Learning Tactical Human Behavior Through Observation of Human Performance

Hans K. G. Fernlund, Avelino J. Gonzalez, *Senior Member, IEEE*, Michael Georgiopoulos, *Senior Member, IEEE*, and Ronald F. DeMara, *Senior Member, IEEE*

***Abstract*—It is widely accepted that the difficulty and expense involved in acquiring the knowledge behind tactical behaviors has been one limiting factor in the development of simulated agents representing adversaries and teammates in military and game simulations. Several researchers have addressed this problem with varying degrees of success. The problem mostly lies in the fact that tactical knowledge is difficult to elicit and represent through interactive sessions between the model developer and the subject matter expert. This paper describes a novel approach that employs genetic programming in conjunction with context-based reasoning to evolve tactical agents based upon automatic observation of a human performing a mission on a simulator. In this paper, we describe the process used to carry out the learning. A prototype was built to demonstrate feasibility and it is described herein. The prototype was rigorously and extensively tested. The evolved agents exhibited good fidelity to the observed human performance, as well as the capacity to generalize from it.**

***Index Terms*—Context-based reasoning, human behavioral modeling, genetic programming, simulation.**

I. Background and Problem Definition

WE BELIEVE that the most significant obstacle in the development of *tactical agents* has been the inability of developers to quickly and effectively build robust agents upon demand. We define tactical agents as *computer models of humans that deal with their environment in a way similar to how a human would if facing the same situation while in the execution of a mission*. These agents demonstrate *tactical behavior*, which we further define as *the continuous and dynamic process of decision making by a tactical agent (human or otherwise) who interacts with its environment while attempting to carry out a mission in the environment.*

The high level of complexity of many of the existing agent architectures makes the task of building these tactical agents very difficult. We assert that this difficulty and its consequent expense have curtailed the widespread use of *computer-generated forces* (CGFs) in military training simulations. It would be beneficial if models of tactical human behavior could be easily built and fielded within a few days of their being requested. This quick turn-around ability could for example, permit virtual opponent entities in a simulation to be developed for use in just-in-time mission or game rehearsal.

Furthermore, there is evidence in the literature suggesting that models developed through an interaction between the model engineer (ME) and the subject matter expert (SME) tend to represent generic, doctrine-like behavior [6], [19], [27], [33]. That is, the behavior is quite logical and possibly optimal, but often with little resemblance to a real human's actions [8]. Pew and Mavor [28, p. 15] support this hypothesis by stating that such doctrinally-based agents are "... expressed as verbal descriptions of institutional, social, and political processes" and such verbal models are not capable of reflecting the complex behavior required in tactical situations. These agents fail to represent the personal characteristics of different people that can cause their performance to vary, possibly significantly. It would then be beneficial if models could be sufficiently fine-grained to reflect individual human traits, such as aggressiveness, passiveness, incompetence and fatigue as well as the optimal behavior.

We assert that the cause of these shortcomings lies partly in the way the agents are represented and partly in the means employed to elicit the knowledge used in these models. Our investigation addresses the second problem by devising a method through which agents can be built relatively quickly through the (automated) observation of a human expert executing the desired mission or task in a simulator. Our hypothesis is that one does not need to understand the cognitive process behind the observable actions of a human in order to quickly build a model of a human's behavior. Furthermore, this process of learning from observation permits the agent to reflect the personal characteristics of the human being observed with a fidelity at least as good as that of agents created in a traditional manner.

II. Learning From Observation

Automatic model construction could conceivably improve the development process of human-like tactical agents. If the agents could be created automatically merely by observing a human executing a mission on a simulator or on a well instrumented live exercise, it would dramatically facilitate the agent development process. We refer to this as *learning from observation*.

The term *learning from observation* has its roots in biology. Infants of many species often learn things by observing adults. Studies have shown that humans fully develop observational learning by the age of 24 months [1]. By that age, children can easily learn a simple task by observing another person performing it. Inspired by how humans and other mammals seem

Manuscript received May 6, 2004; revised December 10, 2004 and March 20, 2005. This work was supported in part by the Swedish Knowledge Foundation under the project Promote IT. This paper was recommended by Associate Editor F. Gomida.

H. K. G. Fernlund is with the Department of Culture, Media and Computer Sciences, Dalarna University, Dalarna, 78188 Borlänge, Sweden (e-mail: hfe@du.se).

A. J. Gonzalez, M. Georgiopoulos, and R. F. DeMara are with the Intelligent Systems Laboratory, Electrical and Computer Engineering Department, University of Central Florida, Orlando, FL 32816 USA (e-mail: gonzalez@mail.ucf.edu; michaelg@mail.ucf.edu; demara@mail.ucf.edu).

Digital Object Identifier 10.1109/TSMCB.2005.855568

to learn by observation, the machine learning community has developed several theories on learning from observation as applied to different areas. The AI literature often refers to learning from observation as a method of learning the behavior of another agent or human by observing its action [3], [23], [32], [34]. Such a characterization of learning from observation merely dictates how the data for learning are to be collected—through observation. It makes no statements on what learning paradigms to use. We, thus, define learning from observation as:

> The agent shall adopt the behavior of the observed entity solely from interpretation of data collected by means of observation.

Our specific objective is to learn the tactical behavior of an observed human and create a tactical agent that is able to reproduce this behavior in a simulation or in the real world. We present a new approach to building tactical agents from observation, where the resulting agents inherit personal (i.e., individual) behavior patterns for realistic, but nonoptimal tactical behavior. The novel contributions of this investigation are both, the objective of the research (i.e., personalized models of tactical behavior) and the means to develop them [i.e., the synergistic combination of context-based reasoning (CxBR) and genetic programming (GP)].

A. Literature Review

Gonzalez *et al.* [16] argue that learning from observation is especially well-suited to acquiring *tactical knowledge*—that knowledge used to select the most appropriate action for a given situation. Some of this tactical knowledge could consist of behavior patterns about which the human is unaware, undesirable behavior patterns, or behavior based on intuition. Prior work in modeling human behavior through learning from observation has been conducted in the area of humanoid robots [7], maneuvering a car [29], flying an aircraft [23] and driving a tank [20]. However, most of the work has been done to create an optimally performing agent and often regarding low-level motor skills. The objective in the area of humanoid robots is to mimic the human motion pattern and have the robots perform in a human-like fashion. The similarity to human behavior here, however, lies in the motor skills and not in tactical decision making. When it comes to controlling a vehicle, the task could be to control the vehicle in the most appropriate manner (i.e., the best humanly possible) [32]. In other words, the objective was, for example, to create a model that represents an expert driver who follows all the traffic rules. If, on the other hand, the objective is to create a realistic simulation, then models of both good and bad drivers should be incorporated and the models of individuals should behave differently from each other.

Few results have been presented in the literature where the focus is on personalized behavior patterns and/or tactical decision making. Sammut *et al.* [31] presented results where personalized agents learned individual behavior of 20 different pilots flying a specific route. The agents cloned the pilots' behaviors and only learned to fly the given route. However, the agents could not fly any other route or be used in other environments. Modeling tactical behavior by observation was also investigated by Sidani [32]. Here, two learned behavior patterns were merged and the agent had to make a tactical decision on which of the two behavioral patterns to use. However, the tactical decision made by the agent was not learned from observation, nor evaluated with the actor's performance.

A feasible way of collecting data and recording the actions of the user is to employ a simulator to implement learning from observation, as in the work of Gonzalez *et al.* [15]. By using a simulator instead of the real world, data collection can be simplified as there is no need to worry about instrumentation and perception issues. Additionally, it would be possible to compose and observe difficult or dangerous situations that would not be feasible if done live. Furthermore, it is much easier to design scenarios supporting the objectives of the research and to put the actor in situations worth monitoring. Lastly, there is no need for complex sensors or image recognition algorithms to be able to understand the simulated environment.

B. Our Approach to Learning From Observation

We devised a learning strategy that is compatible with the objective of building agents that display tactical human behavior. To achieve this, we combine GP as the learning strategy and CxBR as the underlying modeling paradigm in which the resulting models will be expressed and executed. This combination turns out to be quite synergistic because of the following.

1) GP is a nontransforming learning paradigm. That is, it does not transform the learned knowledge into an obscure representation, as does a neural network (e.g., its weights). The output of a GP process is source code. Therefore, the resulting model can be directly inserted into the CxBR model structure with no modification. The nontransforming nature of GP makes it possible to interpret and analyze the learned knowledge.
2) CxBR provides a hierarchical structure that modularizes the knowledge for the GP learning algorithm to operate in. It serves to prune the search space that could enhance the learning capabilities of GP when the GP is learning [21].

Before continuing, very brief discussions of CxBR and GP follow as a background to the design decisions that had to be made for the new approach to learning from observation.

C. Context-Based Reasoning

The concept of using contexts for modeling human intelligence has received significant interest from researchers in the AI literature. McCarthy [25] developed an algebra for defining and manipulating contexts. Guha [18] employed a similar algebra to incorporate the concept of contexts in the development of the CYC system. Brezillon [5] developed context-based intelligent assistant systems while Turner [35], [36], Bass *et al.* [2] and Gonzalez and Ahlers [13] have all independently developed context-based approaches to modeling human behavior.

Gonzalez and Ahlers presented CxBR as a modeling technique that can efficiently represent the tactical behavior of humans in intelligent simulated agents. Results have shown that it is especially well-suited to modeling such behavior. CxBR is based on the idea that:

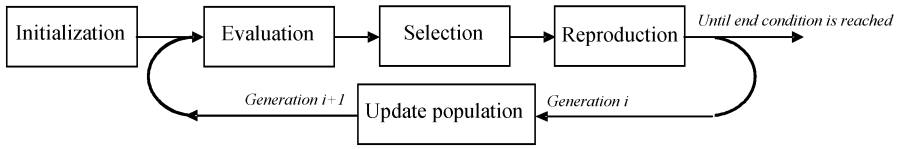


Fig. 1. Evolving GP.

- a situation calls for a set of actions and procedures that properly address the current situation;
- as a mission evolves, a transition to another set of actions and procedures may be required to address a new situation;
- things likely to happen under the current situation are limited by the current situation itself.

CxBR encapsulates knowledge about appropriate actions and/or procedures for specific situations, as well as compatible new situations, into hierarchically-organized *contexts*. All the behavioral knowledge is stored in the *Context Base* (i.e., the collection of all contexts). The top layer of contexts in the hierarchy contains the *Mission Context*. At the next layer are *Major Contexts* and below them, a number of *Subcontext* layers can exist. Mission Contexts define the mission to be undertaken by the agent. While it does not control the agent per se, the Mission Context defines the scope of the mission, its goals, the plan, and the constraints imposed (time constraints, weather, etc.). The Major Context is the primary control element for the agent. It contains *action knowledge* (the knowledge that controls the agent's action within the context) and *transitional knowledge* (the knowledge that determines the activation and de-activation of the context), plus a list of compatible Major Contexts that can follow the current one. Identification of a new situation can now be simplified because only a limited number of all situations are possible under the currently active context. Subcontexts are abstractions of functions performed by Major Contexts which may be too complex for one function, or that may be employed by other Major Contexts. This encourages re-usability. Subcontexts will de-activate themselves upon completion of their actions.

Transitions between contexts are typically triggered by events in the environment—some planned others unplanned. However, events internal to the agent (i.e., breakdown) can also trigger transitions. Expert performers are able to recognize and identify the transition points quickly and effectively.

Our work assumes only one mission and one major context active at any one time. For the purposes of simulated training exercises, this is an acceptable limitation. We leave the extension to simultaneous missions/contexts for future research.

CxBR is a very intuitive, efficient and effective representation technique for human behavior. It has proven successful in several applications to tactical behavior modeling. While it has no built-in means of learning such as case-based reasoning or neural networks, it does lend itself to facilitating learning and knowledge acquisition [17]. Our work described here takes advantage of this fact. A full description of CxBR can be found in Gonzalez and Ahlers [14].

D. Genetic Programming

GP [22] is derived from genetic algorithms. Like genetic algorithms, GP is a stochastic search algorithm inspired by Darwin's theories of evolution. The GP search process looks for the best suitable program that will solve a problem. The target system for the GP could be a CPU, a compiler, a simulation, or anything else that could execute the predefined instructions. We will refer to the target system as simply a *program*. In this investigation the GP evolves source code statements. This makes it very compatible for use within CxBR. GP can build complete software programs that support the internal construction of the context-base.

To make GP work, some basic requirements must be satisfied. First, we need to have a population of *individuals*—programs that represent different solutions to the problem. Note that the representation of the source code needs to ensure that even if the code is randomly generated, or randomly modified, the source code should be syntactically correct end executable. The steps of the GP algorithm are described in Fig. 1 (enhanced from Eklund [10]). The *Initialization* of the individuals takes place prior to learning. In the learning phase, all the individuals need to be evaluated in some manner as to what degree they are able to solve the problem. This is done by a *fitness function* comparing each individual against the observed human performance (*Evaluation*). The individuals with better fitness would preferably be preserved and survive, and/or breed new individuals to the next generation (*Selection*). The next GP step is to evolve the individuals (*Reproduction*) in some manner to preserve the "good" features and develop even better individuals. Reproduction is conducted by applying genetic operators, such as *crossover* and *mutation*, to the selected individuals. When a new population has been evolved the old one can be discarded and the learning process continues on with the new population (*Update population*).

Individuals with a better fitness value are more likely to be selected to participate in breeding the next generation of individuals than those with a worse fitness value. This natural selection process continues over many generations, encouraging better performing individuals and pruning worse performing ones. When the evolutionary process is finished in GP, there then exists a program that will solve the problem using the best-fit individual. More information on GP can be found in [22].

Because GP builds source code, it could be used to incorporate knowledge in any context level or in any instances within CxBR where human behavior is encoded. This means that we could choose to implement learning in any specific part of CxBR and construct the knowledge therein.

E. Proposed Learning Strategy That Implements Learning From Observation

Before continuing, it is important to remember that there are two types of knowledge in CxBR: 1) Correct *action-knowledge* to handle a specific situation and 2) *transitional knowledge* to determine which context to apply to the current situation (i.e.,

situational awareness). Learning the action knowledge for a specific task becomes a nonlinear regression analysis where the discrepancies between the model and the observed human are being minimized and a nonlinear curve fitting takes place. The situational awareness, on the other hand, is more of a classification problem where the learning algorithm must classify the current situation in order to activate the correct context. In order to build the context base, the learning algorithm must meet two requirements: 1) it must be able to cope with regression problems and 2) with classification problems. GP has successfully been applied to those two problem domains [4], [9]. Hence, we assert that the GP learning approach can indeed support the process of building a context base.

Our work makes a small compromise to our definition of learning from observation by requiring that the *context organization* be defined *a priori*. The context organization here refers to a definition of the contexts and subcontexts that will be used by the agent in executing its required task. The context organization represents a design of the system components (the contexts and subcontexts) that will be used by the agent to perform the specific task envisioned. The context organization is highly mission-specific, and it is designed by the knowledge engineer charged with developing the agent, possibly with assistance from a subject matter expert. Note however, that the contexts and subcontexts in this predefined organization are all empty—they initially contain no code. This *a priori* definition of structure eliminates the need for the learning strategy to suggest new contexts. While the latter would be very useful, we leave it for future research. It also gives the learning algorithm a structure within which to work that prunes the search space for the learning algorithm.

Wolpert and Macready [37], [38] conclude in the No Free Lunch theorem that all machine learning paradigms need to be tuned for the problem at hand to enhance their performance. In some way, the learning algorithm needs to incorporate problem-specific knowledge into the behavior of the algorithm. By using the context hierarchy of CxBR, it makes the CxBR and GP approach a synergistic approach to learn human behavior from observation. Similar work [21] has presented results that indicate that splitting the knowledge into a hierarchy actually improves the performance of the GP learning algorithm. GP evolves the knowledge in an interpretable and well-known manner (i.e., source code statements) applicable in all parts of a context base (i.e., correct action for the current situation and situational awareness knowledge). On the other hand, predefining the CxBR context organization enhances the probability of a successful GP evolution because it facilitates the learning task.

In the next section, we describe how these two techniques were combined into a practical artifact to build models of human performance.

III. An Artifact That Learns by Observation Through CxBR+GP

With the above concepts in mind, we developed an algorithm that builds an agent that emulates human individual performance by observing a human perform an action or task on a simulator. We call this algorithm *Genetic Context Learning* (GenCL). To evaluate the effectiveness of GenCL, we developed and tested an artifact that incorporates this approach. This section describes both the development and the testing of GenCL. We identified two main difficulties with the implementation of conventional GP learning (i.e., any learning algorithm) in CxBR:

1) hierarchical complexity;
2) interdependency among contexts.

The first problem points to the need to implement the learning at the different levels of contexts (e.g., major contexts and subcontexts). Simultaneously evolving human behavior at all CxBR hierarchical levels is very complex. The number of combinations to try in the evolution process (the search space) would indeed be very large. Different approaches have been devised by others to address this problem. One approach is to reduce this search space by letting the GP evolve more *complex behavior* by using a meta-language, where the function set is a set of *detailed behavior* routines [24]. In the traditional CxBR approach, this complex behavior (also referred to *as high-level behavior*) is represented at the major context level, while the detailed behavior (also referred to as *low-level behavior*) would be modeled at the subcontext level. This technique would require manual construction of the behavior in the lower context levels that is not evolved by GP. This restricts the problem solving in many ways but also contradicts the idea of learning from observation in that there would be significant manual development required. Hsu and Gustafson [21] propose an alternative GP learning approach where the high-level behavior (complex behavior) would be broken down into lower level behaviors (detailed behavior) that are learned first. The completed lower-level behaviors (lower-level contexts) are then used as available functions in the GP's function set when the higher-level behavior is being learned later. The approach is called layered learning GP (LLGP) and it decomposes the complex problem into a hierarchy of subproblems. Hsu and Gustafson showed promising results in boosting the learning performance by using the LLGP strategy. This is a type of bottom-up strategy that suits the needs of our problem domain well because the problem domain representation in CxBR already has a hierarchical organization.

Different contexts constituting tactical human behavior are unavoidably correlated with one another. Such is the case, for example, when driving in city traffic and approaching an intersection with a red traffic light where cars are already waiting at the light. If one subcontext controls a car's actions at the traffic light while another controls its behavior in the presence of other cars in the same lane, those two subcontexts are interdependent in that situation—the learning of these two instances could not be performed mutually exclusively. Note that the performances of the two contexts are *interdependent*, i.e., successful performance depends on an interaction and collaboration between the activation and de-activation of the two contexts. However, only one of the contexts can be active at any particular moment. For a new context to be activated, the active context at the same level needs to release its control of the agent (de-activate it). The interdependent performance of contexts at the same level needs to be evaluated in some way during the learning process. This means that the embedded mechanism

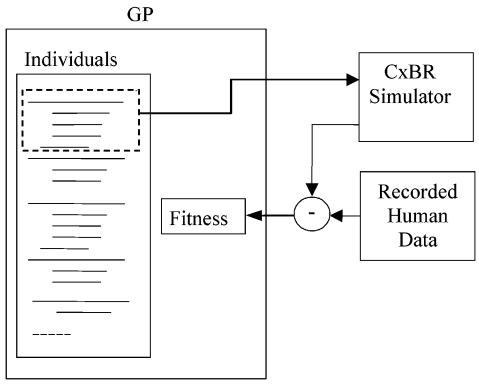


Fig. 2. GenCL.

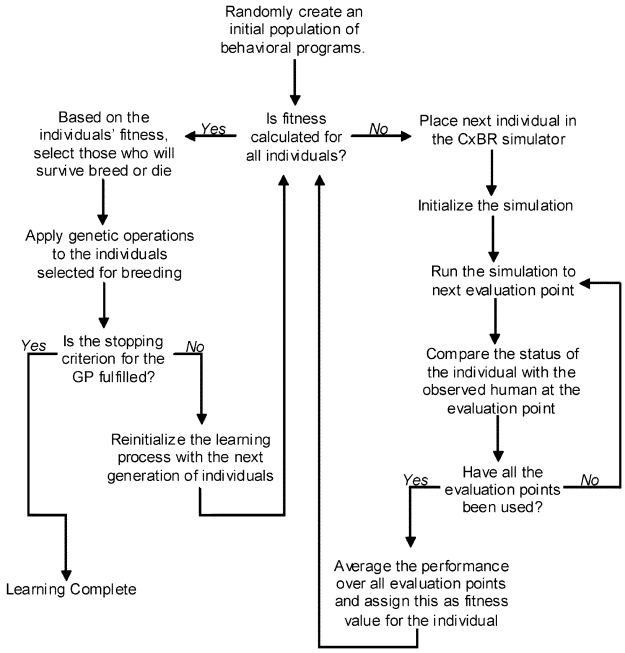


Fig. 3. GenCL learning algorithm.

that controls the activation of contexts cannot evolve separately. Instead, they need to evolve together, simultaneously. An approach to this problem is to use Potter and DeJong's Cooperative Co-Evolutionary strategy [30]. In this strategy, it is possible to have different populations evolving solutions to interdependent problems in parallel. When it comes to evaluation of the performance (i.e., calculation of the individual's fitness) of their joint effort, the best individuals from the other populations are included to produce a fitness value for the individuals in the population in focus. In other words, the fitness function for individual k in population 1 is not only a function of this individual, but also includes the best individual from population 2: $f_{1,k}(i_{1,k}, i_{2,\text{best}})$. According to this, we allow the embedded mechanism that controls the action to evolve in parallel and evaluate their joint performance when the fitness value is calculated. We use LLGP and co-evolution as basic GP strategies for learning human tactical behavior from observation in CxBR.

A. Learning in GenCL

The learning process begins by observing and collecting data from a human performer carrying out the mission/task of interest in a simulator. The observed and logged data are then manually parsed according to what data sequences apply to which predefined context. This recorded human performance data will serve as the fitness measure base that determines the appropriateness of the individuals being evolved. Fig. 2 shows a diagram of the components and their interactions in GenCL.

The source code individual in the GP module represents parts of the context base. It is fed into a *CxBR simulator* that executes it to produce the performance results of that individual. The behavior resulting from the CxBR simulator is then compared with the observed human performance (the *fitness function*) and a fitness measure is computed. Remember that, depending on the current learning task, an individual can describe either the action within a specific context or the set of rules that determine context activation (i.e., situational awareness). Accordingly, the population either contains individuals competing to represent the action within the context, or individuals competing to represent the context activation process. If the evolving context is not at the lowest context level, the individual might contain action knowledge from contexts at the lower level as part of its function set. The sequence in which contexts are evolved follows the principles of LLGP described earlier.

The GP process automatically builds the action knowledge within the contexts and the knowledge to apply the right context in a specific situation, thereby providing the CxBR context base with appropriate knowledge. The evolutionary process strives to minimize the discrepancies between the performances of the agents being evolved by the GP and the observed human performance. Fig. 3 describe the GenCL algorithm in detail.

An example of human traffic behavior is when a driver is approaching slower traffic on a rural two-lane road with intermittent head-on traffic present in the opposite lane. How different people behave in such a situation can vary significantly. Some drivers may decide to pass the slower traffic as soon as possible and thereby take risks, while others may be more careful and await a safer gap in oncoming traffic.

Let us presume that we have collected sample data when a driver is approaching slower traffic. The learning procedure begins by creating a GP population of randomly created source code individuals as described in Fig. 3. The first individual is placed in the CxBR simulator and the agent (with the individual's behavioral representation) is initialized with the same speed, position and heading as the sample where the driver is furthest away from the slower traffic. The CxBR simulator is then executed with the source code of the individual (i.e., its behavioral representation) for a number of simulation cycles (i.e., the number of cycles it took the driver to arrive to the next data sample selected). The simulation is now temporarily halted and the agent's speed and position are now compared to those observed of the real driver. Upon recording the discrepancies, if any, the simulation then proceeds unaltered until the next sample point and the deviation to the real driver is once again calculated and recorded. When all data samples have been covered, the deviation is averaged over all sample points and this will now constitute the fitness value for this individual. The same procedure

is then applied to all the individuals in the population and fitness values are computed and attached to all individuals. The GP reproduction process can begin and individuals are selected to produce the next generation of individuals based on their fitness values. This is done by applying genetic operators (e.g., crossover and mutation) with some probability to the selected individuals. If no genetic operator is applied to the selected individual, it will be cloned into the next generation. When a new population is created in the GP process, the simulation and evaluation of each individual will start all over again. This continues until the stopping criterion of the GP is reached (e.g., acceptable fitness is reached or number of generations executed).

In brief, the GP initially evolves the (pre-identified) lower level contexts of the agent by creating individuals that seek to match the data logged by the observed human expert. Once the lower level contexts are evolved, GenCL then uses them as a function to evolve higher-level contexts. The lower level context (i.e., a function that contains its action knowledge) can be part of the function set when the higher level context evolves its action knowledge or when the transitional knowledge for the lower level contexts evolves. In both cases, the lower level contexts will be part of the higher level context's complete behavior. In order to evolve the complete behavior of a context, all subcontexts that it could possibly use must be complete and functional. This is the essence of the Layered Learning algorithm described above.

When the actions involved in all contexts at a specific level as well as the parent context are evolved, the artifact turns its attention to the transitional knowledge that provides the agent's situational awareness. The transitional knowledge represents the criteria used to transition agent control from one context to the next. In order to simultaneously evolve all the transitional knowledge in the different contexts (Cooperative Co-evolutionary GP), it is necessary that the action knowledge for all the possible contexts be already developed and deployed. The fitness function for an individual in one of the populations is now a function of the individual itself and the best individuals so far from the other populations. A simulation now takes place where all of the different contexts involved (including the parent context) should be active sometime. The transition knowledge, in the form of rules (i.e., the best ones from the other populations and the one belonging to the individual under investigation), are now executed and at every simulation cycle one of them will activate its context and execute its action knowledge. During this simulation, the discrepancies with the real driver are recorded. These discrepancies will serve as the basis for the fitness value after completion of the simulation. When the transitional knowledge is evolved, it completes the evolution of the agent. The agent's behavior is now defined as the set of action knowledge and transitional knowledge evolved for the complete predefined context organization.

IV. Testing of Prototype and Evaluation of Concept

A prototype system was built and applied to automobile driving. A commercial, full-scale driving simulator was used to observe five different human test drivers drive a simulated car in an urban and rural terrain database. The drivers were university students with at least five years of driving experience and between 25 and 37 years of age. The simulator lacked motion capability, but was otherwise an exact replica of an automobile cab, with three screens in front of the driver and a rear view screen. Each driver was allowed to take one informal test drive to familiarize himself with the simulator, and was thereafter formally subjected to observation during a subsequent 30-minute drive. This collection of data is now referred to as the *training data set.* Approximately four months later, the same five drivers were subjected to a similar but not identical and slightly shorter (20 min) route for use as validation data (*validation data set)*. The training drive included city driving (speed limit 50 km/h) with 11 traffic lights and nine intersection turns and rural driving (speed limit of 70 km/h or 90 km/h). During the drive, the drivers experienced seven hazardous situations of varying severity. However, the tests described here only used city driving to validate the performance of the GenCL approach. The city driving portion of the data showed the most interesting behavioral patterns among the drivers, such as diversity and variability. Therefore, rural driving and hazardous situations were not used and will not be further discussed here.

All of the traffic lights in the training data set triggered a change of state when the driver passed a point 30 m in front of the light. Four of the 11 lights changed from red to green; six from green to yellow to red, and one remained green. Although the virtual city environment was the same in both data collection scenarios, the drivers took an alternative route in the validation run. Furthermore, the light state triggering distance was varied between 30, 35, and 40 m ahead of the light for the validation run. Hence, the drivers experienced different traffic light scenarios than they did in the training run.

The environment for the learning process was arranged to ensure that the behavioral patterns of the drivers were neither predictable nor trivial. One feature of human behavior is variability. An example of how to trigger unpredictable behavior from the drivers is found in a traffic light changing from green to yellow and then to red. If this change takes place when the car is at a certain distance from the light, the drivers will have to make a decision on whether to slow down to a stop when the light turns yellow, or continue and pass the light while yellow. A short investigation among the simulator operators showed that this distance to trigger diverse behavior among people seemed to be when the car is at a point 30 m in front of the light when driving at city traffic speeds. This triggering distance succeeded in capturing this variable behavior among the drivers who participated in these experiments. The same driver sometimes stopped at a light about to turn red while other times he continued on. Discrepancies among the different drivers could also be noticed, as some were more careful and decided to stop more often than other drivers.

It might be tempting to present the same situation to the actor several times and base the learning upon some average measure of the performance. The risk in these repetitive situations is that the human bases his or her action on prior knowledge of the situation and might not behave naturally. Conversely, letting the human act in a realistic environment with similar, but not identical, situations introduces disorder (as well as realism) to the training data. In two very similar situations, the actor might act differently. In the worst case, contradictory data might be introduced into the data set. If

learning from observation is to be rigorously implemented, this is an important issue to address. The learning conducted in this research left any disorderly and contradictory data within the data set to investigate how well GenCL handles them. The probability of successful learning will, of course, be reduced by this disorder. However, human behavior is not always consistent, nor based on some average performance.

The training data consist of:

— the observed driver's speed;
— distance to an intersection or a traffic light;
— two Boolean variables indicating, respectively, the presence of intersections (within 100 m) and traffic lights;
— a variable indicating the state of the traffic light when the car is within 100 m of the light.

During the collection of training samples, the simulator sampled the drivers' behavior at a rate of 10 Hz. This resulted in a data set larger than 5000 samples per driver, when in city traffic. To make the learning feasible, the data were reduced from these samples to less than 350 samples per driver over all the contexts evolved. This reduced the extent of the computations for determining the fitness value of each individual and made the process tractable. These samples were sufficiently complete to represent all possible situations within city driving with different state changes of traffic light, intersections and normal city driving. The training data were partitioned to contain a similar amount of data from the different contexts. This was done to ensure that the learning conducted would not favor any particular situation. The 350 total training samples were collected from all involved contexts. Training data for each context were chosen from typical situations for the specific context under investigation. The objective was to collect somewhere between 50 and 100 samples with a constant time period for each context. If the data set is too small, it will not be sufficient for learning and if too large, the computational complexity of the process increases. As different drivers behave differently among themselves and also from time to time and the availability of data, the number of samples came to vary between 25 and 80 samples per context to cover each driver's behavior pattern within a specific context. As an example, if **Traffic-Light-Driving** is to be learned, data from several different traffic light situations (e.g., stopping at light turning red, running yellow lights, passing traffic lights when making an intersection turn, etc.) were included in the training data to cover the complete behavior in that context. All the data used for learning can be found in Fernlund [11].

Note that the agent is unaware of any traffic rules and regulations. Its only means of learning is to mimic the driver's behavior. As an example, the agent is never told to stop at a red light but learns this if the observed driver stops at the lights turning red. As mentioned earlier, however, the drivers do not always stop when a light turns yellow, about to turn red.

Prior to learning, the empty context hierarchy is identified. The hierarchy used in this investigation is described in Fig. 4. This will give the GP learning algorithm a frame within which to operate. Details of the prototype are found in Fernlund [11].

When the context hierarchy is defined, the evolution of the agents can take place as per the GenCL algorithm. The reduced

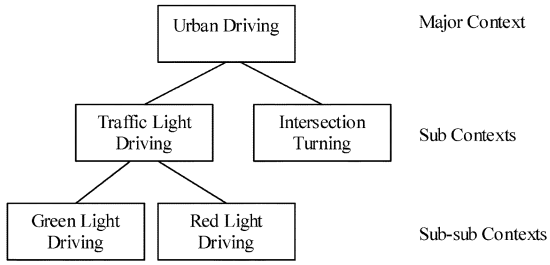


Fig. 4. Context hierarchy defined *a priori* for this application.

training data set is now used to compare the individual's performance during training with the actual driver's performance. The deviation in performance between the driver and the individuals is the base for the fitness function that propels the learning algorithm, as described in Fig. 3. Five agents were evolved from the observed data, one corresponding to each driver. They are labeled Agent A through Agent E, respectively, corresponding to Driver A though Driver E. Once evolved, the agents were subjected to the following evaluations.

1) **Learning capabilities test**: How well did the agent learn the behavior with which it was trained? The training data were used as the comparison benchmarks here. This was deemed to be the easiest of the tests.
2) **Generalization Test**: How well did the agent perform in situations different from those seen during training? The validation runs were used as the comparison benchmark in this set of tests.
3) **Long-term stability test**: How well did the agent perform after a substantial period of time in continuous operation? The agents' stability and reliability over a long period of time were tested here to ensure the absence of small error accumulation.
4) **Utility test**: This test compared the performance of two of the agents evolved through GenCL against two traditionally (i.e., manually) developed agents. The comparison was related to the agents' comparative fidelity, not to development time or effort. The behaviors of the GenCL-developed agent as well as that of the manually developed agent were compared to the validation data set in this test.

The *Learning capabilities test* is a simple test of comparing outputs from the agent with outputs from the human drivers when they are presented with the same inputs. This will provide a measure on how well the agents were able to handle the learning tasks. However, in order to investigate the agents' performance as autonomous entities it was necessary to test the agents' ability to perform in normal environment (i.e., running in a simulation). Merely testing the correctness of a model (represented by the agents) by validation of correct output for certain inputs to an evolved function was not deemed sufficient. Such a testing could show good performance (i.e., input/output mapping) but it would not reveal much about the agents' accumulated errors and their long-term stability. We need to ensure that the agents' performances in this simulated environment are comparable to the drivers', even after minutes and hours. This was done by using a simulation that exercised the agent's context base to let it freely and autonomously act in the environment.

Besides the potential accumulation of errors, another issue typically arises when GP is used as a learning algorithm. GP produces *noncoding regions* during the evolution of individuals. Noncoding regions in GP are code segments that are never accessed during the learning phase, but as the agents are inserted into a simulation, they could be occasionally triggered, causing the agents to behave erratically. Two different types of noncoding regions could exist in a program evolved by GP. The first one is code that never could be executed or redundant code. The other type of noncoding regions is when some branches of code are never tested during training but could very well be executed if the right conditions occur. To detect when the noncoding regions influence an agents' performance, the agents must be extensively tested as autonomous agents in an unpredictable environment. Even when noncoding regions create undesirable behavior, it has been proven that they can actually enhance the learning capabilities of GP [12], [26]. Evaluations of agents developed through GenCL are later indirectly evaluated for the presence of harmful noncoding regions. We now look at the results of the four evaluations and discuss their significance.

A. Learning Capabilities Test

Table I shows how well the agents learned their tasks when compared to the performance of their corresponding driver. This test used the training data as the comparison standard, thereby evaluating how well the agent learned their behaviors. To generate these results, inputs (i.e., presence of traffic lights or intersections, distance to lights or intersections and the status of the traffic lights) were presented to the agents and their corresponding outputs (i.e., speed) were compared to the actual drivers' cars' speed (i.e., the agent is treated as a black-box). Note that for the agents to produce an output, they need to evaluate the current situation (i.e., all the momentary inputs) and determine which context to activate. The active context will then produce the corresponding output. The output from the driver's car and the corresponding agent is the speed at a specific moment. Approximately 150 samples from the training data are used for each agent. If the correlation coefficient is close to 1.0, there is a high correlation between the two data sequences. If the correlation coefficient is close to zero, there is little correlation. A correlation coefficient close to -1.0 would indicate that there is a high inverse correlation. A high level of correlation was expected for this test, and the results showed exactly that.

A slightly worse performance can be detected for agent E when compared to the other agents. Investigation of driver E's behavior shows a slight inconsistency within the training set. Traffic light #2 is located 30 m after an intersection turn. This means that the speed of the driver at this point is rather low, and the most obvious action is to stop at the light that is about to turn red, as all the other drivers did. However, driver E ran light #2. His attention might have been on other things and he might not have spotted the light early enough. This might explain his decision to run the yellow light. This might complicate the learning task of agent E since driver E stops at other lights turning red, even when his approaching speed is higher. Note, furthermore, that the training data set lacks much information about the environment, such as buildings and other objects by the road that could have influenced the performance of the drivers.

TABLE I
LEARNING CAPABILITIES

	Speed deviation		Speed
	RMS [km/h]	Std. Dev.	Correlation
Driver A/Agent A	2.98	2.29	0.988
Driver B/Agent B	3.68	3.07	0.983
Driver C/Agent C	2.57	1.79	0.990
Driver D/Agent D	2.41	1.92	0.989
Driver E/Agent E	8.10	7.84	0.852

B. Generalization Test

This test involved a comparison of the agent's performance against the driver's in their validation test run. This test involved data not used in training and represented new situations not seen by the agents in training. The purpose was to evaluate the ability of the agents to generalize their performance.

During generalization testing, the agents operated as autonomous agents in a simulated environment, facing the same situations that their corresponding drivers experienced during the validation test run. The distance where the traffic lights changed their state was now varied and not kept constant at 30 m as in the training scenarios.

Table II depicts the qualitative decisions made by the agents vs. their corresponding drivers at those lights that turned from green to yellow to red. Note that some discrepancies now are evident. This was anticipated because the tests bring out the variable behavior of automobile drivers when the lights turn to red from yellow.

If we look at agent E and driver E, we can see that the behavior differs significantly vis-a-vis the traffic lights. In actuality, Driver E performed very differently in the validation run compared to the training run. He was very reckless and ran more yellow lights than did the other drivers in the training run. Conversely, he was very careful and stopped at almost every light in the validation run. Agent E's poor performance can be attributed to this inconsistent performance of driver E between the training and the validation runs. Such inconsistency is common human behavior, although it certainly makes it more difficult to learn one's "normal" behavior. The common way of learning algorithms to approach problem with inconsistent data is to average the output over these occasions. However, in tactical applications, an average behavior makes little sense unless one introduces an element of randomness into the behavior. It would be better for the agent to make the most probable decision than to make an average decision. If the training data sets are too small and limited, inconsistent data probably occurs more frequently. If more training data are available, then the most probable reaction from a specific person at any situation would be easier to determine. One cannot escape the fact that inconsistent behavior is always difficult to model and predict.

When we look at the quantitative comparisons in Table III, the two measurements of comparison, besides speed correlation, were speed and time deviation between the agents and their corresponding drivers. The agent is now operating autonomously in the environment and the deviation is continuously measured as the agent experiences different traffic lights and/or intersections and regular city driving. As the agent operates in the environment, the agent's speed is compared to the speed of the driver

TABLE II
QUALITATIVE COMPARISON OF THE DRIVERS/AGENTS PERFORMANCE

	Light 1	Light 3	Light 4	Light 5	Light 6	Light 7
Driver A/Agent A	R/R	OK	S/S	R/R	R/R	S/R
Driver B/Agent B	R/R	OK	S/S	R/R	R/R	R/R
Driver C/Agent C	S/S	OK	S/S	S/S	S/S	S/S
Driver D/Agent D	R/R	OK	S/S	R/R	S/R	S/R
Driver E/Agent E	R/R	OK	S/S	S/R	S/R	S/R

S stands for stop and R for running the light while it's still yellow. OK indicates that the agent performs in accordance to the driver at those lights that turned from red to green.

TABLE III
SPEED AND TIME DEVIATION DURING THE VALIDATION TESTING

	Speed deviation [km/h]		Time deviation [s]		Speed
	RMS	Std. Dev.	RMS	Std. Dev.	Correlation
Driver A/Agent A	7.47	7.44	1.47	1.47	0.880
Driver B/Agent B	7.14	6.19	2.56	1.75	0.896
Driver C/Agent C	7.12	7.11	3.60	2.80	0.926
Driver D/Agent D	10.5	9.23	9.10	6.78	0.712
Driver E/Agent E	17.0	12.0	38.4	30.3	0.550

at each position. This is a measurement of the agent's momentary performance at the different locations in the environment. As the agent is autonomous, it will take the agent some time to reach a certain position in the environment. The same is true for the driver. Hence, by measuring the deviation in time to reach each specific position in the environment we get a measurement of the agent's performance *vis-à-vis* the driver's. The time deviation will then serve as a measure of the long-term deviation between the driver and the agent.

Looking at the speed and time deviations of the validation run in Table III, agents A, B, and C perform very well. Agent E, on the other hand, does not perform well at all. This can be explained by the afore-mentioned inconsistency of driver E between the training run and the validation run. The time deviations are affected rather dramatically after a traffic light when the stopping behavior at that traffic light differs between the driver and the agent. This is because the time for the light to turn green will be added to the time deviation.

It should be reiterated here that the objective is to train the agent to act as the observed driver, and not to evolve the best skills for car driving. If the observed driver is drunk, the agent should evolve drunken driving behavior. In other words, the only thing that measures how good the agent learned its task is how similar its behavior is to the observed driver.

The results for agent A are quite good even when its misbehavior at one of the traffic lights is kept in the comparison. However, a somewhat worse performance can be observed from agent D. The misbehavior of agent D is more interesting to analyze because during the validation tests agent D actually runs some yellow lights where the driver D actually stops. The suspected cause of this was the constant light state-triggering distance of 30 m in the training data for Agent D. Since GP is evolving source code, it is possible to investigate the evolved code to determine the cause of the agent's action.

Fig. 5 shows part of the code evolved in Agent D that controls the activation of the **Traffic-Light-Driving** context. Here we can see that if the agent is about to make a turn at an intersection where the traffic light is, it will immediately activate the

```
if(distance<76.485488)
{
  if(!lightPresent)
        return 0;
  else
  {
        ...
  }
}
else
{
  if(intTurning && lightPresent)
        return 1;
  else
        return 0;
}
```

Fig. 5. Part of agent D's evolved code for Traffic-Light-Driving activation.

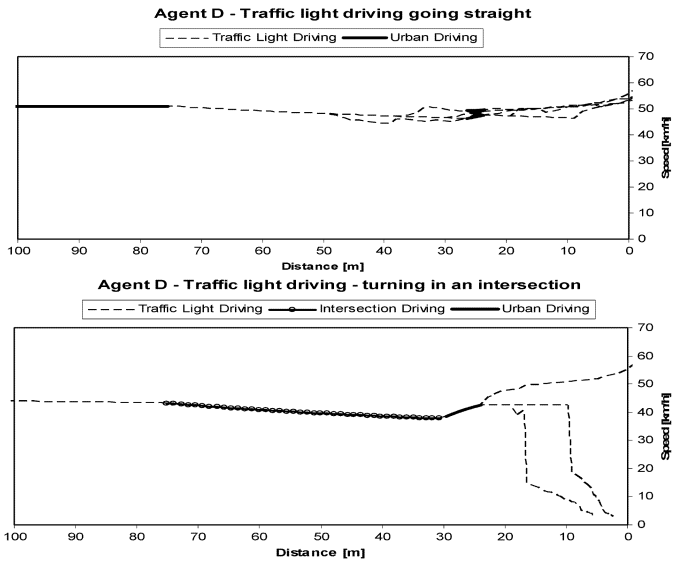


Fig. 6. Behavior of agent D at traffic lights changing from green to red.

Traffic-Light-Driving context. If not, Agent D does not consider anything before the agent is closer than 76.5 m to the light. With this knowledge, we conducted a test where the agent was presented to two different traffic lights. In one, the agent was to

TABLE IV
AGENTS' LONG TERM BEHAVIOR

	Light turning Red		Light turning Green
	Average Stopping Distance	Standard Dev.	Correct behavior
Agent A	34.7	12.9	20/20
Agent B	8.04	1.95	22/22
Agent C	5.89	1.03	8/8
Agent D	4.50	1.31	6/6
Agent E	13.5	0.551	11/11

turn at the intersection where the light is located. In the other, it was to continue straight. The agent was set to approach each light six times and the distance that triggers the light to change from green to yellow was varied between 100, 50, 40, 30, 20, and 10 m.

In Fig. 6, we can see that if Agent D is about to make a turn in the intersection, it activates the **Traffic-Light-Driving** context early to lower the speed of the agent. Furthermore, when it is closer than 76.5 m, it will activate the **Intersection-Turning** context to further slow down the car. If the traffic light changes its state when the agent is far enough from the light, the agent will stop at the light. Note that each diagram in Fig. 6 consists of six different lines but at several occasions they are overlaid since the behavior of the agent was the same. Note that the agent needs to release the activation of one subcontext for another subcontext to get activated. In this transition, the major context will temporarily get activated (i.e., **Urban-Driving** in this case).

When the agent is going straight, on the other hand, the **Traffic-Light-Driving** context will not be activated until the agent is closer than 76.5 m to the light. In this case, the speed of the agent will be too high for the agent to be able to stop at the light with the help of the **Traffic-Light-Driving** context. Consequently, the agent runs the yellow light. The interesting result of this investigation is that the poorly behaving Agent D does not trigger on the obvious 30-m mark. Rather, the action is triggered by whether Agent D is going to make a turn or not at the intersection with the light. Upon inspection of the training data for Agent D, it reveals that Driver D actually stops at all lights (that turn yellow) where he is going to turn, while he runs through all those in which he doesn't turn. Hence, there is a correlation between stopping at the yellow light and whether the driver is making a turn or not. The GP found this relation and learned it. The problem could have been addressed by including richer training scenarios, where the driver had a more varied experience to be observed.

This analysis shows that the use of GP as the learning algorithm that produces knowledge in an interpretable and analyzable manner could be used to discover nontrivial behavior patterns of the observed human.

The generalization test shows that agents A, B, and C generalize the problem very well and are able to handle new situations. Worse performance can be detected for agent D because of a lack of richness in the training data (i.e., the training data lacked occurrences where the driver stopped at a traffic light without making a turn). Agent E does not perform well because Driver E was highly inconsistent with himself. The problem with agent D and agent E actually derive from the issue of designing appropriate training scenarios and the collection of training and validation data.

C. Long Term Stability Test

The long-term stability test was conducted to investigate whether the agents exhibit stable behavior even after an arbitrarily long simulation run. When the knowledge and behavior within the agent is developed by a machine learning algorithm it is important to investigate if the cumulative errors are increasing over time resulting in unstable behavior.

The five agents were allowed to operate within the simulated environment for 40 min of real time, pass more then 60 traffic lights and 25 intersections. The agents were exposed to a variety of traffic light scenarios where none were the same as the others. To be able to compare their long term stability, their behavior was recorded when the traffic light ahead of them was either yellow or red, since this is one of the occurrences where variable behavior was detected previously.

If an agent was not stable enough to invoke **Intersection-Turning** when making a turn, the agent will approach the turn too fast and will leave the road. If this happened, the agent would be stuck, since no recovery algorithm was implemented for the agent to find a way back to the road. Hence, the fact that all the agents were still running after 40 min proves stability of the agents in terms of activating the **Intersection-Turning** Major Context.

Since the traffic lights now change their states at different distances (i.e., the lights are time triggered and not related to the agents distance) and agents might approach the lights at different speeds, it is difficult to make an exhaustive statistical analysis of their behavior. Table IV shows a simple compilation of the agents' behavior when they approach lights that are either yellow or red. Two different events occur: the light turns from green to red or from red to green.

As the agents come to a stop at the red lights, a comparison could be made on their different stopping distances. Table IV shows that all the agents (except agent A) stop at almost the same distance every time. Therefore, their standard deviation on the stopping distance is small. Agent A stops at a different distance almost every time. Actually, the surprising fact is that the other four agents manage to generalize so well that they stop at approximately the same distance to the lights, even when the times of the light changes are different. In all the training data presented to the agents during learning, lights changed their state when the driver was at a point 30 m in front of the light. Hence, all the agents stopped consistently at the same distance during training (approximately 30 m after the light turns from green to yellow). So, the most obvious thing would have been for the agents to not generalize as well as they did, but rather to stop 30 m after the light changes from green to yellow. The objective here, however, is not to compare their distance to the traffic light

TABLE V
COMPARING EVOLVED AND ENGINEERED AGENTS IN THE VALIDATION ENVIRONMENT

	Speed [km/h]		Time [s]		Speed
	RMS	Std. Dev.	RMS	Std. Dev.	Correlation
Engineered agent C vs. Driver C	8.52	8.38	4.05	3.10	0.902
Evolved agent C vs. Driver C	7.12	7.11	3.60	2.80	0.926
Engineered agent D vs. Driver D	9.02	8.64	7.43	7.21	0.876
Evolved agent D vs. Driver D	10.5	9.23	9.10	6.78	0.712

after stopping. Rather, the objective is to measure the agents' consistency in their behavior. All agents behave consistently and stop close to the light except agent A, who slows down and stops when the light turns red. Even if the behavior is different than the other agents, it is still consistent.

The final observations on the agents' long-term performances are their behavior when approaching red traffic lights turning green. Two observations can be made here. The first thing that reflects correct behavior on the part of the agents is that they do not stop at the red light when they are far from the light. The other is that they lower the speed as they get closer and that they pick up speed when the light turns green. The column that describes the correct behavior at a light turning green in Table IV compares the number of correct behaviors to the total numbers of lights turning green experienced by each agent. All agents show correct behavior all the time as they approach a red light about to turn green.

This test indicates that the agents show consistent and stable performance throughout the long-term stability test. Four of the five agents perform more consistently in traffic light driving than could be expected with regards to the training data presented during the learning phase (i.e., the constant 30-m stopping distance). This also infers that the noncoding regions in GP are not affecting the agents' performance in a significant manner.

D. Utility Test

In order to determine how useful the automatic creation of simulated agents through GenCL is, two agents were manually developed by an independent human developer in the traditional manner. We want to compare the new approach of building simulated agents with human behavior to the traditional way of agent building. In the traditional agent building task, the knowledge engineer interviews the human and if possible allows him to act in his normal environment and observe his behavior. Both methods aim to create general applicable agents. Hence, the developer interviewed and rode with two of the original drivers - C and D. The driving part of the knowledge acquisition lasted for approximately 45 min in city driving (the 30 min in the driving simulator covered both city and rural driving). This is slightly more time than the drivers spent in the driving simulator for the training set of data. One might note that collecting appropriate data in the real world is considerably more complicated that in a simulator. The simulator environment can be configured to focus on specific and interesting events. In the real world, however, one can only try to act so those events might occur but it might be difficult to achieve. However, the objective is to ensure that the new way of building agents does not deteriorate agent behavior compared to traditional agent modeling.

The developer knew that his agents would be compared to those developed by the GenCL system and was told to focus on the behavior patterns so far implemented by the GenCL (i.e., Traffic-Light-Driving, Intersection-Turning, and Urban-Driving). Hence, the initial knowledge available to the human developer was the same as available to GenCL (i.e., the same empty context organization). The task for the developer was to collect knowledge through interviews and by observing the drivers drive a real car. Of course, the virtual city used in the simulator runs does not exist. Therefore, a route through the city of Linköping, Sweden was chosen that was reasonably similar to the virtual one. The developer was then to model the drivers' specific behavior as they drove in city traffic with specific focus on intersections and traffic lights. After the knowledge was collected and analyzed, two agents were developed and inserted in the same CxBR traffic simulator as the agents developed by GenCL. Note that the development was done by an independent researcher without any influence from those developing GenCL. Now the two different approaches to building human behavior models, implemented in simulated agents, could be compared. Table V compares the agents developed by the human developer and that of the GenCL agents to the driver's behavior in the Driving Simulator.

Comparing GenCL Agent C and the engineered Agent C with Driver C, the GenCL agent performs consistently better than the engineered Agent C in the validation environment. Comparing the D agents to Driver D, the GenCL agent performs slightly worse than the engineered Agent D in the validation environment. The interesting result here is that the GenCL agent is able to perform nearly as well as an agent developed by traditional means even when the GenCL Agent D was seriously affected by the lack of richness in the training data, as previously described in Section IV-B. Hence, the GenCL algorithm is able to construct agents with performance at least comparable to that of agents developed in the traditional way under the worst of circumstances, and better under normal circumstances.

The development time (including preparations, knowledge acquisition, knowledge processing and implementation) when the agents were developed with the knowledge engineering approach was three weeks of full time workload, or 120 person hours of effort. To evolve one agent (including knowledge in all contexts) on a Pentium 4, 1.8-GHz machine with 512-MB internal memory takes less than 36 h. Adding another five hours for the preparation of the driving simulator and five more for preprocessing the data for the GenCL algorithm brings the time estimate for creating an agent with GenCL to 46 person hours of effort. This is roughly one-third of the manual effort. Because GenCL evolves source code statements, the knowledge evolved is ready for use as soon as the learning phase is complete. Note

that this is a rough approximation, since the experiments were conducted at the same time as the new methodology was being developed.

V. Summary and Conclusions

We have proposed, described and tested a new approach to learning human performance in a tactical situation. The results indicate that this technique can be successfully used to create agents that model the human performance of tactical tasks and the accompanying decision-making. Our approach employs Context-based Reasoning and Genetic Programming (CxBR+GP). These agents are built automatically with data logged while a human expert performs a (tactical) mission on a simulator. We refer to this process as learning from observation. A prototype incorporating this algorithm, called GenCL, is able to learn and generalize the actions of the humans being observed in a simulator. Furthermore, the performance of the GenCL algorithm is comparable to agents developed by the traditional knowledge engineering approach. The evaluation of the agents was done at several levels. First, the agent's output for a specific set of inputs was compared to that of their corresponding human drivers to evaluate their ability to exhibit the same driving tendencies as their respective driver. Secondly, we compared the output of the agent performing the task in a simulation to that of its corresponding driver in a previously executed validation run. The agents were never exposed to this validation run during learning. These validation runs by the humans were performed four months after the original training run and they included significant differences from the training runs. This was done to gauge the agents' ability to generalize by subjecting them to situations not seen during training. Thirdly, the agents were subjected to a long term performance test. Here the agents were placed into a simulation run for an arbitrarily long time to determine whether they show stable and consistent performance. Lastly, models of two drivers were derived in the traditional knowledge engineering fashion and agents were built "by hand" from these models. These hand-created agents were then compared in performance to the actual drivers as well as to the original evolved agents. This comparison showed that the performance of the evolved models compared favorably to that of the engineered models.

As a matter of hindsight, we had not used the simulation in the learning phase for the very first training sessions. Consequently, the GP had only learned to map correct outputs to given inputs. This input-output mapping introduced severe instability in the initial evolved agents, since no mechanism was present to prune the harmful noncoding regions. We instituted the use of the simulation after experiencing the poor initial results without it. By using a CxBR simulation in the learning phase, accumulated errors and discrete deviations in the behavior affected the fitness value and pruned noncoding regions that can worsen agent performance. Hence, the initial input-output mapping algorithm was discarded and replaced by the one we now call GenCL.

The results obtained suggest that this approach is viable for building agents with human-like behavior for use in training simulations or for analysis. This approach could potentially save significant time and effort. Furthermore, the agents evolved here possess individual human behavior patterns. Prior research in this area has focused on creating optimal behavior from human performance. Our results show that we now can rapidly model for individual humans with highly personal and often far from optimal behavior. Improvements could be achieved by evolving the agents in a massively parallel architecture. The results presented here encourage further research concerning this new learning methodology.

References

[1] E. Abravanel and S. Ferguson, "Observational learning and the use of retrieval information during the second and third years," *J. Genetic Psych.*, vol. 455, no. 2, Dec. 1998. v.159, 4.

[2] E. J. Bass, J. P. Zenyuh, R. L. Small, and S. T. Fortin, "A context-based approach to training situation awareness," in *Proc. 3rd Annu. Symp. Human Interaction with Complex Systems*. Los Alamitos, CA, 1996, pp. 89–95.

[3] D. Bentivegna and C. Atkeson, "Learning from observation using primitives," in *Proc. 2001 IEEE Int. Conf. Robotics & Automation*, Seoul, Korea, May 21–26, 2001.

[4] C. C. Bojarczuk, H. S. Lopes, and A. A. Freitas, "Genetic programming for knowledge discovery in chest-pain diagnosis," *IEEE Eng. Med. Biol. Mag.*, vol. 19, no. 4, pp. 38–44, Jul.–Aug. 2000.

[5] P. Brezillon, "Context-based intelligent assistant systems: A discussion based on the analysis of two projects," in *Proc. 36th Annu. Hawaii Int. Conf. System Sciences*, Jan. 6–9, 2003, pp. 83–91.

[6] R. Calder, J. Smith, A. Coutemanche, J. Mar, and A. Ceranowicz, "ModSAF behavior simulation and control," in *Proc. 3rd Conf. Computer Generated Forces and Behavioral Representation*, Orlando, FL, Mar. 1993, pp. 347–356.

[7] J. Demiris, S. Rougeaux, G. M. Hayes, L. Berthouze, and Y. Kuniyoshi, "Deferred imitation of human head movements by an active stereo vision head," in *Proc. 6th IEEE Int. Workshop on Robot Human Communication*, Sendai, Japan, Sep. 1997, pp. 88–93.

[8] S. Deutsch, "Notes taken on the quest for modeling skilled human behavior," in *Proc. Third Conf. Computer Generated Forces and Behavioral Representation*, Orlando, FL, Mar. 17-19, 1993, pp. 359–365.

[9] J. Eggermont and J. van Hemert, "Adaptive genetic programming applied to new and existing simple regression problems," in *Proc. 4th European Conf., EuroGP 2001*, Lake Como, Italy, Apr. 2001.

[10] S. Eklund, "High performance computer architecture for genetic programming," in *Proc. Conf. Promotion of Research in IT at New Universities and at University Colleges in Sweden*, Ronneby Brunn, Sweden, Apr. 23–25, 2001.

[11] H. Fernlund, "Evolving Models from Observed Human Performance," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Central Florida, Orlando, 2004.

[12] S. Garcia, J. Levine, and F. Gonzalez, "Multi Niche parallel GP with a junk-code migration model," in *Proc. 6th European Conf. Genetic Programming (EuroGP 2003)*, 2003.

[13] A. J. Gonzalez and R. H. Ahlers, "A novel paradigm for representing tactical knowledge in intelligent simulated opponents," in *Proc. IAE/AIE Conf.*, Austin, TX, May 31–June 3 1994.

[14] ——, "Context-based representation of intelligent behavior in training simulations," *Trans. Soc. Comput. Sim. Int.*, vol. 15, no. 4, 1998.

[15] A. J. Gonzalez, M. Georgiopoulos, R. F. DeMara, A. Henninger, and W. Gerber, "Automating the CGF model development and refinement process by observing expert behavior in a simulation," in *Proc. Computer Generated Forces Conf.*, Orlando, FL, 1998.

[16] A. J. Gonzalez, R. F. DeMara, and M. Georgiopoulos, "Vehicle model generation and optimization for embedded simulation," in *Proc. Spring 1998 Simulation Interoperability Workshop*, Orlando, FL, Mar. 1998.

[17] A. J. Gonzalez, W. J. Gerber, and J. Castro, "Automated acquisition of tactical knowledge through contextualization," in *Proc. 2002 Computer Generated Forces and Behavior Representation Conf.*, Orlando, FL, May 7–9, 2002.

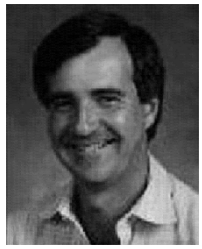
[18] R. V. Guha, "Contexts: A Formalization and Some Applications,", MCC Tech. Rep. ACT-CYC-423-91, Dec. 1991.

[19] A. Henninger, A. J. Gonzalez, W. Gerber, M. Georgiopoulos, and R. DeMara, "On the fidelity of SAFs: Can performance data help?," in *Proc. Inter-Service/Industry Simulation and Education Conf.*, Orlando, FL, 2000.

[20] A. Henninger, "The Use of Neural Network Based Movement Models to Improve the Predictive Utility of Entity State Synchronization Methods for Distributed Simulations," Ph.D. dissertation, Univ. Central Florida, Orlando, 2001.
[21] W. H. Hsu and S. M. Gustafson, "Genetic programming for layered learning of multi-agent tasks," in *Proc. Genetic Evolutionary Computation Conf. (GECCO) 2001*, San Francisco, CA, Jul. 9–11, 2001.
[22] J. R. Koza, *Genetic Programming*. Cambridge, MA: MIT Press, 1992.
[23] M. van Lent and J. Laird, "Learning by observation in a complex domain," in *Proc. Eleventh Workshop on Knowledge Acquisition, Modeling and Management*, Alberta, Canada, Apr. 1998.
[24] S. Luke, "Genetic programming produced competitive soccer softbot teams for RoboCup-97," in *Proc. 3rd Annu. Genetic Programming Conf.*. Los Altos, CA, 1998, pp. 204–222.
[25] J. McCarthy, *Proc. 13th IJCAI—Notes on Formalizing Context*, vol. 1, 1993, pp. 555–560.
[26] P. Nordin, W. Banzhaf, and F. D. Francone, "Introns in nature and in simulated structure evolution," in *Proc. Bio-Computing and Emergent Computation*, 1997.
[27] D. Ourston, D. Blanchard, E. Chandler, and E. Loh, "From CIS to software," in *Proc. 5th Conf. Computer Generated Forces and Behavior Representation*, Orlando, FL, May 1995, pp. 275–285.
[28] R. W. Pew and A. S. Mavor, *Modeling Human and Organizational Behavior: Applications to Military Simulations*. Washington, DC: National Academy Press, 1998.
[29] D. Pomerleau and T. Jochem, "Rapidly adapting machine vision for automated vehicle steering," *IEEE Expert: Special Issue on Intelligent System and Their Applications*, vol. 11, no. 2, pp. 19–27, Apr. 1996.
[30] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Proc. 3rd Conf. Parallel Problem Solving from Nature: Parallel Problem Solving from Nature*, Oct. 9–14, 1994, pp. 249–257.
[31] C. Sammut, S. Hurst, D. Kedzier, and D. Michie, "Learning to fly," in *Proc. 9th Int. Conf. Machine Learning*, Aberdeen, U.K., pp. 335–339.
[32] T. Sidani and A. Gonzalez, "A framework for learning expert knowledge through observation," *Trans. Soc. Comput. Sim. Int.*, vol. 17, no. 2, pp. 54–72, 2000.
[33] S. H. Smith and M. D. Petty, "Controlling autonomous behavior in real-time simulation," in *Proc. Second Conf. Computer Generated Forces and Behavior Representation*, Orlando, FL, Mar. 1992, pp. 27–40.
[34] J. Takamatsu, H. Tominaga, K. Ogawara, H. Kimura, and K. Ikeuchi, "Extracting manipulation skills from observation," in *Proc. Int. Conf. Intelligent Robot and Systems (IROS)'00*, vol. 1, Kagawa, Japan, 2000, pp. 584–589.
[35] R. M. Turner, "Context-sensitive reasoning for autonomous agents and cooperative distributed problem solving," in *Proc. 1993 IJCAI Workshop on Using Knowledge in Its Context*, Chambéry, France, 1993.
[36] ——, *A Model of Explicit Context Representation and Use for Intelligent Agents*. Berlin, Germany: Springer-Verlag, 1999.
[37] D. H. Wolpert and W. G. Macready, "No Free Lunch Theorems for Search,", Tech. Rep. SFI-TR-95-02-010, 1995.
[38] ——, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, 1997.

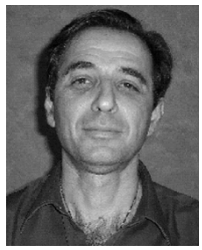
Hans K. G. Fernlund received the Bachelor's degree in computer engineering from the Dalarna University, Borlänge, Sweden, in 1993, the M.S. degree in computer engineering from the University of Central Florida (UCF), Orlando, in 1995, and the Ph.D. degree from UCF in 2004.

He is currently an Assistant Professor at Dalarna University.



Avelino Gonzalez (SM'78) received the B.S. and M.S. degrees from the University of Miami, Coral Gables, FL, in 1973 and 1974, respectively, and the Ph.D. degree from the University of Pittsburgh, Pittsburgh, PA, in 1979, all in electrical engineering.

He is a Professor in the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, specializing in artificial intelligence and simulation.



Michael Georgiopoulos (M'86–SM'01) received the electrical engineering diploma from the National Technical University, Athens, Greece, in 1981, and the M.S. and Ph.D. degrees in electrical engineering from the University of Connecticut, Storrs, in 1983 and 1986, respectively.

In 1987, he joined the University of Central Florida, Orlando, where he is currently a Professor in the Electrical Engineering program of the School of Electrical Engineering and Computer Science.

Dr. Georgiopoulos is a member of the International Neural Network Society and a member of the Technical Chamber of Greece. He has been an Associate Editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS since 2001.

Ronald F. DeMara (M'87–SM'03) received the B.S.E.E. degree from Lehigh University, Bethlehem, PA, in 1987, the M.S.E.E. degree from the University of Maryland, College Park, in 1989, and the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, in 1992.

Since 1993, he has been a full-time faculty member at the University of Central Florida, Orlando, in the Department of Electrical and Computer Engineering and has also served as Associate Chair. He was previously Associate Engineer at IBM Federal and Complex Systems Division and also a Visiting Research Scientist at NASA Ames Laboratory.

Dr. DeMara is an Associate Editor of the *Journal of Circuits, Systems, and Computers* and the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS. He is a Member of ACM and ASEE.