cIdx(MBR_IDX);

     if ~isempty(ORG.Core.Memory(rowIdx,matchIdx).Effect) & ...
           ~isempty(In.Effect)

        in.P.Type = ORG.Core.Memory(rowIdx,matchIdx).P.Type;
        ORG.Core.Memory(rowIdx,matchIdx).Effect = ...
           mergeNodes(In.Effect, ...
           ORG.Core.Memory(rowIdx,matchIdx).Effect,mV);

        ORG.Core.Memory(rowIdx,matchIdx).P = mergeNodes(In.P,...
           ORG.Core.Memory(rowIdx,matchIdx).P,mV);

        ORG.Core.Memory(rowIdx,matchIdx).MergeCount = ...
           ORG.Core.Memory(rowIdx,matchIdx).MergeCount+1;
        ORG.Core.Memory(rowIdx,matchIdx).Iter = ...
           size(ORG.StateTrajectory,2);
```

```
            end;


            return;

         end;

      end;

   end;

if Function == CORE_RECALL

   %
   % use the input object as matching criteria to memorized objects
   % Then genenr

   RetrievedObjects = MEMORY_OBJECT;
   count = 1;
   countWithEffect = 0;

   TotalEffect = PROPERTY_OBJECT;
   TotalCnd = PROPERTY_OBJECT;

   %
   % for visualization purposes enable the POSition property in the
   % longterm cause
   %

   for i=1:ORG.Core.Constraints.Memory(rowIdx).Size

      if find((In.P.Index & ORG.Core.Memory(rowIdx,i).P.Index) == 1)

         if isempty(In.Effect) == ...
               isempty(ORG.Core.Memory(rowIdx,i).Effect)

            ORG.Core.Memory(rowIdx,i).P =...
               updateDynamicProperties(ORG.Core.Memory(rowIdx,i).P);

            RetrievedObjects(count) = ORG.Core.Memory(rowIdx,i);

            if isempty(In.Effect)

               %
               % if template had no effect, generate one here
               % by calling the CORE recursively on each recalled item
               %

               consider = 1;
               if isfield(In,'RangeHi') & ~isempty(In.RangeHi)
```

```
            %
            % the caller specifed a maximum detection criterion
            %

            if RetrievedObjects(count).P.Values(...
                  find(In.P.Index==1)) > ...
                  In.RangeHi(find(In.P.Index==1))

               consider = 0;

            end;

         end;

         if consider == 1

            [RetrievedObjects(count).Effect mv]= ...
               core(CORE_REACT,ORG.Core.Memory(rowIdx,i));

            if ~isempty(RetrievedObjects(count).Effect) & ...
                  (mv > 0)

               RetrievedObjects(count).Effect = ...
                  applytoorg(RetrievedObjects(count));

               TotalEffect.Index = TotalEffect.Index | ...
                  RetrievedObjects(count).Effect.Index;

               ix = find(...
                  RetrievedObjects(count).Effect.Index == 1);
               TotalEffect.Values(ix,:) = ...
                  TotalEffect.Values(ix,:) + ...
                  RetrievedObjects(count).Effect.Values(ix,:);

               countWithEffect = countWithEffect + 1;

               cix = find(...
                  RetrievedObjects(count).P.Index == 1);

               TotalCnd.Values(cix,:) = ...
                  TotalCnd.Values(cix,:) + ...
                  RetrievedObjects(count).P.Values(cix,:);

            end;

         end;


      end;
```

```
            count = count + 1;

         end;

      end;

   end;

   TotalCnd.Index(POS) = 1;

   if countWithEffect

      TotalEffect.Values(ix,:) = TotalEffect.Values(ix,:)./...
         countWithEffect;

      TotalCnd.Values(cix,:) = TotalCn d.Values(cix,:)/countWithEffect;

      RetrievedObjects(1).AverageCnd = TotalCnd;
      RetrievedObjects(1).AverageEffect = TotalEffect;

   else
      RetrievedObjects(1).AverageCnd = [];
      RetrievedObjects(1).AverageEffect = [];
   end;

   return;

end;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Out] = updateDynamicProperties(In)

globdecl;

Out = In;
Out.Values(ITER+1) = SET_LARGE_POS - ...
   (Out.Values(ITER)/SIM.Current.Iteration);

if In.Index(POS) == 1

   rpos = Out.Values(POS,1:3)-ORG.Current.S.Values(POS,1:3);
   Out.Values(DIST) = sqrt(sum(rpos.^2,2));

end;

Out.Values(ITER) = SIM.Current.Iteration;
```

```
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [M, replaceableV,output] = clusterMbr(Cluster, in)
% CLUSTERMBR Provides a memebrship value representing how well the new
% object belongs to this cluster
%

globdecl;

if in.P.Type ~= Cluster.P.Type
   M = 0;
   replaceableV = 0;
   output = [];
   return;
end;

mv = mbr(in.P,Cluster.P);
if isempty(mv)
   M = 0;
   replaceableV = 0;
   output = [];
   return;
end;

idx = find((in.P.Index & Cluster.P.Index) == 1);
iS= sum(mv(idx),2)/size(idx,2);

M = size(idx,2)/size(find(Cluster.P.Index==1),2)*iS;
output = [];

%
% depending on how many

if ~isempty(Cluster.Effect)


   output = Cluster.Effect;
   pIdx = find(output.Index == 1);
   output.Values(pIdx,:) =Cluster.Effect.Values(pIdx,:) * M;

end;

%
% now calculate a value that represents how replaceable this
% cluster is...
% the smallest the number the more desireable this cluster
% becomes
%
```

```
replaceableV = 1-M + Cluster.Iter/SIM.ORG.PropertyObject.RangeHi(ITER)
...
   + Cluster.MergeCount + ~isempty(Cluster.Effect);

if (Cluster.P.Type ~= in.P.Type)
   M = M*0.99;
end;

return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function corrProperties(in)
% CORRELATEPROPERTIES
% as we learn new rules we can keep track of what cnd
% properties are of importance and are associated with what
% effect/output properies. The matrix where the associations are kept
% is NxN matric where N= MAX_PROP_IDX at i(n,n) is the total weight of
% the property correlation...
%
globdecl;

if isempty(in.Effect)
   return;
end;

ox = find(in.Effect.Index == 1);
ix = find(in.P.Index == 1);

ORG.Core.PropCorrMatrix(ix,ox) = ...
   ORG.Core.PropCorrMatrix(ix,ox) + ones(size(ix,2),size(ox,2));

return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [NewNode] = mergeNodes(n1,n2,m)
% n2 is the existing node, we are going to merge n1 with

globdecl;

NewNode = n2;

pIdx = find(n1.Index == 1);
NewNode.Values(pIdx,:) = n1.Values(pIdx,:);
NewNode.Index = n1.Index | n2.Index;

pIdx = find((n1.Index & n2.Index) ==1);
```

```
for k=1:size(pIdx,2)

   nv1 = n1.Values(pIdx(k),1:SIM.ORG.PropertyObject.Size(pIdx(k)));
   nv2 = n2.Values(pIdx(k),1:SIM.ORG.PropertyObject.Size(pIdx(k)));

   w1 = m;
   w2 = 1-w1;

   nv2 = ((nv1*w1 + nv2*w2) + nv2)/2;
   NewNode.Values(pIdx(k),1:SIM.ORG.PropertyObject.Size(pIdx(k))) = nv2;

end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Memory Support functions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function clusterMerge(Mv, Idx)
% clusterMerge - It merges clusters that are too much alike...

globdecl;

if size(ORG.Core.Memory,2) < 5
   return;
end;

return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [SortedIndex, SortedValues, Effect] = ...
   bestMatch(in,GenerateEffect);

globdecl;

RT_ROW_IDX = 1;
LT_ROW_IDX = 2;

rowIdx = 0;
Effect = [];


if bitand(in.P.Type, STIM_RT)

   %
   % short memory clustering
   %

   rowIdx = RT_ROW_IDX;
```

```
    if isempty(ORG.Core.Memory)
       SortedValues = [];
       SortedIndex = [];
       Effect = in.Effect;
       return;
    end;

else

    rowIdx = LT_ROW_IDX;
    if isempty(ORG.Core.Memory) | (size(ORG.Core.Memory,1) == 1)
       SortedValues = [];
       SortedIndex = [];

       return;
    end;

end;

mx = 0;
matchIdx = [0 0];
nodesUsed = 0;

if ~GenerateEffect
   corrProperties(in);
else
   Effect = PROPERTY_OBJECT;
end;


for i=1:ORG.Core.Constraints.Memory(rowIdx).Size

   if GenerateEffect

      if ~isempty(ORG.Core.Memory(rowIdx,i).Effect)

         [matchIdx(i,1) matchIdx(i,2) tEffect]= ...
            clusterMbr(ORG.Core.Memory(rowIdx,i), in);

         if ~isempty(tEffect)

            Effect.Index = Effect.Index | ...
               ORG.Core.Memory(rowIdx,i).Effect.Index;

            ix = find(tEffect.Index == 1);
            Effect.Values(ix,:) = Effect.Values(ix,:) + ...
               tEffect.Values(ix,:);

            nodesUsed = 1 + nodesUsed;
```

```
            end;

        end;

    else

        nodesUsed = nodesUsed + 1;
        [matchIdx(i,1) matchIdx(i,2) tEffect] = ...
            clusterMbr(ORG.Core.Memory(rowIdx,i), in);

    end;

end;

if size(matchIdx,1) > 1

    [SortedValues SortedIndex] = sort(matchIdx);
    SortedValues = flipud(SortedValues);
    SortedIndex = flipud(SortedIndex);

elseif nodesUsed == 0

    SortedIndex = [];
    SortedValues = [];
    Effect = [];
    return;

else

    SortedIndex = [ 1 1];
    SortedValues = matchIdx;

end;

if GenerateEffect

    ix = find(Effect.Index == 1);
    Effect.Values(ix,:) = Effect.Values(ix,:)./nodesUsed;

end;


return;
```

Mbr.m

```
function [membershipValue] = mbr(Nn,Bn)
% Bn is the node we compare nN against
```

```
% a gaussiann membership is used, with the support being
% the values of of each property. Only proprties found in both
% Nn and Rn are considered ofcourse

globdecl;

pIdx = find((Nn.Index & Bn.Index) == 1);

if isempty(pIdx)
   membershipValue = [];
   return;
end;

mbv=PROPERTY_OBJECT.Index;
Size = SIM.ORG.PropertyObject.Size;
RangeHi = SIM.ORG.PropertyObject.RangeHi;
RangeLo = SIM.ORG.PropertyObject.RangeLo;
MbrFunction = SIM.ORG.PropertyObject.MbrFunction;
Sigma = SIM.ORG.PropertyObject.Sigma;

for k=1:size(pIdx,2)

   newV = Nn.Values(pIdx(k),1:Size(pIdx(k)));
   baseV = Bn.Values(pIdx(k),1:Size(pIdx(k)));
   mbv(pIdx(k)) = 1;
   for j=1:Size(pIdx(k))

      if (newV(j) > RangeHi(pIdx(k),j)) | ...
            (newV(j) < RangeLo(pIdx(k),j))

         %
         % out range for this membership..
         %

         mbv(pIdx(k)) = 0;

      else

         range = max(abs(RangeHi(pIdx(k),j) - baseV(j)), ...
            abs(RangeLo(pIdx(k),j)- baseV(j)) );

         %
         % for each active property generate a gaussian
         % membership. use product inference
         %

         switch MbrFunction(pIdx(k))
         case MEMBERSHIP_FUNCTION_GAUSSIAN,
            tmp = exp(-((newV(j)-baseV(j))/range)^2*Sigma(pIdx(k)));
         case MEMBERSHIP_FUNCTION_STEP,
```

```
            tmp = 1 - abs(((newV(j)-baseV(j))/range));
         case MEMBERSHIP_FUNCTION_TRIANGULAR,
            tmp = 1 - abs(((newV(j)-baseV(j))/range));
         end;

         mbv(pIdx(k)) = mbv(pIdx(k)) * tmp;

      end;

   end;

end;
membershipValue = mbv;
```

Longterm.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function longterm(RtStim,bAddLtObject)
% LONGTERM geenrates a long term stimulus for the org.
% it could be a dynamic goal (changing each iteration) or
% static one (pre determined).
% This particualr M file is for a specific long term goal(s),
% that are implementation specific

globdecl;

if ~SIM.ORG.Flags.StaticLtGoals
   return;
end;

Effect = RtStim;

LtEffect = [];
numSensed = 0;

if mod(SIM.Current.Iteration,SIM.ORG.LtCheckInterval) == 0

   TotalEffect= PROPERTY_OBJECT;
   numSensed = 0;
   ltStimuli = [];

   %
   % update upper bound of ITER field
   %

   SIM.ORG.PropertyObject.RangeHi(ITER) = SIM.Current.Iteration;

   %
   % the long term stimuli are clustered in the CORE.
```

```
    % To retrieve them we need to call the recall function.
    % Create a matching template to use for selecting which
    % goals to recall
    %

    template = MEMORY_OBJECT;
    template.P = PROPERTY_OBJECT;
    template.P.Type = bitor(STIMULUS_EXTERNAL,STIM_LT);
    template.P.Index(DIST) =1;
    template.RangeHi = SIM.ORG.PropertyObject.RangeHi;
    template.RangeHi(DIST) = ...
       SIM.ORG.Constraints.StimulusUpdateThreshold*0.75;
    ltStimuli = core(CORE_RECALL,template);

    numSensed = size(ltStimuli,2);

elseif ~isempty(ORG.Current.LtEffect)

    LtEffect = ORG.Current.LtEffect;
    cIdx = find(RtStim.Index == 1);
    Effect.Values(cIdx,:) = RtStim.Values(cIdx,:);

    cIdx = find(LtEffect.Index == 1);
    Effect.Values(cIdx,:) = LtEffect.Values(cIdx,:);

    cIdx = find((RtStim.Index & LtEffect.Index) == 1);
    Effect.Values(cIdx,:) = (LtEffect.Values(cIdx,:) + ...
       RtStim.Values(cIdx,:))/2;

    return;

end;


if numSensed > 0


    if isempty(ltStimuli(1).AverageEffect) & ...
          ~isempty(ORG.Current.LtEffect)

       %
       % no long term goal was worthy to be considered
       % in this iteration return no long term effect.
       % instead make ltEffect = rtEffect
       %

       LtEffect = ORG.Current.LtEffect;
       cIdx = find(RtStim.Index == 1);
       Effect.Values(cIdx,:) = RtStim.Values(cIdx,:);
```

```
    cIdx = find(LtEffect.Index == 1);
    Effect.Values(cIdx,:) = LtEffect.Values(cIdx,:);

    cIdx = find((RtStim.Index & LtEffect.Index) == 1);
    Effect.Values(cIdx,:) = (LtEffect.Values(cIdx,:) +...
       RtStim.Values(cIdx,:))/2;

    return;

end;

if isempty(ltStimuli(1).AverageEffect)
   return;
end;

LtEffect = ltStimuli(1).AverageEffect;
idx = find(LtEffect.Index == 1);

%
% now combine the lt effect with rt effect..
% only proprties active in both RT and LT cause
% and effect are combined first copy properties active
% only in rt or lt, then overwrite common ones with average
%

cIdx = find(RtStim.Index == 1);
Effect.Values(cIdx,:) = RtStim.Values(cIdx,:);

cIdx = find(LtEffect.Index == 1);
Effect.Values(cIdx,:) = LtEffect.Values(cIdx,:);

cIdx = find((RtStim.Index & LtEffect.Index) == 1);
Effect.Index = RtStim.Index | LtEffect.Index;

Effect.Values(cIdx,:) = (LtEffect.Values(cIdx,:) + ...
   RtStim.Values(cIdx,:))/2;


%
% Global Variable effect
%

ORG.Current.Effect = Effect;
ORG.Current.LtEffect = LtEffect;

%
% combined averaged cause
%

cIdx = find((ORG.Current.RtCause.Index & ...
```

```
        ltStimuli(1).AverageCnd.Index) == 1);

    ORG.Current.Cause.Values(cIdx,:) = ...
        (ORG.Current.RtCause.Values(cIdx,:) + ...
        ltStimuli(1).AverageCnd.Values(cIdx,:))/2;

    ORG.Current.LtCause = ...
        ltStimuli(1).AverageCnd.Values(cIdx,:);


end;
```

ApplyToOrg.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Effect] = applytoorg(Object)
% This function converts effects on motor properties, to ORG motor
% functions.
% It intepretes changes in properties based on the current application
% of the ORG algorithm. So this function is SIMULATION AND PROBLEM
% specific.
% Each abstract property translation must be dealt in the correct
% order..
%

globdecl;

StateTransition = Object.Effect;
Cnd = Object.P;

po = SIM.ORG.PropertyObject;

if StateTransition.Index(DIST+1) == 1

    %
    % Active Position change property
    %

    StateTransition.Index(POS+1) =1;
    spos = Cnd.Values(POS,1:3) - ORG.Current.S.Values(POS,1:3);
    [sTh,sR] = cart2pol(spos(1,1),spos(1,2));
    [x y] = pol2cart(sTh,-StateTransition.Values(DIST+1,1)*...
        SIM.ORG.PropertyObject.RangeHi(DIST+1));
    StateTransition.Values(POS+1,1:2) = [x y];

    StateTransition.Index(POS) = 1;
    StateTransition.Values(POS,1:3) = ...
        ORG.Current.S.Values(POS,1:3) + ...
```

```
      StateTransition.Values(POS+1,1:3);

   StateTransition.Values(POS+1,1:3) = ...
      StateTransition.Values(POS+1,1:3);

   %
   % NOTE this assumes we are constrained in 2d plane
   % instead of assuming it use the  Enviroment setting
   % and PropertySize(POS)
   %

   dx = StateTransition.Values(POS+1,1);
   dy = StateTransition.Values(POS+1,2);



   % get suggested asimuth
   [th,r] = cart2pol(dx,dy);
   StateTransition.Index(THETA) = 1;
   StateTransition.Values(THETA,1) = abs((th *180)/pi);


end;

if StateTransition.Index(NUM_OBJECTS+1) == 1

   StateTransition.Index(NUM_OBJECTS) = 1;
   StateTransition.Values(NUM_OBJECTS,1) =  max(1,...
      min(SIM.ORG.PropertyObject.RangeHi(NUM_OBJECTS),...
      ORG.Current.S.Values(NUM_OBJECTS,1) + ...
      StateTransition.Values(NUM_OBJECTS+1)));


end;


% THETA is disabled
if 0
   if StateTransition.Index(THETA+1) == 1

      %
      % we have a decision that will affect our direction(THETA)..
      %

      if StateTransition.Index(POS+1) == 1

         dx = StateTransition.Values(POS+1,1);
         dy = StateTransition.Values(POS+1,2);

         % get current suggested vector
```

```
            [th,r] = cart2pol(dx,dy);
            thDeg = (th *180)/pi;

            %
            % average calculated new velocity vector azimuth with corrected
            % one
            %

            newThDeg = (thDeg  + abs(ORG.Current.S.Values(THETA) - ...
               StateTransition.Values(THETA+1,1)))/2;

            %
            % convert back to cartesian...
            %

            newTh = (newThDeg * pi) / 180;
            [dx dy] = pol2cart(newTh,r);

            StateTransition.Values(POS+1,1:2) = [dx dy];

        end;

    end;
end;

Effect = StateTransition;

%
% update the decision threshold for binary effect properties
%

return;
```

Speculation.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function  speculation
% speculation - Speculate the effects of alternate rules/parameters
%  on the ORG behavior for just the future N steps.
%  This function essential spawns a second exact replica of the ORG
%  at time t, runs it until time t+N, takes a snapshot of the
%  enviroment, then allows the original ORG for N iterations,
%  and compares the enviroments using our intent properties as a
%  measure of how we did..
%  This can only be performed in simulatoion. But what the ORG learns,
%  is kept for the real world
%
```

```
globdecl;

if (SIM.ORG.Flags.Speculate == 1) & ...
      ORG.Current.S.Index(SPECULATE) %& ...
      %ORG.Current.S.Values(SPECULATE) > 0 )

   SIM.Speculation = [];
   internalPropertyIx = 0;
   newRule = [];

   %
   % save ORG state
   %

   currentOrg = ORG;

   %
   % take enviroment snap shot.
   %

   currentEnv = SIM.Env;
   currentSimState = SIM.Current;

   maxIter = SIM.MaxIterations;
   SIM.MaxIterations = SIM.Current.Iteration + ...
      SIM.ORG.SpeculationDuration;

   %
   % run the current ORG for N iterations
   % disable speculation so we are not called recursively
   %

   SIM.ORG.Flags.Speculate = 0;

   orgsim;

   %
   % take enviroment snap shot after normal ORG run.
   %

   SIM.Speculation.CurrentOrg = ORG;
   SIM.Speculation.CurrentEnv = SIM.Env;
   SIM.Speculation.CurrentSimState = SIM.Current;


   newOrg = ORG;
   newEnv = SIM.Env;
   newSimState = SIM.Current;

   %
```

```
% now re-wind to original ORG state when  entered this function..
%

ORG = currentOrg;
SIM.Env = currentEnv;
SIM.Current = currentSimState;

%
% create a new rule that applies to both internal and external
% stimuli
%

%
% for internal rules (rules applied to our own state) we
% can generate a rules that will be applied to the ORG when
% we self-reflect
% The rule uses the current ORG state as the condition
%

if SIM.ORG.Flags.SelfReflect

   idx = min(size(ORG.Core.ActiveParameters,2),...
      ceil(rand*size(ORG.Core.ActiveParameters,2)) );

   internalPropertyIx = ORG.Core.ActiveParameters(idx);
   newInternalValue = ...
      ceil((SIM.ORG.PropertyObject.RangeHi(internalPropertyIx) - ...
      SIM.ORG.PropertyObject.RangeLo(internalPropertyIx))*rand + ...
      SIM.ORG.PropertyObject.RangeLo(internalPropertyIx));

   %
   % check if we have generated an rule like this before..
   %

   m = MEMORY_OBJECT;
   m.P = PROPERTY_OBJECT;
   m.P.Index(internalPropertyIx) = 1;
   m.P.Values(internalPropertyIx,...
      1:SIM.ORG.PropertyObject.Size(internalPropertyIx)) = ...
      newInternalValue;

   [notUsed mv] = core(CORE_REACT,m);

   if mv > 0

      %
      % we have tried this before, discard..
      %

      internalPropertyIx = 0;
```

```
      ORG = newOrg;

   else

      ORG.Current.S.Values(internalPropertyIx) = ...
         newInternalValue;

   end;

end;

if SIM.ORG.Flags.SpeculateNewRules

   %
   % create a rule applied to external stimuli
   % only two input properties active
   %

   cnd = find(SIM.Env.ElementType.Index == 1);
   idx = min(size(cnd,2),ceil(rand*size(cnd,2)) );
   idx1 = min(size(cnd,2),ceil(rand*size(cnd,2)) );

   m = MEMORY_OBJECT;
   m.P = PROPERTY_OBJECT;
   m.P.Index([cnd(idx) cnd(idx1)]) = [1 1];

   idx = cnd(idx);
   idx1 = cnd(idx1);

   m.P.Values(idx,1) = ...
      rand*(SIM.ORG.PropertyObject.RangeHi(idx)-...
      SIM.ORG.PropertyObject.RangeLo(idx)) + ...
      SIM.ORG.PropertyObject.RangeLo(idx);

   m.P.Values(idx1,1) = rand*...
      (SIM.ORG.PropertyObject.RangeHi(idx1)-...
      SIM.ORG.PropertyObject.RangeLo(idx1)) + ...
      SIM.ORG.PropertyObject.RangeLo(idx1);

   m.P.Type = bitor(STIMULUS_EXTERNAL,STIM_RT);

   %
   % BUGBUG disable guessing which motor property to
   % change. For apl problem we only have DIST+1
   %

   if 0 %not enabled

      idx = min(size(ORG.Core.MotorProperties,2),...
         ceil(rand*size(ORG.Core.MotorProperties,2)) );
```

```
    idx1 = min(size(ORG.Core.MotorProperties,2),...
       ceil(rand*size(ORG.Core.MotorProperties,2)) );

    if idx1 == idx
       idx1 = size(ORG.Core.MotorProperties,2)-(idx-1)
    end;

    idx = ORG.Core.MotorProperties(idx);
    idx1 = ORG.Core.MotorProperties(idx1);

else

    idx1 = DIST+1;
    idx = DIST+1;


end;

m.Effect = PROPERTY_OBJECT;
m.Effect.Index(idx) = 1;
m.Effect.Index(idx1) = 1;

%
% normalize effect delta (range for derivative
% properties is always normalized within -1 and 1
%

m.Effect.Values(idx) = ...
    (SET_LARGE_POS-SET_LARGE_NEG)*(rand-0.5);
m.Effect.Values(idx1) = ...
    (SET_LARGE_POS-SET_LARGE_NEG)*(rand-0.5);

[notUsed mv] = core(CORE_LEARN,m);

if mv > 0

   %
   % we have seen this before..
   % see if it imrpoved perfomance..
   % BUGBUG check for active "intent"
   % properties such as PROB+1 for now discard
   %

   m = [];

else

   newRule = m;
   clear m;
```

```
    end;


end;

%
% check if speculation produced any valid rules
% if not, discard speculation until the next step
%

if (internalPropertyIx == 0) & isempty(newRule)

   ORG = newOrg;
   SIM.Env = newEnv;
   SIM.Current = newSimState;

   SIM.MaxIterations = maxIter;

   %
   % re-enable speculation
   %

   SIM.ORG.Flags.Speculate = 1;
   SIM.Speculation = [];

   dbgprint(1,...
      sprintf('Speculation: Aborting,no valid rules'));

   return;

end;

%
% now run simulation for N steps. We change the
% MaxIterations global variable so the orgsim.m loop
% stops when we want
%

orgsim;

%
% now take another enviroment snapshot
%

altOrg = ORG;
altEnv = SIM.Env;
altSimState = SIM.Current;

%
```

```
% compare the two... and keep the best ORG
% If required restore the current simulation state
% By comparing we can also learn new rules
%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%
%
% The ORG needs to look at the all the stimuli properties it can
affect
% and based on the instincts/learned rules determine the performance
% criteria
% BUGBUG hardcode here PROB+1(change in probabality, and number of
new
% or old objects that came into view...
% The properties we look to determine performance should be derived
% from the instincts: They are the ones in the Effect that show
% intent-> they cant be translated into a motor action


deltaAltProb = 0; i= 0;
for x=1:SIM.Env.Grid(1,1)
    for y=1:SIM.Env.Grid(2,1)
        for z=1:SIM.Env.Grid(3,1)

            if  altEnv.ElementIndex(x,y,z) > currentSimState.Iteration
                deltaAltProb = deltaAltProb + ...
                    (currentEnv.ElementArray(x,y,z).Values(PROB)- ...
                    altEnv.ElementArray(x,y,z).Values(PROB));

                i=i+1;
            end;

        end;
    end;
end;


deltaNewProb = 0; i= 0;
for x=1:SIM.Env.Grid(1,1)
    for y=1:SIM.Env.Grid(2,1)
        for z=1:SIM.Env.Grid(3,1)
            if  newEnv.ElementIndex(x,y,z) > currentSimState.Iteration
                deltaNewProb = deltaNewProb + ...
                    (currentEnv.ElementArray(x,y,z).Values(PROB)- ...
                    newEnv.ElementArray(x,y,z).Values(PROB));

                i=i+1;
            end;
```

```
      end;
   end;
end;


dbgprint(1,...
   sprintf('Speculation: CurrentOrg dProb %x, altOrg dProb %x',...
   deltaNewProb,deltaAltProb));

%
% note that the probability change is not averaged on purpose:
% we want the highest probability change overall, including that
% due to new objects
% we have never seen before
%

if deltaNewProb >= deltaAltProb

   %
   % discurd alternative
   %

   ORG = newOrg;
   SIM.Env = newEnv;
   SIM.Current = newSimState;

else

   %
   % count the times the speculated alternative was used
   % (for internal rule only) attempt to learn why the alternative
   % was better
   %

   SIM.SpeculationAltUsed = SIM.SpeculationAltUsed+1;

   m = MEMORY_OBJECT;
   m.P = PROPERTY_OBJECT;
   m.P.Type = bitor(STIM_RT, STIMULUS_INTERNAL);
   m.P.Index(internalPropertyIx) = 1;
   m.P.Values(internalPropertyIx,...
      1:SIM.ORG.PropertyObject.Size(internalPropertyIx)) = ...
      newInternalValue;

   m.Effect = PROPERTY_OBJECT;
   m.Effect.Index(PROB+1) = 1;
   m.Effect.Values(PROB+1) = SET_SMALL_NEG;
   [mv notUsed] = core(CORE_LEARN,m);

end;
```

```
    SIM.MaxIterations = maxIter;

    %
    % re-enable speculation
    %

    SIM.ORG.Flags.Speculate = 1;
    SIM.Speculation = [];

end;
```

Decision.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [FinalNextState] = decision
% DECISION

%
% define global variables
%

globdecl;

%
% preserve all properties active in our current state
% not affected by the state transition
%

idx = ORG.Current.Effect.Index;
idx = find((ORG.Current.S.Index & ~idx) == 1);
ORG.Current.Effect.Values(idx,:) = ORG.Current.S.Values(idx,:);

if SIM.ORG.Flags.SelfReflect

    %
    % use the intended change in our current state as an input into
    % our fuzzy system
    % The output might be a modified state transition that takes into
    % account learned experience
    %

    delta = ORG.Current.Effect;
    effect = ORG.Current.Effect;

    delta.Values = ORG.Current.Effect.Values - ORG.Current.S.Values;
    delta.Index = ORG.Current.Effect.Index;
```

```
    effect = ORG.Current.Effect;

    m = MEMORY_OBJECT;
    m.P = delta;
    m.P.Type = bitor(STIM_RT,STIMULUS_INTERNAL);
    [m.Effect, mV] = core(CORE_REACT,m);

    if mV > 0

       %
       % translate the changes if necessery
       %

       effect = ORG.Current.Effect;

       index = find((delta.Index & ORG.Current.S.Index) == 1)';
       effect.Values(index,:) = (delta.Values(index,:) +...
          m.Effect.Values(index,:))/2 + ...
          ORG.Current.S.Values(index,:);

    end;

    %
    % the second type of self reflection is to look at our current state
    % as a stimulus (condition) and see if it generates a reaction from
    % the core. For example, we might have an instinct that
    % directs us to do something based on our internal state
    % this type of stimulus should be considered long term
    % to distinguish from associations created from the enviroment filter
    %

    m = MEMORY_OBJECT;
    m.P = ORG.Current.S;
    m.P.Type = bitor(STIM_LT,STIMULUS_INTERNAL);
    [m.Effect, mV] = core(CORE_REACT,m);

    if mV > 0

       %
       % translate the changes if necessery
       %

       idx = find((m.Effect.Index & ORG.Current.S.Index) == 1)';
       effect.Values(idx,:) = m.Effect.Values(idx,:) + ...
          effect.Values(idx,:);

    end;


else
```

```
   effect = ORG.Current.Effect;

end;

%
% since this is a simulation, pass our intended state transition vector
% through a filter that simulates the enviromental effects and
% constraints the filter produces a modified/feasible state
% transition

[FilteredState] = envFilter(effect);

%
% calculate THETA
%

dx = FilteredState.Values(POS+1,1);
dy = FilteredState.Values(POS+1,2);

%
% get current suggested vector
%

[th,r] = cart2pol(dx,dy);
thDeg = (th *180)/pi;

FilteredState.Values(THETA,1) = thDeg;

%
% preserve active motor properties
%

FilteredState.Index = FilteredState.Index | ORG.Current.S.Index;

%
% try to learn if the final next state is different than the new
% state we wanted
% This is how we can discover enviromental effects..
% we are passing in the Effect before we self-reflected
%

if SIM.ORG.Flags.LearnFromEnvironmentalEffects
   learnenv(ORG.Current.Effect, FilteredState);
end;

FilteredState.Values(ITER) = SIM.Current.Iteration;

FinalNextState = FilteredState;
```

```
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [FilteredState] = envFilter(NextState)

globdecl;

%
% make sure sonar transition is valid
%

if NextState.Values(SNR_DEPTH,1) < ...
      SIM.ORG.PropertyObject.RangeLo(SNR_DEPTH,1)

   NextState.Values(SNR_DEPTH,1) = ...
      SIM.ORG.PropertyObject.RangeLo(SNR_DEPTH,1);

elseif NextState.Values(SNR_DEPTH,1) > ...
      SIM.ORG.PropertyObject.RangeHi(SNR_DEPTH,1)

   NextState.Values(SNR_DEPTH,1) = ...
      SIM.ORG.PropertyObject.RangeHi(SNR_DEPTH,1);

end;

if mod(NextState.Values(SNR_DEPTH,1)-...
      SIM.Env.SE.Format(3,3),SIM.Env.SE.Format(3,2)) ~= 0

   %
   % the sonar depth has to be set in one of the depths supported,
   % since we only have data for these
   %

   ix = min(SIM.Env.SE.Format(3,1),...
      abs(round((NextState.Values(SNR_DEPTH,1)-...
      SIM.Env.SE.Format(3,3))/SIM.Env.SE.Format(3,2))));
   ix = max(1,ix);

   NextState.Values(SNR_DEPTH,1) = SIM.Env.SE.Format(3,3)- ...
      ix*(SIM.Env.SE.Format(3,2));

end;

%
% enforce constant speed
%

[th,mag] = cart2pol(NextState.Values(POS+1,1),...
   NextState.Values(POS+1,2));
```

```
if (mag  <  SIM.ORG.PropertyObject.RangeLo(DIST+1,1)) | ...
      (mag  >  SIM.ORG.PropertyObject.RangeHi(DIST+1,1))

   mag = SIM.ORG.PropertyObject.RangeLo(DIST+1,1);

   % convert back to cartesian
   [x y] = pol2cart(th,mag);

   if sign(x) ~= sign(NextState.Values(POS+1,1))
      x = -x
   end;

   if sign(y) ~= sign(NextState.Values(POS+1,2))
      y = -y
   end;

   NextState.Values(POS+1,1:2) = [x y];

end;

%
% combine current velocity with new velocity to smooth
% out state transitions (due to inertia in this case).
% the weights are arbitrary
%

NextState.Values(POS+1,1:2)=NextState.Values(POS+1,1:2)*...
   0.4+ORG.Current.S.Values(POS+1,1:2)*0.6;

%
% enforce constant elevation constraint
%

NextState.Values(POS+1,3) = 0;

%
% re-calculate position change
%

NextState.Values(POS,1:3) = ORG.Current.S.Values(POS,1:3) + ...
   NextState.Values(POS+1,1:3);

FilteredState = NextState;
return;

%%%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function learnenv(iState,fState)
%LEARNENV observe the changes the enviroment incurred and try to
% correlate them with your stimuli current or your intended action
```

```
%
globdecl;

% first find the difference between the current state and the
% filtered state..

delta = fState.Values - ORG.Current.S.Values;

% then find the difference between intended state and current state
% (which will give us back the state transition vector after all
% translation)

deltaI = iState.Values - ORG.Current.S.Values;

diS = sum(deltaI,2);
dS = sum(delta,2);

idx = find ((diS-dS) ~= 0);

if ~isempty(idx) > 0

   %
   % changes occured, create a rule that puts the delta between
   % intended state and current state
   % as the condition, and the delta between filtered state and
   % current state as the effect
   %


   m = MEMORY_OBJECT;

   m.P = PROPERTY_OBJECT;
   m.P.Values(idx,:) = deltaI(idx,:);
   m.P.Type = bitor(STIM_RT, STIMULUS_INTERNAL);
   m.P.Index(idx) = 1;

   m.Effect = PROPERTY_OBJECT;
   m.Effect.Values(idx,:) = delta(idx,:);
   m.Effect.Index(idx) = 1;

   core(CORE_LEARN,m);

end;
```

Replay.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [TotalAvgProb] = replay
```

```
globdecl;

%
% this is used to play back a saved simulation
%

% BUGBUG limit iterations to 200
%SIM.Current.Iteration= 200;


%
% draw the SE maps.
%
clear se;

%
% display average probability per depth column, at different iterations
%

if (SIM.Visual.Replay.DrawAvgProb == 1) & ...
      (SIM.Current.Iteration > SIM.ProbSaveInterval)

   sz = floor(SIM.Env.Size(1,:) ./SIM.Env.Grid(:,2)');

   prob = zeros(sz(1,1),sz(1,2));
   figure(SIM.Visual.ProbabilityFigNum);
   clf;

   %
   % take a snapshot of the probability space on the current iteration
   %

   for x=1:SIM.Env.Grid(1,1)
      for y=1:SIM.Env.Grid(2,1)
         for z=1:SIM.Env.Grid(3,1)
            if  SIM.Env.ElementIndex(x,y,z) > 0
               SIM.StateTrajectory(...
                  SIM.Current.Iteration).ProbMap(x,y,z)...
                  = SIM.Env.ElementArray(x,y,z).Values(PROB);
            end;
         end;
      end;
   end;

   for k=SIM.ProbSaveInterval:...
         SIM.ProbSaveInterval:SIM.Current.Iteration

      if k+SIM.ProbSaveInterval > SIM.Current.Iteration
         k = SIM.Current.Iteration;
      end;
```

```
      for i=1:SIM.Env.Grid(3,1)
         [x y] = size(SIM.StateTrajectory(k).ProbMap(:,:,i));
         prob(1:x,1:y) = prob(1:x,1:y) + ...
            SIM.StateTrajectory(k).ProbMap(:,:,i);
      end;

      prob = prob ./SIM.Env.Grid(3,1);

      plotProbability(prob,k);

      axis auto;
      pause;

      TotalAvgProb(k) = ...
         sum(sum(sum(prob(:,:,:))))/...
         (SIM.Env.Grid(1,1)*SIM.Env.Grid(2,1)*SIM.Env.Grid(3,1));

      prob = zeros(sz(1,1),sz(1,2));

   end;

end;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%
% Visualize SE maps, if isomorphic enviroment was used
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


if (SIM.Visual.Replay.DrawSeMaps == 1) & ...
      SIM.Env.SE.Flags.IsomorphicEnviroment

   figure( SIM.Visual.SeMapFigNum);
   clf;

   for s=1:SIM.Env.SE.Format(3,1)
      for r=1:SIM.Env.SE.Format(1,1)
         for d=1:SIM.Env.SE.Format(2,1)

            se(d,r,s)=SIM.Env.SE.Data(s,...
               SIM.Env.SE.NumInputs+(d-1)*SIM.Env.SE.Format(1,1) + r);

         end;
      end;


   end;

   subplot(2,4,1),surf(se(:,:,1));
```

```
    view(0,270);

    subplot(2,4,2),surf(se(:,:,2));
    view(0,270);

    subplot(2,4,3),surf(se(:,:,3));
    view(0,270);

    subplot(2,4,4),surf(se(:,:,4));
    view(0,270);

    subplot(2,4,5),surf(se(:,:,5));
    view(0,270);

    subplot(2,4,6),surf(se(:,:,6));
    view(0,270);

    subplot(2,4,7),surf(se(:,:,7));
    view(0,270);

    pause;

end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Visualize trajectory through enviroment map
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(SIM.Visual.EnvFigNum);
clf;

xyz=[];
set(gca,'XGrid','off');
set(gca,'YGrid','off');

hold on;
prob = 0;

maxIteration = SIM.Current.Iteration;

for x=1:SIM.Env.Grid(1,1)
   for y=1:SIM.Env.Grid(2,1)
      for z=1:SIM.Env.Grid(3,1)

         if (SIM.Env.ElementIndex(x,y,z) > 0)
            plotobj(SIM.Env.ElementArray(x,y,z),...
               PLOT_VIRTUAL_TARGET,SIR);
         end;

      end;
```

```
      end;
   end;


   SIM.Visual.ColorMap(ITER,:,:) = hsv(512);
   xyz = ORG.StateTrajectory(1).S.Values(POS,1:3);
   h =text(xyz(1,1),xyz(1,2),xyz(1,3),'START');
   set(h,'Color',[ 1 1 1]);
   for k=1:maxIteration-1

      po = ORG.StateTrajectory(k).S;
      po.Values(ITER,1) = k;
      plotobj(po,PLOT_ORG,ITER);

      %
      % plot cause centers
      %

      po = ORG.StateTrajectory(k).Cause;
      if ~isempty(po)
         po.Type = bitor(po.Type,STIM_LT);
         %plotobj(po,PLOT_ORG,SIR);
      end;

   end;

   hold off;

   set(gcf,'Color',[ 0.1 0.1 0.1]);
   set(gca,'Color',[ 0 0 0]);
   set(gca,'XColor',[ 1 1 1]);
   set(gca,'YColor',[ 1 1 1]);
   set(gca,'ZColor',[ 1 1 1]);


   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   function plotProbability(prob,k)
   globdecl;

   %
   % plot surface
   %

   surf(prob');

   caxis([0 1]);
   h=title (sprintf('Average probability per depth column, i=%d',k));
   set(h,'Color',[ 1 1 1]);
```

```
set(gca,'XGrid','off');
set(gca,'YGrid','off');
set(gcf,'Color',[ 0.1 0.1 0.1]);
set(gca,'Color',[ 0 0 0]);
set(gca,'XColor',[ 1 1 1]);
set(gca,'YColor',[ 1 1 1]);
set(gca,'ZColor',[ 1 1 1]);

axis ([0 12 0 12 0 1]);
axis normal;
view(-27,42);

if SIM.Visual.Replay.OverlayPath == 1

   %
   % plot ORG path
   %

   hold on;

   SIM.Visual.ColorMap(ITER,:,:) = hsv(512);
   xyz = ORG.StateTrajectory(1).S.Values(POS,1:3);
   h =text(xyz(1,1),xyz(1,2),xyz(1,3),'START');
   set(h,'Color',[ 1 1 1]);
   for j=1:k-1

      po1 = ORG.StateTrajectory(j).S;
      po1.Values(POS,1:2) = po1.Values(POS,1:2)./1000;
      po1.Values(POS,3) = 0.8;
      po1.Values(ITER,1) = j;
      plotobj(po1,PLOT_ORG,ITER);

      %
      % plot cause centers
      %

      po = ORG.StateTrajectory(j).Cause;
      if ~isempty(po)

         po.Type = bitor(po.Type,STIM_LT);
         po.Values(POS,1:2) = po.Values(POS,1:2)./SIM.Env.Grid(1,2);
         po.Values(POS,3) = -po.Values(POS,3)./SIM.Env.Size(1,3);
         plotobj(po,PLOT_ORG,SIR);

         if mod(j,5) == 2

            %
            % every N iterations draw a line between a cause center
            % and the ORG position
            %
```

```
            x(1) = pol.Values(POS,1);
            x(2) = po.Values(POS,1);

            y(1) = pol.Values(POS,2);
            y(2) = po.Values(POS,2);

            z(1) = pol.Values(POS,3);
            z(2) = po.Values(POS,3);

            h= line(x,y,z);
            set(h,'Color',[1 1 0]);

         end;

      end;

   end;

   hold off;

end;
```

Plotobj.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function plotobj(SObject, Type,idx);
% It plots in the virtual enviroment, a single enviroment Object or
% the representaion of an ORG
%

globdecl;

%
% print a message with the object behaviour only if its enabled
%

p = SIM.ORG.PropertyObject;

xyz = SObject.Values(POS,1:3);

s = SObject.Values(idx,1:p.Size(idx));
range = p.RangeHi(idx,1:p.Size(idx)) - p.RangeLo(idx,1:p.Size(idx));

s = max(1,...
   floor(max(((abs(s-p.RangeLo(idx,1:p.Size(idx)))) / range)*...
   size(SIM.Visual.ColorMap(idx,:,:),2),1)));

if s > 512
```

```
   s= 512;
end;

obj_color = SIM.Visual.ColorMap(idx,s,1:3);

if bitand(SObject.Type, STIM_RT)
   h=plot3(xyz(1,1),xyz(1,2),xyz(1,3),'.');
   set(h,'Color',obj_color);
else
   h=plot3(xyz(1,1),xyz(1,2),xyz(1,3),'*');
   set(h,'Color',obj_color);
end;
return;
```

Vita

George Chrysanthakopoulos received his B.S degree in Computer Engineering from Virginia Tech in December 1995. Upon graduation he joined Microsoft Corporation as a software design engineer in the Windows NT Kernel Development group (NT Base) and enrolled in the Electrical Engineering graduate program at the University of Washington. He received his M.S degree in December 1997 and PhD degree in September 2000, both in Electrical Engineering. He is currently a development lead for device drivers and digital streaming in the X-Box Operating System group at Microsoft.