

Electronic Theses and Dissertations, 2004-2019

2004

Evolving Models From Observed Human Performance

Hans Karl Gustav Fernlund
University of Central Florida

 Part of the [Electrical and Computer Engineering Commons](#)
Find similar works at: <https://stars.library.ucf.edu/etd>
University of Central Florida Libraries <http://library.ucf.edu>

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2004-2019 by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

STARS Citation

Fernlund, Hans Karl Gustav, "Evolving Models From Observed Human Performance" (2004). *Electronic Theses and Dissertations, 2004-2019*. 90.
<https://stars.library.ucf.edu/etd/90>

EVOLVING MODELS FROM OBSERVED HUMAN PERFORMANCE

by

HANS KARL GUSTAV FERNLUND
B.S. Dalarna University, 1993
M.S. University of Central Florida, 1995

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical and Computer Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2004

ABSTRACT

To create a realistic environment, many simulations require simulated agents with human behavior patterns. Manually creating such agents with realistic behavior is often a tedious and time-consuming task. This dissertation describes a new approach that automatically builds human behavior models for simulated agents by observing human performance. The research described in this dissertation synergistically combines Context-Based Reasoning, a paradigm especially developed to model tactical human performance within simulated agents, with Genetic Programming, a machine learning algorithm to construct the behavior knowledge in accordance to the paradigm. This synergistic combination of well-documented AI methodologies has resulted in a new algorithm that effectively and automatically builds simulated agents with human behavior. This algorithm was tested extensively with five different simulated agents created by observing the performance of five humans driving an automobile simulator. The agents show not only the ability/capability to automatically learn and generalize the behavior of the human observed, but they also capture some of the personal behavior patterns observed among the five humans. Furthermore, the agents exhibited a performance that was at least as good as agents developed manually by a knowledgeable engineer.

*This dissertation is dedicated to my beloved parents,
Erik Gustav Fernlund and Ingrid Elisabet Fernlund.*

ACKNOWLEDGMENTS

First, I would like to thank the founder of this project, the Knowledge Foundation. Special thanks to Owen Eriksson, Nils Nordqvist and Ellus J. O. Brorsson, the people who provided the means for me to be part the Knowledge Foundation's promoting research at Sweden's new universities and university colleges.

Special thanks also goes to my adviser Avelino J. Gonzalez who supported me gratefully throughout the process of this dissertation. I would also like to thank the rest of my committee, Michael Georgiopoulos, Ronald F. DeMara, Annie S. Wu and Michael Proctor, for their support.

Special recognition goes to my sparring partner Sven E. Eklund who provided me with helpful ideas and feedback on my work.

TABLE OF CONTENTS

LIST OF FIGURES	x
LIST OF TABLES	xii
LIST OF ABBREVIATIONS	xiv
CHAPTER 1: INTRODUCTION	1
CHAPTER 2: PROBLEM DEFINITION	5
2.1. Problem Statement	5
2.1.1. Criteria for Correct Personalized Behavior	6
2.2. Hypothesis	6
2.3. Contributions	7
CHAPTER 3: BACKGROUND	9
3.1. Representing Human Behavior	9
3.1.1. Frameworks for Human Behavioral Modeling	14
3.1.2. Context-Based Reasoning	17
3.1.2.1. The CxBR Framework	21
3.2. Learning by Observation	24
3.2.1. Prior Work in the Field of Learning by Observation	26
3.3. Different Machine Learning Approaches	30

3.3.1. Reinforcement Learning	31
3.3.2. Inductive Learning	33
3.3.2.1. Decision Trees	33
3.3.3. Connectionist Learning	35
3.3.3.1. Artificial Neural Networks	36
3.3.4. Evolutionary Learning	38
3.3.5. Genetic Programming	43
3.3.5.1. GP Selection	45
3.3.5.2. Representation of the Individuals and Genetic Operators	48
3.3.5.3. Problem Space and Search Space	52
3.4. Summary	53
CHAPTER 4: LEARNING BY OBSERVATION – CONCEPTUAL APPROACH	55
4.1. Choosing an Appropriate Paradigm for Simulated Agents Exhibiting Human Behavior .	56
4.2. Learning by Observation with CxBR	59
4.3. Choosing an appropriate learning algorithm for CxBR	60
4.3.1. Reinforcement Learning	60
4.3.2. Using Different Machine Learning Approaches	61
4.3.2.1. Transforming the Search Space	62
4.3.2.2. Transparency	63
4.3.3. Using GP to Implement Learning in CxBR	64
4.4. Employing CxBR and GP towards Learning by Observation	66
4.5. Empirical Studies – CxBR and GP	70

4.5.1. In Depth Study of CxBR.....	70
4.5.2. Formalizing GP.....	73
4.6. Integrating CxBR and GP.....	76
4.6.1. Plan for the Integration.....	76
4.6.2. Strategy for Learning within CxBR.....	79
4.7. The GenCL Algorithm.....	82
4.8. The Synergistic CxBR – GP Integration.....	85
4.9. Summary.....	86
CHAPTER 5: THE MODEL BUILDING PROCESS.....	87
5.1. Data Collection.....	88
5.1.1. Selecting and Partitioning Training Data.....	97
5.2. Configuring the Context Base Prior to Learning.....	100
5.3. Experimental Test-Bed – The GenCL Artifact.....	101
5.3.1. GP Implementation.....	101
5.3.1.1. Population Initialization.....	102
5.3.1.2. Selection Methods.....	102
5.3.1.3. The Model of a Car Model.....	104
5.3.1.4. The Fitness Function – A Micro Simulation.....	106
5.3.2. GP Configuration.....	109
5.3.3. Evolving the Models.....	112
5.4. Summary.....	115
CHAPTER 6: RESULTS AND CONCLUSIONS.....	116

6.1. Evaluation Criteria	116
6.2. Test of Learning Capabilities	119
6.3. Test of Generalization	123
6.3.1. Generalization in the Training Environment	125
6.3.2. Generalization in the Validation Environment	130
6.3.2.1. Richness Analysis of Training Data	132
6.3.2.2. Capturing Individual Behavior	138
6.4. Long-term Reliability Test	142
6.5. Test of Usefulness	146
6.6. Ease of Use Evaluation	150
6.7. Experiments: Summary and Conclusions	155
CHAPTER 7: SUMMARY, CONCLUSIONS AND FUTURE WORK	156
7.1. Research Observations	157
7.1.1. The Relationship to Reinforcement Learning	158
7.1.2. Initial Stability Problems	159
7.1.3. Implementation to Maximize Computational Power	162
7.2. Conclusions	163
7.3. Future Work	164
APPENDIX A: LONG-TERM RELIABILITY	168
APPENDIX B: SENTINEL RULES OF AGENT B AND AGENT D	176
APPENDIX C: COMPLETE DESCRIPTION OF THE DATA SETS	178
APPENDIX D: TRAINING DATA	187

APPENDIX E: EVOLVED CODE	200
LIST OF REFERENCES	213

LIST OF FIGURES

Figure 1: Modified stage model.....	15
Figure 2: The CxBR hierarchical structure.....	19
Figure 3: Context-Based Reasoning in simulation	22
Figure 4: Decision tree example	34
Figure 5: The simple neuron model.....	37
Figure 6: Neuron connections in Multi Layered Perceptron, Hopfield Net and ART 1... 38	
Figure 7: Evolutionary Algorithms.....	39
Figure 8: Taxonomy of GA/GP applications.....	41
Figure 9: The iterative GA/GP search process	44
Figure 10: Fitness Uniform Selection.....	48
Figure 11: A possible GP individual's instruction tree and its corresponding C-code.....	48
Figure 12: Two examples of linear representation of GP individuals	49
Figure 13: Crossover creating one of the two possible offspring.....	50
Figure 14: Mutation example.....	50
Figure 15: Learning by Observation with GenCL	67
Figure 16: The GenCL algorithm	83
Figure 17: Virtual Technologies' Driving Simulator in use.....	88

Figure 18: City driving training data	93
Figure 19: The setup of the city driving during validation scenarios.	94
Figure 20: The context hierarchy of basic city driving.....	100
Figure 21: The model of a car model.....	105
Figure 22: Comparison of individual’s micro-simulation and Driver B.....	108
Figure 23: Example of evolved code – Intersection turning, agent C	114
Figure 24: Comparison at closest position, with same sample rate but different speed .	125
Figure 25: Behavior of agent B and Person B in the training environment.....	126
Figure 26: Agent D’s behavior at traffic lights.....	134
Figure 27: Agent B’s behavior at traffic lights	136
Figure 28: Behavior of agent C when repeatedly approaching traffic lights 7 and 8	143
Figure 29: Learning through input – output mapping.....	160
Figure 30: Learning through micro-simulation.....	162
Figure 31: Long term behavior of agent A	170
Figure 32: Long term behavior of agent B.....	171
Figure 33: Long term behavior of agent C.....	172
Figure 34: Long term behavior of agent D	173
Figure 35: Long term behavior of agent E.....	174
Figure 36: The city driving during the collection of the training data set	180
Figure 37: The rural driving during the collection of the training data set.....	182
Figure 38: The city driving during the collection of the validation data set.....	183

LIST OF TABLES

Table 1 Behavior of the five drivers at the six traffic lights that changed from green to red.	91
Table 2 Changes of traffic lights during the validation data collection.....	95
Table 3 Behavior of the five drivers at the lights in the validation set, changing from green to red	96
Table 4 GP configuration during the experiments.....	111
Table 5 Fitness values for the five evolved agents	120
Table 6 Learning capabilities.....	121
Table 7 Qualitative validation of the agents in the training environment.....	127
Table 8 Relationship between the agents and the drivers in the training set environment.	129
Table 9 Qualitative comparison of the drivers / agents performance	131
Table 10 Speed and time deviation during the validation testing.....	131
Table 11 Behavior of drivers B and E at the traffic lights changing from green to red .	133
Table 12 Speed correlation between the agents and the drivers	139
Table 13 Correlation between the different training sets.....	140
Table 14 Correlation between the different validation sets	141

Table 15 Agents' Long term behavior	144
Table 16 Comparing GenCL and Knowledge Engineer agents in the training environment	148
Table 17 Comparing GenCL and Knowledge Engineer agents in the validation environment	148
Table 18 Fitness variations for agent A with different GP settings.....	152
Table 19 Fitness variations for agent B with different GP settings.....	152
Table 20 Fitness variations for agent C with different GP settings.....	152
Table 21 Fitness variations for agent D with different GP settings.....	153
Table 22 Fitness variations for agent E with different GP settings	153
Table 23 Traffic light characteristics in the validation data set.....	184

LIST OF ABBREVIATIONS

ACT-R	Adaptive Control of Thought
ANN	Artificial Neural Network
CCTT SAF	Close Combat Tactical Semi Automated Forces
CGF	Computer Generated Forces
COGNET	COGnition as NETwork of Tasks
CxBR	Context-Based Reasoning
DoD	Department of Defense
EA	Evolutionary Algorithm
GA	Genetic Algorithm
GenCL	Genetic Context Learning
GP	Genetic Programming
LLGP	Layered Learning Genetic Programming
ModSAF	Modular Semi Automated Forces
SME	Subject Matter Expert

CHAPTER 1: INTRODUCTION

Simulations have become an essential tool for training individuals and teams for tactical missions. In certain training tasks, it becomes necessary to incorporate virtual agents with human-like behavior to make the simulation realistic. Two types of simulations that often have this need are military training simulations and traffic simulations. The former are used to train soldiers in military tactics and the latter to analyze the traffic capacity of a new road before it is built. The Department of Defense [17] highlights several advantages of using simulators in training. Simulators:

- increase the accessibility to practice sessions,
- reduce the time and effort to produce after-action review material, and
- provide more effective evaluation of new operational plans, doctrines and tactics.

To exhibit realism, these simulations require virtual agents that can act as human players.

Modeling intelligent agents that exhibit human behavior is a complex task. The process involves collecting knowledge about the domain to be modeled from subject matter experts (SMEs), or establishing a more formal mathematical description of the domain. In complex systems, the cost and effort to build realistic agents can be high. This is partly because in the real world, problem

domains often exist where human behavioral knowledge is incomplete, imprecise or even conflicting. Different experts could reach different conclusions when solving the same problem. In certain areas, the modeling becomes even more complex, such as in the modeling of tactical human behavior or skills. Here it is almost impossible to develop a mathematical formalism, and it can be quite difficult to extract the knowledge from SMEs. Often, the models are built on inflexible doctrines. This can cause the entities to behave “too perfectly” with little similarity to human performance [32]. It has also been shown that what is taught by the manuals is not necessarily what is used by the experts themselves. Deutsch [16] gives an example of how Air Force instructor pilots do not scan their instruments themselves as they teach their trainees to do.

Several approaches and techniques have been applied to develop human behavior representations. Most of those models are based on knowledge acquisition from experts through interviews or other manual methods. After the knowledge is collected, the next step is to interpret and analyze the knowledge for the design and implementation of the models. Getting an SME to express his knowledge in a suitable way, and to translate it into representable knowledge are interactive, difficult and time-consuming tasks. This knowledge acquisition problem has been referred to as the *bottleneck* of building expert systems [35]. If the issue is to model human behavior, the knowledge acquisition task is even more complicated because human behavior can be difficult to express and does not follow a particular set of rules. Alternatively, it would also be considerably easier for a SME to physically perform the tasks rather than explain them. The use of a learning system that could automatically extract knowledge and construct a model could reduce the problems mentioned above. An approach to learning and modeling behavior by

observing someone performing a task is called *Learning by Observation*.

This research investigates learning human behavior by observation and applies it to simulated agents. The intent is to use the observations to learn the behavior of the observed entity. To construct a system that can model human behavior by observing a human in action presents several issues. If such a system should be feasible and applied to constructing simulated agents with human behavior patterns, two important prerequisites exist:

1. There must be a suitable modeling structure to represent human behavior in simulated agents.
2. Appropriate machine learning algorithms must be defined.

The objective of this research is to show that the synergistic combination of *Context-Based Reasoning* (CxBR) and *Genetic Programming* (GP) fulfills these prerequisites. Furthermore, these have been integrated in a tool able to learn by observation. CxBR is a modeling paradigm developed especially to implement simulated agents with human behavior. GP is an offspring of Genetic Algorithms (GA) and is a learning algorithm that evolves computer programs (i.e. automatic programming).

This dissertation begins by presenting problem definitions in Chapter 2. Also defined are the hypothesis and the contributions of this research. Chapter 3 discusses the necessary background to modeling human behavior, learning by observation and various machine learning approaches.

This background is needed to fully understand Chapter 4, which describes the new approach to learning by observation by integrating CxBR and GP. The configuration and execution of the experiments are described in Chapter 5, and Chapter 6 contains the results and conclusions from the experiments presented. Finally, Chapter 7 concludes this dissertation with conclusions, summary and future research that could be tangential to this work.

CHAPTER 2: PROBLEM DEFINITION

2.1. Problem Statement

The problem addressed in this research is how to efficiently and accurately create simulated agents exhibiting individual human behavior. Individual behavior means that each simulated agent should exhibit a behavior pattern that is specific to that agent, just as each human individual in the real world would have his or her own behavior pattern. The traditional way of creating simulated agents exhibiting human behavior includes interviews, observations from practice and other means to determine the behavior patterns of the human to be modeled. The knowledge is then analyzed, and the knowledge base is designed. The knowledge extracted is implemented in that knowledge base. Sometimes it becomes very difficult for us to describe our own behavior patterns because they are unknown even to ourselves. It could also be that some of our behavior is not commonly accepted or “by the book”. Therefore, even if we are aware of these behavioral patterns, we might not wish to express them. Furthermore, even if the behavior is well known and accepted, it could be very hard to describe in a way that is manageable to implement in a simulated agent. For example, trying to express how one slows down at a red traffic light could be very difficult to do. The answer would likely include fuzzy comments such as; “speed is rather high”, “visibility good” and “distance close”. Such statements can be difficult

to convert into computer code.

To conclude this discussion, we can state that the general problem is how to more easily develop human-like agents with individual behavior patterns. To further concretize the problem statement, we will investigate how this could be done by implementing learning by observation. In order to implement learning by observation, a suitable machine learning algorithm capable of building human behavior models in a suitable manner needs to be identified.

2.1.1. Criteria for Correct Personalized Behavior

Agents with personalized behavior are not necessarily the best performing agents according to doctrine or to well-accepted rules and regulations. The interest here is to build agents that mimic as well as generalize the behavior of the humans being observed. This means that if the human is breaking the rules and regulations, the agent should also break rules and regulations. What constitutes a good performing agent is that its behavior is very close to the human model, even if the human is not performing optimally or according to common rules.

2.2. Hypothesis

The foundation of the research conducted here and described in this dissertation is based on this hypothesis:

A machine learning strategy can automatically build tactical agent knowledge by observed human performance. Additionally, these agents can approximate and generalize personalized human behavior.

Tactical human knowledge affects the task performed by the human. Human behavior, as it pertains to emotions, psychology or human motor skills, is not of any interest in this research. The objective is to apply the automatically created tactical knowledge within simulated agents such as cars, aircrafts, submarines, troops, etc. Hence, activities inside such agents that are not observable from outside are not interesting. The only interesting aspect of human behavior is the effects of the human behavior - not what caused the effect. Furthermore, the information that forms the basis for learning emanates only from observing human performance. No additional data from doctrines, regulation or common rules are part of the learning process.

2.3. Contributions

This research strives to implement a new model of learning by observation that could contribute to the research community in the area of simulated entities with human behavior. The contributions of this research are:

- Show that individualized human agents can be automatically built by observing human behavior in a simulation.

- The combination of CxBR and GP can, with synergistical advantages, be used as the basis for the system that learns from observation.
- A learning methodology to support the automatic creation of those agents was created and tested using CxBR and GP.
- A tool to accomplish the construction of agents through observation was built and will be made available to other researchers.
- Data was collected to investigate automatic creation of human tactical behavior and is available for other researchers to use.

CHAPTER 3: BACKGROUND

This chapter describes and discusses the background concepts necessary to understand the research conducted in this dissertation. We begin by first discussing the representation of human behavior. CxBR is our modeling paradigm of choice for this research and is described in detail in this chapter. Learning is a key topic here, as learning by observation is the basis of this research. The background and definition of learning by observation is described because it strongly affects the learning paradigm later chosen. Different learning approaches that could implement learning by observation are then discussed. The learning technique particularly relevant to this research is GP. Hence, the last part of this chapter is devoted to this topic.

3.1. Representing Human Behavior

One main objective of this research is to facilitate and enhance the creation of simulated agents that exhibit human behavior. Hence, human behavior representation is an important issue.

The defense modeling and simulation community has defined the term *human behavior representation* to mean models of the human behavior or performance executed in military simulations [56]. These models exist to represent opponents or teammates in a mission

simulation. These are called *Computer Generated Forces* or CGFs. More specifically, in CGFs, the main objective of human behavior representation is to model the tactical intent of the war fighter.

Nevertheless, a wide range of other meanings has been associated with human behavior models. In the humanoid robotics area, mimicking human movement patterns could be the essence of modeling human behavior. The difference between the CGF and humanoid robotic approaches to human behavior lies in the generalization of the model. In modeling human movements, the best movement pattern might be the most general scheme that deviates least from the general population. In the case of CGFs, or in the area of traffic simulation, the human behavior in the different entities should be more personalized and must exhibit a degree of variability.

Henninger [33] defines the term behavior as “any observable action or reaction of a living organism.” The objective of building entities with human behavior is to capture the action, reaction and conscious attributes of the subject of study in the model. The attributes that constitute human behavior could be expressed at many different levels. Banks and Stytz [5] observe two different components. The first component is the correct output modeling where the model produces the right output in a human-like manner. The second component is unpredictability where the model behaves in a manner such that it is hard to predict any preprogrammed behavior pattern. Sidani and Gonzalez [70] had a similar classification, but focused on the implicit and explicit knowledge involved in human behavior. Explicit knowledge is that which could easily be verbalized and represented in symbolic form. Conversely, implicit

knowledge is that which is hard to model and highly intuitive. Whitmore et al. [80] highlight three issues that improve the humanness of an autonomous agent: 1) response time, 2) fatigue level and 3) expertise level. To make the agent act human-like, the response time must be comparable with human response time. Humans also suffer from physical exhaustion and other factors that can lower their capacity and influence their performance - the fatigue level. The expertise level will have influences on the agent's performance. As a human gains experience, his or her performance will improve. Tambe et al. [74] describe in detail the human behavior requirements for building simulated autonomous pilots in a virtual environment. They state the following possible requirements to agents with human behavior:

- Goal-driven behavior
- Knowledge-intensive behavior
- Reactivity
- Real-time performance
- Conforming to human reaction times and limitations
- Overlapping performance of multiple high-level tasks
- Multi-agent coordination
- Communication
- Agent modeling (especially opponent modeling)
- Temporal reasoning
- Planning

- Maintaining episodic memory
- Explanation

Tambe et al. [74] provide a full explanation of the requirements. Depending on the application of interest, the level of humanness of the model can vary greatly. Many applications may only require simple behavior patterns while others may require the agents to be able to communicate with other agents or with real people. In some applications, the ability to explain its actions is also included in the model. As the number of human features increases, the model's level of complexity also dramatically increases.

On the issue of human behavior implementation, one approach would be to build a model of the biological brain. The complexity of such a task is enormous and the connection through which the biological model would reflect intelligence is not fully understood. Nevertheless, good results have been reported in modeling human behavior by simulating simplified neural brain connections, i.e. artificial neural networks. Another promising approach is to model human behavior at a level closer to human problem solving. When humans try to solve a problem, we often decompose them into a number of sub-problems that we can solve one by one, and which together solve the original problem. Humans do not think or solve problems in a mathematical or regression-like manner.

In the area of modeling human behavior, the U.S. army sponsored a substantial amount of research to make more realistic simulators. Three types of simulations can be identified: *live*,

virtual and *constructive*. In live simulations there are no simulated agents present. The simulation is a training session where the actors act in the real world. In constructive simulations, all the actors are simulated agents and the only human interference is the initial configuration or when some limited parameters are adjusted during the simulation. All human behavior events in the simulation come from simulated agents with human behavior. Virtual simulation is when the simulator incorporates both real human players and simulated agents with human behavior. Because this research pertains to simulated agents with human behavior, the interest is in the two latter types of simulations (i.e. virtual and constructive). In constructive simulations, the human behavior models are fairly simple, based on doctrine knowledge and might have probabilistic elements. One such simulation is JANUS [57]. It provides simulation of battling forces. This simulation could be used to evaluate new doctrines or tactics and to train leaders and decision makers.

More challenging are virtual simulations, where both real humans and simulated agents coexist and interact. Here, a well-performing agent should be difficult to distinguish from a human player. The requirements of the human behavior models are much higher and should incorporate more human features. Two commonly used simulations of this type are Modular Semi Automated Forces (ModSAF) and Close Combat Tactical Trainer - Semi Automated Forces (CCTT SAF) [57]. These two simulations can be used to train both individual combatants and decision makers. ModSAF is an open architecture for constructing advanced distributed simulations with CGF support. The paradigm used to model CGFs in ModSAF is finite-state machines. There is no fundamental model of human behavior in ModSAF, so the behavior must

be incorporated into the finite-state machines. This makes using ModSAF somewhat cumbersome to construct a general purpose behavioral or learning model. The human behavior model in CCTT SAF is based on rule-based knowledge. CCTT SAF contains no support for managing human behavior models, and the models are mostly based on doctrine knowledge.

3.1.1. Frameworks for Human Behavioral Modeling

Besides full working simulations, there have been a number of frameworks developed to support the modeling of human behavior. Most of these assume that a human can be described as an input/output system. A way of describing such a system is the modified stage model, shown in Figure 1. This model shown here is derived from Wickens' [81] work.

The modified stage model is a generic model of a simulated agent with human behavior features. The sensing and perception module transforms the external stimuli received from the environment into an internal information representation. Working memory holds the temporary data needed for the cognitive process at its current situation. Long-term memory holds a large amount of data to handle all possible situations in which the agent could operate. The cognition module is the engine that propels the agent's behavior. The cognition process uses the knowledge stored in the memory, handles the situational awareness, scheduling, multitasking, makes the decision and manages the learning process of the agent. The motor behavior simply models the neuromuscular system to carry out the actions selected by the cognitive process. The focus of the research presented in this dissertation is to automatically build, by observation, the

knowledge residing in the cognition module, working memory and long term memory. Note that the learning in the cognition module (see Figure 1) regards the agent's ability to learn from experience. This is not the same sort of learning as *Learning by Observation* used in this research, where the initial tactical human behavior is created by observing a human's performance.

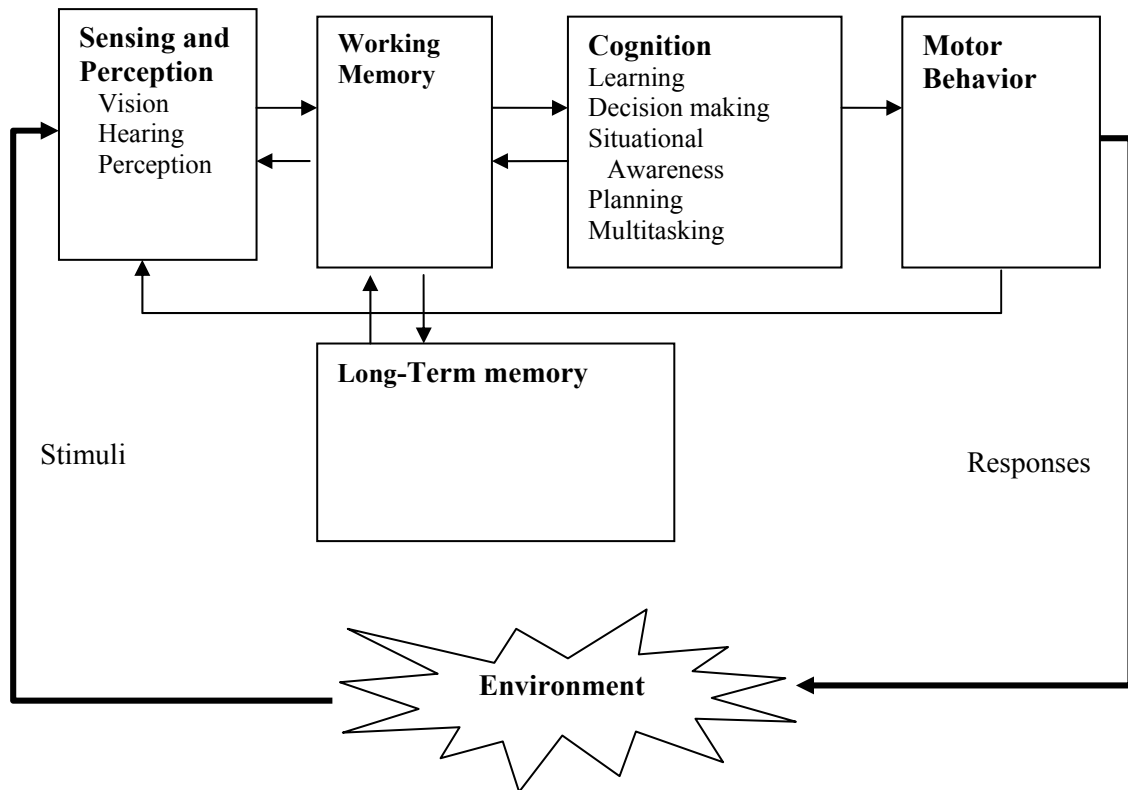


Figure 1: Modified stage model

Some well-explored models of human cognition are Adaptive Control of Thought (ACT-R),

COGnition as NETwork of Tasks (COGNET) and Soar.

ACT-R was originally developed by Anderson [3] as a general model for problem solving and learning. ACT-R categorizes knowledge into two types: declarative and procedural. Declarative knowledge mainly retrieves information from the environment and describes the situation (i.e. situational awareness). Procedural knowledge is represented in production rules. These production rules are goal driven. Between the two knowledge modules, there is a pattern-matching process where the different production rules compete to get executed. There are several learning mechanisms in ACT-R. Learning can take place in the declarative knowledge, the procedural knowledge or in the pattern-matching process between the knowledge modules.

One hypothesis of cognition is that humans perform multiple tasks in parallel. These tasks compete for the human's attention. However, even if these tasks are, as a whole, important for problem solving, the most important tasks are executed first. In a similar manner, COGNET [57] models these parallel tasks as problem-solving agents. Each task (i.e. problem-solving agent) has a set of trigger conditions, and when those are satisfied, it triggers its activation. Activation of the tasks relates to the priority of a task's goal. The basis of COGNET is a blackboard model where communication between tasks takes place and information from the environment is posted. COGNET's trigger evaluation process and attention focus manager monitor the blackboard to manage the task's trigger conditions, activation and execution. COGNET does not have any learning capabilities.

Soar was, as ACT-R, developed as a general problem solving architecture. There are also some learning capabilities within Soar. The basis of Soar was inherited from Newell's model of human cognition [53]. Soar is a goal seeking state machine where a state is the current problem-solving situation. By applying a rule to the current state, it will put the problem solving into a new state and Soar searches for and applies operators until a goal is reached. The Soar architecture is very similar to the modified stage model (see Figure 1), but all the processors, including the motor processors, work through the working memory and not the cognition module as in Figure 1. The working memory contains the current state of the problem solving process and the production rules on how to apply actions resides in long-term memory. Soar has a learning mechanism, called chunking, that is able to create new production rules.

Another architecture developed for modeling human behavior in simulated agents is Context-Based Reasoning (CxBR). This architecture is not as widely used as the other three described here, but requires special attention because it is inspired from a hierarchical model of human behavior based on a contextual approach, making this architecture very intuitive to use for modeling human behavior. Because this is essential to this investigation, it is described in detail in the next section.

3.1.2. Context-Based Reasoning

CxBR has been developed to build simulated agents with human behavior through the use of context-partitioned knowledge [26]. CxBR is based on the concept that humans think in terms of

contexts. A context is an abstraction of a specific situation, where only a limited number of things are expected to occur. Biking in the forest differs from biking in the city, and different types of changes in the environment are likely to occur. This implies that different actions and reactions are expected from the biker in the two different environments. While biking in the forest, you might expect something to obstruct the way around the next bend, but no traffic signs are likely to appear. By attaching the desired behavior of the agent to the contexts, a very suitable structure of hierarchical knowledge is developed. CxBR is based on the idea that:

- A recognized 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 the new situation.
- Things that are likely to happen while under the current situation are limited by the current situation itself.

CxBR encapsulates knowledge about appropriate actions and/or procedures as well as compatible new situations into hierarchically organized *contexts*. See Figure 2. By modeling the agent with different contexts, the scope of the knowledge in each context can be limited and the entire knowledge base becomes well-structured and easier to build and reuse.

The top level of contexts, the Mission Context, describes an overall goal or objective of the mission (e.g. **Drive-Car-Home**). This top level of contexts is the most abstract one. Contexts

further down in the hierarchy become more concrete. A Mission Context does not control the agent per se. Instead, it defines the scope of the mission, its goals, the plan, and the constraints imposed (time, weather, rules of engagement, etc). The Mission Context describes an overall goal that can be the same or different for many agents in a multi-agent environment. On the other hand, the levels from the Major Context down are typically specific for a singular agent.

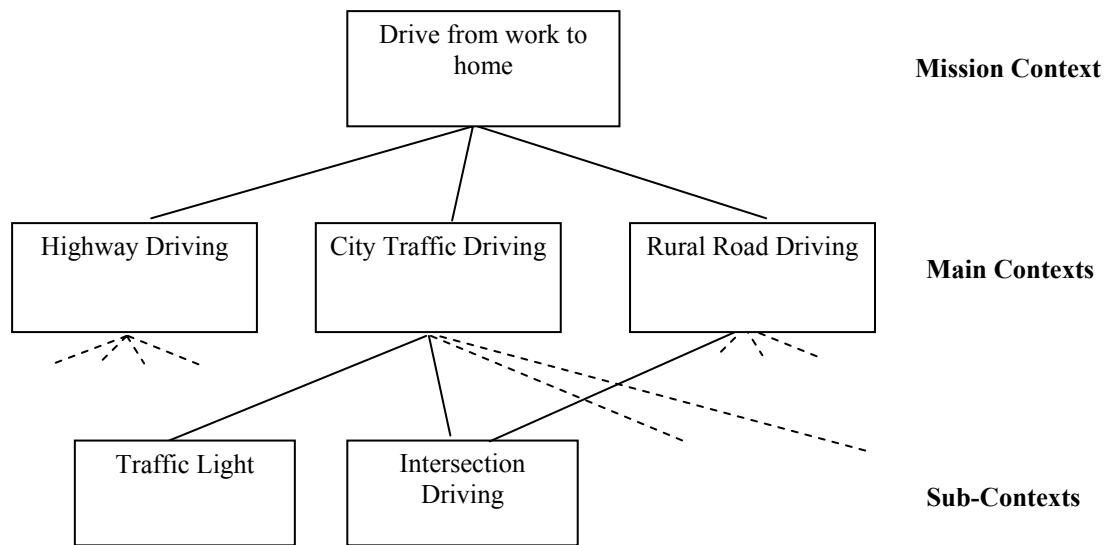


Figure 2: The CxBR hierarchical structure

The Major Context is the primary control element for the agent. It contains functions, rules and a list of compatible next Major Contexts (i.e. possible context transitions). Identification of a new situation can now be simplified because only a limited number of all situations are possible under the currently active context. Sub-Contexts are abstractions of functions performed by the Major Context which may be too complex for one function, or that may be employed by other

Major Contexts. This encourages re-usability. Sub-Contexts can be applicable and reusable in several different Major Contexts. Context transitions are more probable in the lower levels, while the Mission Context seldom or never expires during a training mission. Only one context at the same level can be active at one time (i.e. same level contexts are mutually exclusive).

When the situation changes, a transition to another Major Context may be required to properly address the emerging situation. For example, the automobile may enter an interstate highway, requiring a transition to an **Interstate-Driving** Major Context. Transitions between contexts are triggered by events in the environment – some planned others unplanned. CxBR is an intuitive, efficient and effective representation technique for human behavior representation. For one, CxBR was specifically designed to model tactical human behavior. As such, it provides the important hierarchical organization of contexts.

Two major parts can be identified in the context-base that together constitutes the behavior of the agent. All the knowledge in the context base is stored in the *action rules* and the *sentinel rules*. Within the different contexts and Sub-Contexts, the action rules control the behavior of the agent in a specific context. The other part is the set of sentinel rules that determine when a new context should become active (i.e. context transition). Observe that the expression *rules* do not limit the knowledge to be stored in IF-THEN rules. In fact, the action rules and sentinel rules could be composed of production rules, various functions and operators and Sub-Context calls, or other more complex data and code structures. Even if the name of the behavioral knowledge containers are action rules and sentinel rules, these names now refer to the collection of rules, functions,

operators, Sub-Context calls or data structures that might constitute its behavior.

When a context transition takes place, the collection of sentinel rules determine that the currently active context is no longer the most suitable one for the situation at hand, and will activate another context. Sentinel rules can be implemented in two different ways: direct transition or competing contexts transition. Within direct transition, each context is self-aware and its sentinel rules are fully contained within the context. If a context at a specific level in the hierarchy requests activation, either one of two prerequisites must be fulfilled for this context to be activated. Either no other context is active at the same level or the already activated context needs to release its activation. If two contexts request activation at the same time, the activation will go to the context first in line (i.e. contexts at the same level will be prioritized at the implementation stage). Contexts will always yield their activation to a requesting Sub-Context, if and only if the Sub-Context is a Sub-Context to the context currently active. In competing context transition [65], the context at the same level competes for activation. When a new situation occurs, each context evaluates the situation and comes up with some score on how well the context addresses the current situation. The context with the highest score will then be activated. So far, the competing context has not been thoroughly investigated or implemented, but it will probably be part of CxBR implementation in the future.

3.1.2.1. The CxBR Framework

A framework was developed by Norlander [55] to facilitate the execution of simulated agents

within a simulator. The framework provides tools for building a basic simulation engine with agents implemented in a CxBR structure. It provides the developer with structures and tools to connect the different contexts together. It also provides the functionality on how the context switching takes place and how the different agents' sentinel rules are executed. Figure 3 describes the parts of the framework.

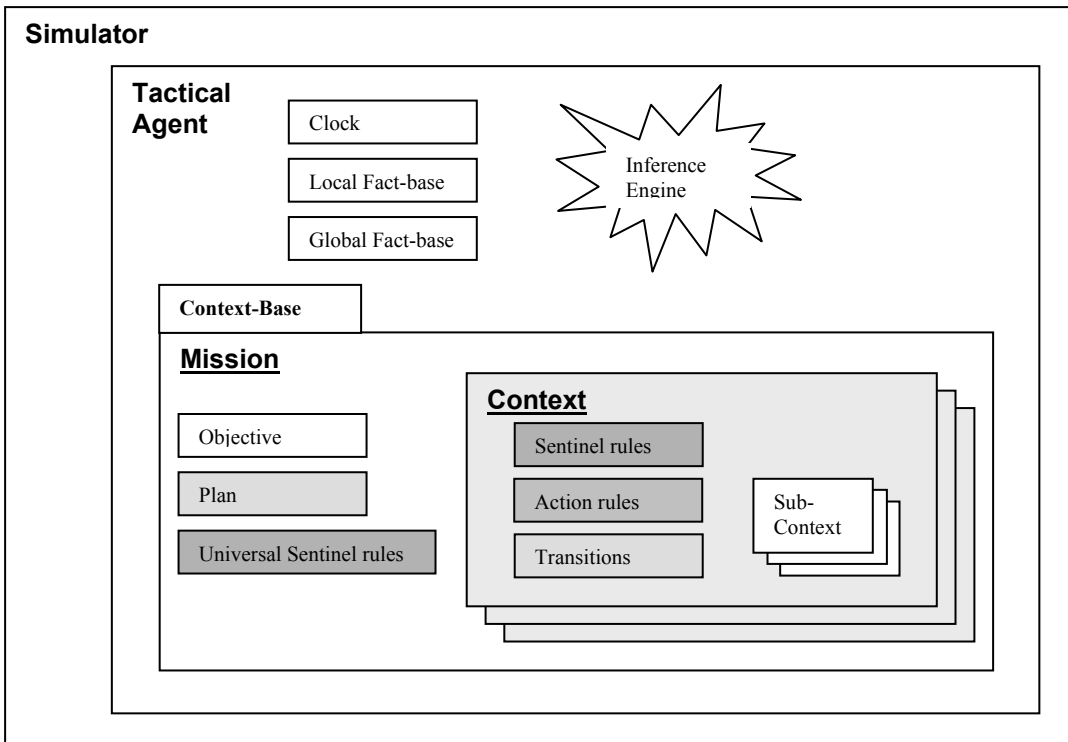


Figure 3: Context-Based Reasoning in simulation

The framework is implemented in C++ and gives the developer a set of base classes to easily derive and create agents within the CxBR paradigm. With the framework, the developer can concentrate on implementing the knowledge in the action rules and sentinel rules of the different

agents within the simulation. Direct transition context switching is implemented in the framework, but it does not yet support competing context transitions, although there are no restrictions within the framework prohibiting its extension into competing contexts.

All the agent's contexts in the hierarchy (i.e. mission Contexts, Major Contexts and Sub-Contexts) are stored in a context-base. The mission context contains criteria (i.e. objectives) that determine when the mission has been completed. The mission also contains a plan that strives to complete the mission objective. Each context also has a transition list, which specifies to what other contexts this context could transition.

This framework provides an inference engine as well as tools to create and manage the local and the global fact base. The local fact base in the framework stores the facts that are only to be known by the particular tactical agent (i.e. private knowledge), while the global fact base contains facts that are available to all agents. Hence, the global fact base describes the simulated world (i.e. the environment). The framework provides a tool for searching and retrieving facts from the local and global fact base and makes them available to the agent. By updating the fact bases, the knowledge stored in the context base can make the agent interact with the environment. Each agent in the simulation is stored in a list and in every simulation step all the agents in the list act. When an agent is acting in the environment, the local and global fact bases are updated so the agent and all other actors in the simulation can be aware of this agent's prior actions. The agents in the list are all things in the simulation that could change state (i.e. traffic lights and human behavior agents). As an agent fulfills its mission goals, it will be removed from

the list and its existence in the simulated environment cease.

The framework was further developed as part of the research described in this dissertation by adding a simple graphical interface. This graphical interface shows the traffic simulation with the streets, cars and traffic lights. It was developed with the OpenGL standard library. The framework was further enhanced with recording capabilities that could trigger to capture the behavior of the simulated cars at specific instances. The recording feature is also able to include information in the recording from a reference car located in a separate database. This feature makes it easy to compare different agent's and human's performance.

3.2. Learning by Observation

Automatic model construction would certainly improve the development process for human behavior models. If the models could be created by merely automatically observing the human being modeled, it would dramatically ease the model development process. This would fall under *Learning by Observation*.

The term *Learning by Observation* has its roots in biology. Infants of many species often learn things by observing the 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 the task. Inspired by how humans and other mammals seem to learn by observation, the machine learning community has developed several theories on

learning by observation, applied to different areas. Most of the AI literature refers to learning by observation as a method of learning the behavior of another agent or human by observing its action. Note that learning by observation is a direction on how the data for learning is to be collected - through observation. It does not mention what learning paradigms to use or what type of learning takes place. In this research, the definition of learning by observation is as follows:

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

A number of advantages could be gained by using learning by observation instead of the traditional knowledge acquisition and development methods:

- Reduce time and cost of development, debugging and maintenance
- More accurate, realistic and refined representation of human behavior
- Potential to incorporate new features of human-like features, such as emotions
- Relaxes the need for programming skills in the part of the operators
- Reduction of problem domains
- Develop simulated entities in real time
- Ability to specifically model variations of the behavior (e.g., aggressive drivers)

Several researchers have proposed learning by observation as a mean to overcome the knowledge acquisition bottleneck [29], [43], [68]. The use of a system that interacts directly with

the SME as he or she performs the task and automatically extracts the knowledge (i.e. learns by observation), would significantly reduce the time and effort in knowledge acquisition. If the process could be fully automated, learning by observation could conceivably be able to produce or update simulated entities in real time. As an example, while an expert uses a simulator, an automated learning by observation system could, at the same time, be developing a model that emulates his performance. It is much easier for an expert to perform a task than to describe its performance, or by other means try to evaluate someone's performance, as in learning by instruction [44]. The use of learning by observation has additional advantages as it pertains to model behaviors where the knowledge is difficult to express.

The time and cost of correcting, updating and customizing could also conceivably be reduced if learning by observation would be used in lieu of restructuring, adding or removing part of the model by hand. Milzner and Leifhelm [51] propose learning by observation to relax the knowledge update problem in rapidly changing knowledge domains.

3.2.1. Prior Work in the Field of Learning by Observation

Some researchers have stated that certain knowledge could be very difficult to extract with traditional methods [28], [70]. Knowledge that is hard to model and highly intuitive is classified as implicit knowledge. Implicit knowledge is easier to extract by using learning by observation. In fact, it might not even be possible to formalize this knowledge with traditional methods.

Henninger et al. [32] show that by using learning by observation, the models generated are more accurate and comply more fully with human performance. The use of learning by observation will also open the possibility of easily developing many simulated entities with similar, but not identical, behaviors. Each entity could be tuned slightly differently to personalize its behavior.

Schaal [68] states that in an enormously large search space, one approach is to use learning by observing and imitating the behavior to reduce the search space and make it usable. Restricting the learning algorithm to minimize the deviation between the observed entity and the learning agent dramatically reduces the search space. In some applications, learning by observation could be used to reduce or even diminish the need for time-consuming and complex programming. If the agent at hand can learn tasks automatically by observing the task performed by others, the need to program the new behavior by hand is no longer necessary.

The time and cost of correcting, updating and customizing could be reduced if learning by observation were to be used instead of restructuring, adding or removing part of the model by hand. Milzner and Leifhelm [51] propose learning by observation to relax the knowledge update problem in rapidly changing knowledge domains.

In artificial neural networks, the term learning by observation is often used to refer to the fact that the training data is a set of observations. This is not to what this dissertation refers to as learning by observation. Much of the data in the machine learning community is based on real observations, but do not include any behavior knowledge or demonstration on how to perform a

task. Even if many observations are used to learn (for example, recognizing handwritten characters), the observed entity might not teach us any behavioral skills. The intent of this research is not only to use the observations to learn, but also to learn the behavior of the observed entity. Thus, interest herein is to look at learning by observation with respect to gathering knowledge by observing a human in action in order to model his behavior.

In the area of robotics, the use of learning by observation has been previously used to implement human behavior in humanoid robot movements [4], [67], [73]. Many times, the core issue of learning by observation within the robotic area is dealing with image processing, as humanoid robots often use cameras to implement their vision. The learning system often tries to mimic the specific movement of the human, interpreted by the robotic vision system, and not to adopt a general behavior. This refers to the sensing and perception module in the modified stage model (see Figure 1) and is not the objective of the research presented in this dissertation. Schaal [68] makes a distinction between learning by observation and imitation learning. In most humanoid robotics, the objective of the movement pattern is to imitate the human as closely as possible.

Bentivegna and Atkeson [8] used learning by observation to implement behavior skill in a humanoid robot. By observing a human, it learned to play air hockey using a set of action primitives, each describing a certain behavior (e.g. left hit). Prior work in modeling human behavior through learning from observation has also been conducted in the area of maneuvering a car [59] and flying an aircraft [44] [66]. However, the work in modeling human behavior has been done to create the best performing agent or to model low level motor skills. No results have

been found where the focus is on personalized behavior patterns and/or tactical decision making.

Moukas and Hayes [52] used learning by observation to model social behavior in autonomous robots. The social behavior they modeled was the behavior of honeybees. The study of interest was that bees communicate with each other using dances that tell where food has been found.

Moukas and Hayes' reinforcement scheme showed the potential of learning by observation. The social behavior to learn even the things needed to teach others must be classified as a very hard problem that was still solved through observation alone.

The learning algorithm must be able to collect the data from the environment and monitor the actions of the expert. A feasible way of collecting data and probing the action of the user is to use a simulator to implement learning by observation, as in the work of Gonzalez et al [28]. By using a simulator instead of the real world, data collection will likely be much easier, and some situations that are difficult or dangerous (e.g. hazardous situations) could emerge. In the simulator environment, there is no need for complex sensors or image recognition systems to be able to understand the environment. Gonzalez et al [29] argue that learning through observation is especially well suited to acquiring tactical knowledge, the knowledge used to apply the best action for a given situation. Tactical knowledge is often implicit knowledge. Hence, tactical knowledge can be very hard to express and extract from an expert by traditional knowledge acquisition methods. Several different learning strategies have been used to implement learning by observation.

3.3. Different Machine Learning Approaches

In order to achieve learning by observation, a machine learning algorithm needs to be incorporated into the process.

Machine learning algorithms can be applied in two different ways: on-line and off-line. In off-line learning, the data are collected and possibly preprocessed prior to learning. On-line learning is performed while data are being collected. Machine learning can also be classified in three different classes:

- *Supervised learning.* During supervised learning, the correct output is known and the learning algorithm can be supervised. For each input pattern, the corresponding output is distinct and using the correct output the learning can be directed towards its goal.
- *Reinforcement learning.* In reinforcement learning, the correct output is not known but the results of action taken can be evaluated by the learning algorithm. During learning, actions or decisions that result in positive outcomes are reinforced, while those that result in negative outcomes are weakened.
- *Unsupervised learning.* During unsupervised learning, the correct output is also unknown and can't be evaluated as in reinforcement learning. Unsupervised learning algorithms self-organize in some manner to compress or cluster the data.

Since this research aims to model human behavior and the human's performance is known,

unsupervised learning is not of any interest. Reinforcement learning might be an interesting approach. This will be described in more detail next. The rest of this chapter briefly presents some supervised machine learning approaches of interest.

3.3.1. Reinforcement Learning

Sutton and Barto [72] make the following statement on Reinforcement Learning: “The learner is not told which actions to take but instead must discover which actions yield the most reward by trying them.” This learning by trial-and-error is a central issue in Reinforcement Learning. The correct output is not known, but the agent’s actions can be evaluated.

They further state that reinforcement learning is different from supervised learning. In reinforcement learning the output pattern is not known, but there are known methods to evaluate the output. The performance could be measured based on the evaluation of the output. The output can’t be classified as correct or faulty, but it can distinguish if one output is better than another. The most important feature of reinforcement learning is the evaluation of the action taken as the exploration force.

Sutton and Barto [72] identify four sub-elements of reinforcement learning: 1) a policy, 2) a reward function, 3) a value function and 4) a model (optional). The policy defines the behavior at a given time. The reward function rewards the learning algorithm for some action based on the desirability of the current policy. If a policy receives a bad reward, the learning algorithm lowers

the chance of selecting the same policy the next time the situation occurs. The value function estimates the long-time desirability of different state changes. The reward function gives an immediate response from the environment on how good the policy is, but the value function estimates the cumulative reward a certain policy could achieve taking in to account the states that are likely to follow. The reinforcement learning cycle could be described as follows:

1. Interpret the situation regarding inputs, internal state, etc.
2. Choose the most promising action.
3. Evaluate the new situation.
4. Possibly predict the future reward of this action.
5. Give the last action appropriate reward regarding both steps 3 and 4.
6. Adjust/update the part that determines the action.

From this, we can conclude that Reinforcement Learning can be described as a goal-driven agent that learns from experience. The outcome of an action could not be determined to be correct or faulty, but it could be evaluated to be more or less appropriate. Contrary to unsupervised learning, where nothing about the output is known, we can distinguish good actions from bad ones. Furthermore, Reinforcement Learning is almost exclusively on-line learning. To be able to evaluate an action (i.e. experience) the action must be performed. Since the outcome or result of an action might not be deterministic, the action must be conducted or simulated. One might argue that learning can take place off-line if the outcome of each action is estimated and then evaluated.

3.3.2. Inductive Learning

Inductive learning is the process of learning from examples. Hence, inductive learning is most suitable when implemented as an off-line and supervised learning approach. The learning is based on the theory that there exists a hypothesis h that can approximate the function $f(x)$ that transforms the input x to the output y , where $y=f(x)$. In other words, inductive learning tries to approximate the input-output transformation by the examples (x, y) presented to it. Inductive learning is, therefore, more of a collection of learning paradigms that learns by mapping the correct inputs to the correct outputs. The type of learning that falls under inductive learning is mostly related to pattern recognition or classification. To implement inductive learning, various statistical and machine learning algorithms can be used. Several evolutionary and artificial neural network approaches could be considered as inductive learning, but they will be presented in separate sections, since they can be used in a wider area than merely as inductive learners.

Statistical methods, such as nearest neighbor [15], have been used in the area of inductive learning. Nearest neighbor techniques use a distance measure to determine the most similar example stored when a new pattern is presented. The classification is then determined by the class belonging of the closest example found. A far more interesting technique, and commonly correlated to induction learning, is the use of decision trees.

3.3.2.1. Decision Trees

Decision trees are, as the name suggests, trees built of internal decision nodes. Figure 4 shows an

example of a decision tree. Executing the tree is done by traversing the tree and choosing the various paths to eventually reach a *leaf* in the tree. Leafs describe the solutions (i.e. the classes in classifying problems). The paths describe rules used to reach the solutions.

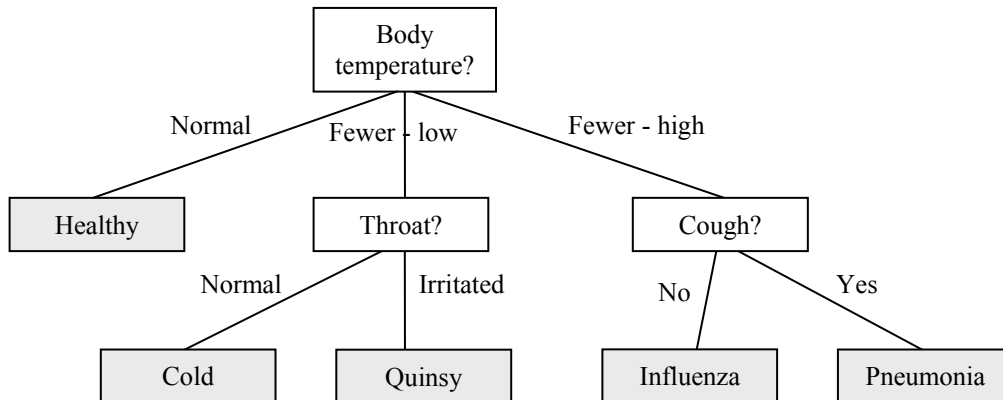


Figure 4: Decision tree example

The problem with a decision tree arises when we need to cover all possible answers to all decision functions. If we look at the simple example in Figure 4, we see that we have no solution to the facts of low-fever and cough. If we want to construct a complete tree with only two different classes but with n attributes (i.e. different facts), the number of functions in the tree will be 2^{2^n} [71]. If we only have six attributes, it means that we will have $2 \cdot 10^{19}$ different functions in the tree. That is a large tree.

The learning strategies applied to these decision trees is to prune the trees to a usable size. There are a number of such learning algorithms as ID3 [62], C4.5 [63] and CART [13]. The basic idea

in finding a small, usable tree is to find the most important decision functions by looking at the training examples. The more the function affects the results, the earlier it should be tested in the decision process. If the tree is described as in Figure 4 the important functions are moved up in the tree, so the most important function is placed at the top and tested first. The more influence a function has on the result the more important the function is. If the answer to the problem can be found by only processing one function/question, this should reside at the top of the tree. In this manner, the size of the trees can be reduced and the average search steps minimized.

When it comes to learning human behavior, decision trees are feasible since the knowledge stored is verbal and similar to human ways of reasoning.

3.3.3. Connectionist Learning

Connectionist learning is a group of learning algorithms whose theories are inspired by the smallest mechanisms that explain intellectual abilities in the human brain- the *neuron*. These algorithms are often referred to as Artificial Neural Networks (ANN). Simplified models of the electro-chemical mechanism in the brain are the commonalities in this group of algorithms.

In 1943, McCulloch and Pitts [48] presented the first simplified model of a human neuron.

Rosenblatt [64] added a learning mechanism and called it *perceptron*. In the late 60s, the single layer network proved to have major limitations and the research was set back for a long time. At that time, computational power was very limited and no learning algorithm had been developed

for multi-layer networks. In the late 80s, the research accelerated again when the computational power improved, and the multi-layered network learning algorithms had been developed and refined. The use of neurons in a multilayer network resolved the limitations of single-layer networks.

3.3.3.1. Artificial Neural Networks

Common in all ANNs is that they are composed of a large number of artificial neurons, which are connected to each other in a network structure. Each connection is equipped with a weight that determines how important the connection is. As signals traverse the net, each single neuron asynchronously processes its local information. Learning in the network takes place as the strength of the connections (i.e. weights) are recalculated and iteratively updated during the training process. The neuron that is common to all ANN has very small variations in different ANNs. Figure 5 show a simple neuron model.

The inputs, x_n , could be binary or real-valued numbers that are multiplied by their corresponding weights, w_{nj} , and those values are then summed in neuron j . This value is then applied to an activation function which “fires” the neuron if the value reaches above a threshold value defined by the activation function. This firing produces an output value, y_j . The activation function is typically a sigmoid or step function. The output of the neuron could be connected to other neurons in the next layer and serve as their input values. If the neuron is in the last layer, then its output contributes to the final output value. Most ANNs require the input values to be

standardized to some range (if they are real valued numbers). When learning takes place, the weight values change. When learning is over, the knowledge stored in an ANN is the different weight values in the network.

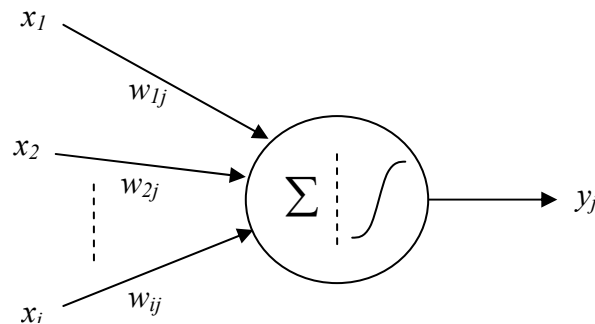


Figure 5: The simple neuron model

The neuron structure, and how they are connected, could be very different in different ANN architectures. Figure 6 shows three different network structures with different neuron connections. The number of neurons is different from case to case and is mostly defined by the ANN architecture. There are some networks that are able to self-adjust the number of neurons needed to solve the problem (e.g. ART architectures). Even if the network topologies look very different, the simple functionality of the neurons and the connections with the weights is very similar. The learning algorithm can also vary between different ANNs. Even if the network topology is the same, the learning (i.e. updating the weights) could differ significantly. If the network is a multi-layered feed forward network, the learning could be back propagation, cascade correlation, genetic algorithms or some other learning algorithm. The types of networks

can also be classified as supervised or unsupervised (see section 3.3) networks. In unsupervised learning ANNs, the correct answer is not known but the network organizes itself in some manner by the examples presented to it. In this manner, the problem space can be organized and modeled automatically in the way the net finds most appropriate.

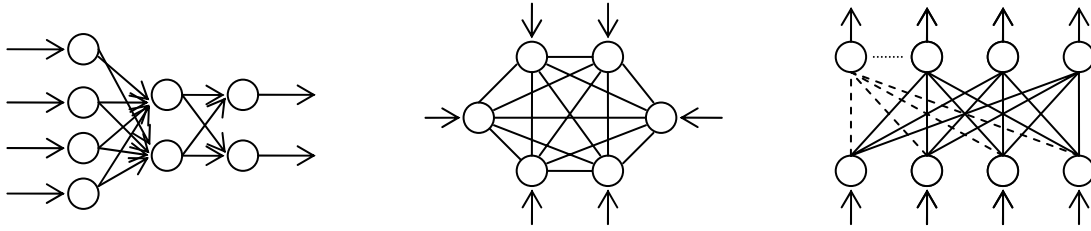


Figure 6: Neuron connections in Multi Layered Perceptron, Hopfield Net and ART 1

3.3.4. Evolutionary Learning

Another very interesting machine learning algorithm is Genetic Programming (GP). GP is part of the *Evolutionary Algorithm* (EA) branch of machine learning algorithms. GP is an offspring of GA, and has the same foundation but differ in some significant areas.

Before getting into GP, it is worthwhile to look at the EA family. Figure 7 illustrates the different branches of the EAs. Evolutionary Strategies and Evolutionary Programming both started to develop during the early 1960's. Both use real valued individuals and the evolutionary operators are based on statistical distributions. Evolutionary Programming looks at evolution at the species level and thus has no recombination operators available. Holland [34] presented Genetic

Algorithms (GAs) in the mid 1970's, and that has become the most popular EA. Classifier Systems are an extension to GAs that Holland presented in the mid-80's. This extension introduces a black box (i.e. autonomous agent) where the GA is the learning engine that receives reinforcement from the environment. Classifier Systems can be seen as an implementation of reinforcement learning. Genetic Programming was developed from GAs and Koza's groundbreaking work [40] in the early 1990's.

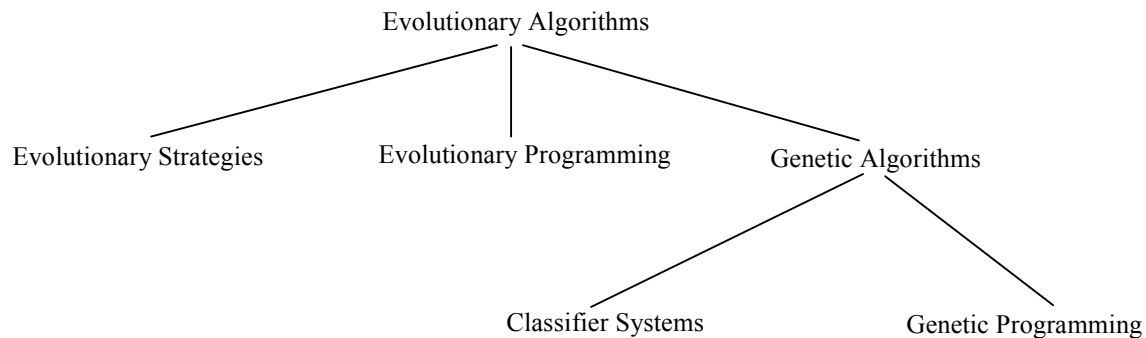


Figure 7: Evolutionary Algorithms

The difference between GAs and GPs is that GPs evolve computer programs. Each individual in a basic GA usually represents a set of values, which in turn represents some solution to a problem. Otherwise, the two basic GA/GP algorithms are very similar. A complete description of the GP algorithm is described in the next section. In fact, the step-by-step procedure described on page 44 is the same for GAs.

The individual's representation within the GA or GP is often expressed as the genotype. The

phenotype is the result when the genotype is transformed and placed into context and will represent the complete solution. In GPs, the genotypes are computer programs usually represented in source code or machine code statements. The transformation from genotype to phenotype, in this case, is done by interpretation or compilation and execution of the program. The use of computer programs as individuals raises the problem solving to a higher abstraction level and opens the possibilities of more easily attacking complex problems from a high-level problem statement. As the GP evolves program (i.e. automatic programming), it states that all problems that could be solved by a computer could also theoretically be automatically created by the GP if sufficient computational power and adequate time is available.

There is no claim that problems solved with GP could not be solved with a GA, but the use of GP sometimes provides a better tool than a GA. Still, many problems could be easily solved with a GA, such as classification and optimization problems with numerical values. When the problem's complexity increases and the objective is to develop complex structures, such as building an amplifier, the use of a GA adds extra customization to arrive at a useful representation scheme - adjust/invent genetic operators to work on the representation, etc. GPs have all those tools, and by representing the individuals as computer programs, more complex and intertwined problems can be tackled.

Genetic Algorithms / Genetic Programming

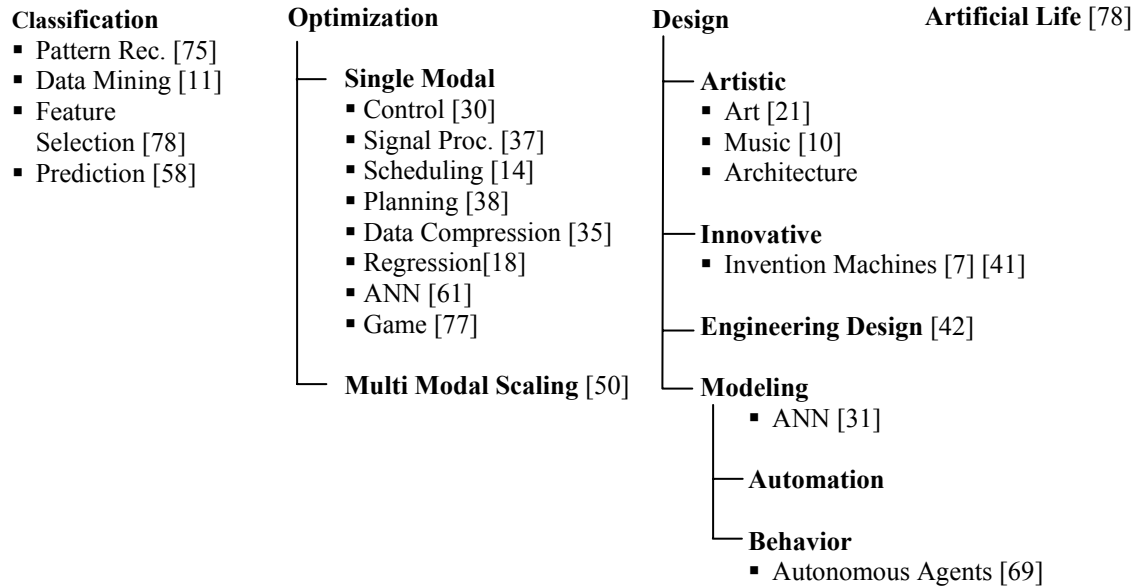


Figure 8: Taxonomy of GA/GP applications

GAs and GP have been used in a vast variety of problem-solving domains. In Figure 8, the taxonomy of GA/GP applications is structured in four main categories: 1) classification, 2) optimization, 3) design and 4) artificial life. The classification category is rather self-explanatory. Optimization, however, is more difficult to define. The argument could be made that all other categories are sub-elements of optimization problems. For example, when the problem is to classify handwritten characters, it could be viewed as an optimization problem where the number of misclassified letters is minimized. Instead, the optimization problems here are described as the existence of a structure, function or formula to be optimized. Multi-modal optimization is when the interest is not only to find the global optima, but also to find local

optimums under investigation.

The third category is design. Here, the problem is one of a creative nature. The task is to create and build something that might have an artistic flavor, where the genetic process creates artistic pictures or learns to play jazz solos [10]. Here, the learning is no longer supervised in the traditional way. It is hard to tell if the output is correct or not. Rather, someone's opinion is often used as the fitness evaluation in an interactive manner.

A sub-category of design is innovative systems. Here, the objective is to create a machine that will invent new things. GP has been successful in presenting solutions that are equal or better than patented results [7], [41]. In engineering design, the genetic process replaces the engineer in designing for example antennas, amplifiers or filters [42]. Here, the GP starts with a blank piece of paper, a set of valid components, and a tool to evaluate the performance of the individual (e.g. a filter simulator). Now the GP chooses the components, does the wiring, tests the individuals and evolves (i.e. designs) the filters from scratch in a manner similar to what an electrical engineer does.

The more traditional machine learning applications (e.g. robot navigation, learning behavior, automation process, etc) are located under modeling.

The last category is artificial life, an area of application that does not fall under the other categories. Here, the EAs have their most natural application area. Artificial life forms are

created within an artificial environment, and the EAs simulate the evolution of the artificial species.

From the discussion above, we can see that GP and GAs have been used in a variety of different machine learning applications. Because GP seems to have a better toolset that applies to many different applications, compared to GAs, we will next describe the the GP algorithm in detail.

3.3.5. Genetic Programming

The main characteristic of GP, and its major difference with GAs, is that each individual in the population is a computer program or something that can be interpreted in a syntactical context. The target system for GP could be a CPU, a compiler, a simulation or anything else that can execute the pre-defined instructions, from now on referred to as a *program*.

GP is a directed stochastic search process that looks for the most suitable program that will solve the problem at hand. The search is an iterative process described in Figure 9. The search process searches for the best individual in a set of individuals (i.e. the population). The individuals in GP are a set of programs that represent different solutions to the problem.

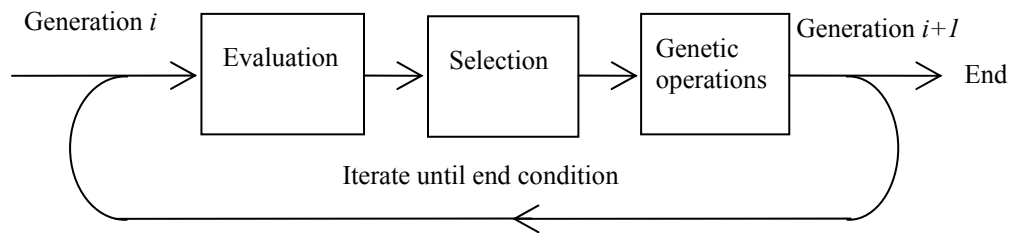


Figure 9: The iterative GA/GP search process

The different individuals could be regarded as different search points in the problem space. Hence, the search process is a parallel process that investigates several possible solutions at the same time. Furthermore, all the individuals need to be evaluated in some manner as to what degree they are able to solve the problem. A *fitness function* is used to do this. The features of the individuals with better suitability would preferably be preserved through a selection process and carry on to breed the next generation of individuals. The selection process directs the search process to choose better performing individuals. The genetic operators (e.g. crossover and mutation) are stochastically applied and will explore new areas of the problem space and hence, support the development and evolution of the individuals.

Evolving a program with GP can be described in five steps:

1. Create an initial population (usually randomly generated).
2. Evaluate the performance of each individual through a fitness function.
3. Based on the evaluation, decide which individuals will survive, reproduce or be killed.

4. Apply genetic operations (e.g. mutation or crossover) to the individuals selected for reproduction.
5. If the criterion of stopping the process is not met, restart at step 2.

The population might number in the hundreds or thousands, but eventually only one - the best of the individuals - will represent the solution. In some applications (e.g. multi modal optimization problems), several individuals might represent the complete solution, but it is most commonly used to select only the very best individual.

The criteria for stopping the evolutionary process can be when 1) a maximum number of evaluations are made, 2) a maximum number of generations are evolved, 3) the fitness reaches a certain level, or 4) other measurable criteria are given. The GP will produce a program that will solve a predefined problem in almost any area when the genetic process is finished (see Figure 8).

3.3.5.1. GP Selection

In each generation, every individual in the population is evaluated and its fitness value calculated. The fitness value affects the selection process so that the more fit individuals are more likely to be selected. Parents will repeatedly be selected during the generation shift until a new population has been created with the same size as the old population. When a parent is selected from the original population, it is not removed from the old population. A parent can be

selected to procreate many times during a generation shift. First, when the new population is complete, the old one is discharged. Many different selection methods can be used in GP. Some of the common methods are Fitness Proportionate Selection, Ranked Selection, Tournament Selection and Fitness Uniform Selection.

In Fitness Proportionate Selection, the individual's probability of being selected is proportional to the individual's fitness value. First, the fitness value is normalized so that the sum of all individual's fitness values in the population is equal to one. Then the probability p_{s_i} is calculated as:

$$p_{s_i} = \frac{F_{\max} - f(s_i)}{\sum_{j=1}^N f(s_j)}$$

Here, the function $f(s_x)$ is the normalized fitness value for individual x and N is the number of individuals in the population. Note that the formula calculates such a probability when a lower fitness value is considered better, as it is in this research, and F_{\max} is the highest normalized fitness in the population.

Ranked selection first ranks the individuals according to their fitness. The probability is then calculated according to their rank and not their fitness value. The sum of the probability for all individuals is normalized to one. The population needs to be sorted prior to selection. The

probability of an individual in ranked selection (i.e. linear ranked selection) is then calculated as

$$P_{s_i} = \frac{R_i}{\sum_{j=1}^N j}$$

where R_i is the individual's ranked position in the population (the worst individual has position 1) and N is the number of individuals.

In Tournament Selection, a number of individuals are randomly selected from the population. The individual in this group with the best fitness will be selected as a parent in the next breeding session.

In Fitness Uniform Selection, a random value is picked in the range between the lowest fitness and the highest fitness in the population. The individual with the fitness value closest to this random value is then selected as the parent. In Figure 10, the individuals are described as circles on the fitness range between the individual with the lowest fitness and the one with the highest fitness. The bold individual is then selected in the example because it is closest to the randomized value in the fitness range.

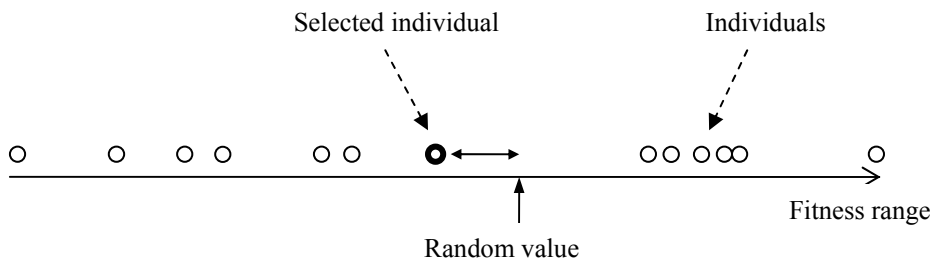


Figure 10: Fitness Uniform Selection

3.3.5.2. Representation of the Individuals and Genetic Operators

The individuals in GP could be described as an instruction tree [40]. Figure 11 shows an example of one individual. Describing source code as an instruction tree helps to understand the impact of the genetic operators, described later. The tree structure is also one way of implementing the GP algorithm.

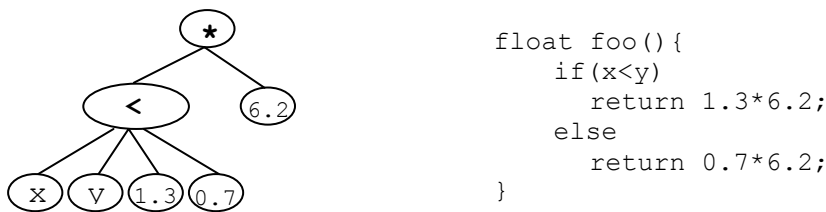


Figure 11: A possible GP individual's instruction tree and its corresponding C-code

The instruction tree consists of a function set and a terminal set. Each node in the instruction tree that expands the tree with new branches is part of the function set. Each leaf in the instruction tree

is part of the terminal set. Hence, the terminal set could be either constant values or variables. The function set is the set of operators/functions that are able to process the data. A function expands the tree one level and points to at least one other function or terminal. The terminal puts an end to a branch. All the leaves of the tree will be a terminal (i.e. a variable or a constant). Both the terminal set and the function set are problem-dependent and must be specified before the evolution process starts.

Other implementations of the individuals, such as linear code representation, have also been used. The linear code representation is common when the individuals are represented in machine code [6]. Figure 12 shows two examples of GP individuals with linear representation. The top one describes an executable machine code individual while the lower one is an assembler language individual. To more easily understand the impact and results of genetic operators, the discussion continues with the tree structured individuals (also the representation chosen in this research).

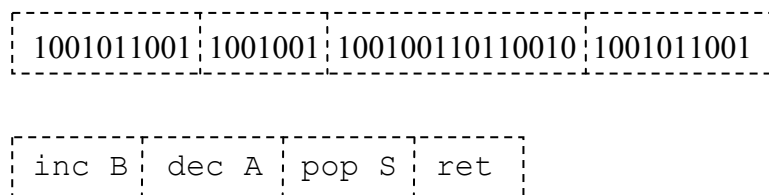


Figure 12: Two examples of linear representations of GP individuals

The genetic operators will combine the parent's building blocks and/or slightly modify them to

create new individuals. When genetic operators are applied, the shape and structure of the trees are changed and altered with the influence of the parents' trees. The most commonly used operators are crossover and mutation. Crossover simply alters two branches between the two parents to create new individuals (see Figure 13).

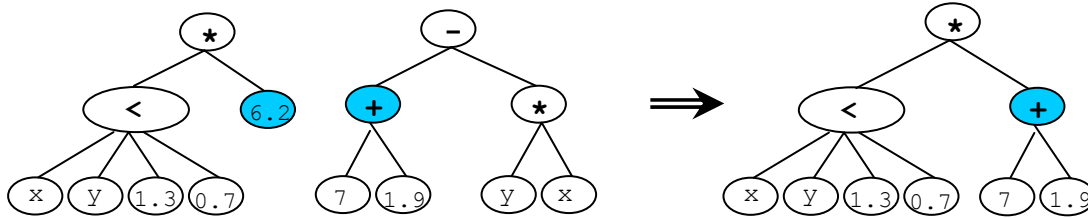


Figure 13: Crossover creating one of the two possible offspring

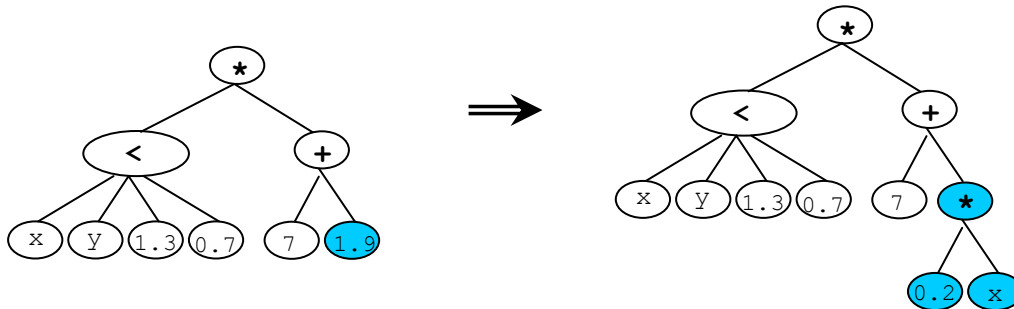


Figure 14: Mutation example

Mutation affects a single individual and changes a point of the tree, as shown in Figure 14 . The point can be either a function or a terminal. The mutation operator removes everything beneath the point and inserts a new randomly created sub-tree at this point. The location in the tree where a genetic operator is applied is randomly chosen, when an individual has been chosen to

crossover or mutate.

The alternative type of mutation is here called *node mutation*. Instead of deleting the whole sub-tree beneath the selected node and a new sub-tree randomly created, in node mutation the selected node is not deleted but the contents of the node are changed. If the node is a variable, an operator or a function, it is randomly changed to one of the other possible variables, an operator or function of the same type. As an example, if the selected node contained a *sine* function, it could be replaced with a *log* function. If the node contains a constant, this constant value is replaced with another value randomly generated within the valid range.

When an individual has been selected by the selection mechanism, it will be exposed to the probability to be modified by the genetic operators. Hence, the operators described here, crossover and mutation, will be applied to the selected individual with certain probabilities: *crossover rate* and *mutation rate*. These rates are features of GP that need to be set prior to learning. For each selected individual, a random number is generated in the range 0 to 1. If this number is less than the operator's rate, the operator will be applied to create a new individual. If none of the operators are applied to the selected individual, it will be copied unchanged to the next generation of individuals (i.e. cloning). Since an individual could be selected many times during a generation shift, it is also possible that one individual will be cloned several times into the new population (i.e. next generation).

3.3.5.3. Problem Space and Search Space

To further explore the nature of GP learning, the search problem within GP will be more thoroughly discussed. The search space is limited by the function set, terminal set and restrictions on the instruction tree size. If we increase the function set and the terminal set, the search space will grow since the GP process will have more combinations of functions and terminals to explore. The size of the instruction tree will also affect the search space. To make GP useful, we must put a limit on the size of the tree; otherwise, the tree can grow infinitely large. Allowing larger trees will result in larger search space.

Smaller search spaces make it easier for GP to find a solution, but the search space must be large enough to be able to cover the problem space. The problem space is the nature of the problem to be addressed by GP. If the problem space is large and complex, GP needs to be equipped to handle many different functions and terminals and might be allowed to build large trees (i.e. large search space). If the problem space is simple and we still permit GP to handle big trees with a large function set, we might make the search space too big and complicate the learning process.

The search within GP could be described as a balance between exploitation and exploration. When GP selects the parents to breed the next generation of individuals, it uses a selection algorithm that selects the parents based on their fitness values. A parent could be selected several times during one generation shift, which could result in the existence of many copies of this

individual in the next generation. Hence, the probability of selecting this parent in the next generation shift will be even greater. This will imply that the algorithm tends to exploit the areas of the problem space that seem to be favorable to investigate, while neglecting those areas that are not promising. This tendency to exploit interesting areas of the problem space is called *search* or *selection* pressure. Different selection algorithms enforce different selection pressure on the search process. If this exploitation and selection pressure becomes overly dominant, the risk is that the GP search will get stuck in a local optimum that is far from the best solution.

The exploration part of GP is managed by the genetic operators. The operators are applied to the selected parents with a probability rate. If the crossover and mutation rates are high, the new individual will explore new areas of the problem space. If this pressure of changing the individual is substantially higher than the search pressure, the influence of good parents is minor since the offspring will not look like the parents anyway, and the search will resemble more of a random walk than a directed search.

3.4. Summary

Chapter 3 has now described the necessary background in human behavior representation, learning by observation and some different machine learning algorithms. It is important to know the features of developing human behavior models and the approach to learning by observation when we next present and validate the methodologies that are the base for the new approach to

learning tactical human behavior by observation.

CHAPTER 4: LEARNING BY OBSERVATION – CONCEPTUAL APPROACH

This chapter describes the new approach, Genetic Context Learning (GenCL), for building human behavioral models from observation. This new approach integrates CxBR and GP to implement the learning part of learning by observation. To complete learning by observation, an observation module needs to complement the learning module. The observer module is not investigated within this research and is posed as a subject for further research by others. This will be further explained in section 4.4. The GP evolves the behavioral knowledge in the context base of CxBR. During the learning process, the individuals within the GP module will be source code programs that represent knowledge in the context base. To be able to evaluate the performance of the GP individuals, a simulation will be run with the individual's code for a period of time, and at specific evaluation points, the individual's behavior will be compared to the recorded human's behavior. This comparison will establish the fitness value for the individual that is essential for the GP learning process.

Before describing details of this new approach to learning by observation, the justification for why CxBR and GP were chosen is presented.

4.1. Choosing an Appropriate Paradigm for Simulated Agents Exhibiting Human Behavior

In order to build autonomous agents with human behavior (e.g. simulated agents) we must choose a paradigm that supports features of human behavior and provides an appropriately structured knowledge base that is appropriate for the task. One important feature of the agent is situational awareness. There exist many definitions of situational awareness. Often, the definition focuses on a specific research subject such as military simulations. In the military context, the definition can include one's own troops, enemies, threats and task goals. In this research, where a generic approach to automatically creating human behavior models is in focus, the definition needs to cover a broader context. Furthermore, we need to distinguish between situational awareness and situational assessment. The assessment concerns the process of how the situational knowledge is achieved, while the awareness is the state of the situation. The assessment phase falls slightly outside the scope of the research covered by this dissertation. Endsley [19] gives a definition of situational awareness that suits this research:

Situation awareness is the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future.

Here we can see that the situational awareness regards the current status of the situation and also what impact it has in the near future. This is essential in the decision of what action to apply. Note that the status of the situation is not only momentarily defined but also includes recent

events that may be important in evaluating the status of the situation.

Situational awareness of the agent infers that the agent must be fully aware of its current situation to be able to choose appropriate actions. If it fails to interpret the environment and the situation at hand, the agent would probably not behave correctly. Turner [76] defines a context as “a distinguished (e.g., named) collection of possible world features that has predictive worth to the agent.” This means that the agent can recognize the current situation as an instance of a known context and then be able to reason about the situation at hand.

Context-Based Reasoning (CxBR) is a modeling paradigm that is using contexts. It has been developed to build agents with human behavior through the use of contextual knowledge. CxBR provides a good hierarchical structure of the knowledge and facilitates situational awareness. The concept with CxBR is focused on situational awareness where the knowledge is structured in contexts applicable to the presently occurring situations. CxBR has proven advantages for modeling agents with human behavior [27]. CxBR shows many effective, positive and efficient features through its design to implement human behavior.

CxBR structures the knowledge in contexts and reduces the search space by considering only the active context. Studying CxBR as an implementation of the modified stage model in Figure 1, the active context is placed in the working memory while the rest of the contexts reside in long-term memory. In this manner, the search space to find the correct knowledge is reduced and the algorithm can operate more efficiently. This structure of the knowledge will also help the

learning mechanism to be implemented, since it also will provide the chosen learning algorithm a structure in which to work. In a similar manner, the hierarchical structure of knowledge in contexts facilitates the simulated agent search for the most appropriate knowledge to apply in a specific situation. Furthermore, it could also facilitate the learning process to learn the knowledge in smaller, more defined chunks of knowledge.

Since the design of CxBR is inspired by how humans break down a problem into sub-problems, the use of CxBR is intuitive. This will ensure that the implementation of learning into CxBR will be flexible and applicable in many different ways. Suppose that the human modeling problem is very complex, which requires that some parts will be manually crafted and others captured by learning. Additionally, in some cases, basic knowledge can be first incorporated within the model that the learning process later refines. In both situations, if the modeling framework is intuitive and easy to use, it will enhance the usability of expert knowledge within the learning process.

Both ACT-R and Soar are structured around the problem-solving process and do not give the intuitive advantage that CxBR does when it comes to model human behavior in simulated agents. Those two algorithms are good general problem-solving architectures, but the clear distinction between human behavior knowledge and general problem-solving knowledge is lost. The learning procedure in ACT-R and Soar is designed to improve knowledge already implemented (i.e. learning through experience). To be able to perform learning by observation in either Soar or ACT-R, new learning mechanisms need to be incorporated. The structure of COGNET does not

include a learning mechanism, although the framework is based on human behavior modeling. None of these three paradigms supports human behavior modeling as intuitively (from a developer's standpoint) as does CxBR, with its clean hierarchical structure that also reduces the search space. This clean hierarchical structure of CxBR might also serve as a suitable structure for a learning algorithm to work in. Where COGNET is structured around a blackboard model wherein all information is posted, CxBR divides the learning problem into smaller sub-problems that might enhance the capabilities of the learning algorithm. The advantages shown here, and the clean structure of CxBR, qualify it as the choice of paradigm. Now we'll extend it with learning and aim to implement learning by observation.

4.2. Learning by Observation with CxBR

To be able to build models automatically, CxBR needs to be equipped with learning capabilities. As mentioned earlier, knowledge within CxBR is composed of action rules and sentinel rules. When incorporating learning into CxBR, the learning paradigm must be able to learn the applicable behavior in a specific context (i.e., action rules), and also the appropriate context switches (i.e., sentinel rules). At different Context levels, the collection of sentinel rules determines which context will be active for that specific level. The structures of the sentinel rules are similar at all context levels. An analysis of CxBR shows that learning sentinel rules are a classification or clustering issue. The objective for the sentinel rules is to classify the current situation and map it to an applicable context. When we look at the knowledge stored within the contexts, the objective is to match the actions within this specific context in the model to the

actions of the human. This tends to be more regression analysis, minimizing the discrepancies between the model and the human performance. This is especially true further down in the hierarchy where the Sub-Contexts or Sub-sub-Contexts approach low-level behaviors such as motor skills. If this is looked upon from a machine learning perspective, the learning tasks in CxBR would incorporate different types of learning strategies, both in classification and regression analysis.

4.3. Choosing an appropriate learning algorithm for CxBR

CxBR is chosen as the modeling infrastructure to construct the human behavior knowledge within the simulated agents. CxBR has to be complemented in some manner to incorporate learning, to automatically create the human behavior models. This section investigates various machine learning approaches and the one with the best prospects.

4.3.1. Reinforcement Learning

Reinforcement Learning is not a strict definition of a specific algorithm. Rather, it is the name of a family of algorithms that peruse learning in a specific manner. Even so, there are many similarities to the new algorithm presented later. From section 3.3.1, we can conclude that Learning by Observation and Reinforcement Learning are different groups of learning problems. Learning by Observation focuses on how the data is collected, while Reinforcement Learning focuses on how the learning is accomplished. Neither of them look at the implementation nor at

the learning techniques to be used for the problem. Learning by Observation covers the problems where the learning is conducted only by processing the observed behavior and adopting it. Reinforcement Learning, on the other hand, includes those problems where an agent can improve its own behavior by observing its own actions. The two different groups of problems are closely related to the fact that observing is a central part in both paradigms. However, since Reinforcement Learning does not comply with Learning by Observation, Reinforcement Learning will not be investigated any further as a method to implement Learning by Observation.

4.3.2. Using Different Machine Learning Approaches

In the current status of CxBR, there are no extensive research results where learning has been incorporated into all parts of the CxBR's context base. Most of the learning within CxBR has so far been focused on low-level behavior, but no one has presented a generic approach to model tactical or unpredictable behavior at different hierarchical levels including context switching [33], [70]. In order to implement learning by observation, it is an objective to incorporate a learning paradigm into CxBR that could learn knowledge in all the different parts of the context base. As mentioned earlier, this means that the learning algorithm needs to be able to handle both classification and regression problems. One approach to learning is inductive learning based on the thesis of *learning from examples*. This places additional strain on the preprocessing of the learning data to create applicable examples prior to learning. This might be the same for other learning paradigms but not as obvious. If we look at decision trees, their knowledge representation would be suitable for implementing learning within CxBR. The drawback with

decision trees is that they focus on classification problems. They would probably not be suitable for the regression problem within the action rules of CxBR. GP implemented with an instruction tree has some commonalities with decision trees, even if the learning is different. GP has a more diverse function set and is not restricted to the production-like rules used in decision tree approaches. Hence, GP is a more generic learning algorithm applicable in a wider area, as regression problems.

4.3.2.1. Transforming the Search Space

Many machine learning algorithms enforce a transformation of the search space to enable learning. For example, artificial neural networks transform the search space to a set of weights whose values are optimized during learning.

Wolpert and Macready [83], [84] 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. When the search space is transformed prior to learning, the knowledge to improve the learning also transforms. Instead of expert comprehension of the problem, the intellectual capacity is now focused on the learning paradigm instead of the problem at hand. A non-transforming learning paradigm supports the use of problem-specific knowledge to improve learning. In a non-transforming algorithm such as GP, the learning prerequisites are closer to the expert knowledge than in a transforming algorithm. Let's look at GP and ANN as contrasting

examples. In GP, one major feature influencing the learning performance is the selection of an appropriate function set. Since GP is a non-transforming learning paradigm, there is a straightforward correlation between the function set and the search space. This implies that SME knowledge would be useful in determining appropriate function sets. In ANN, one thing that influences learning is the structure of the network, such as numbers of nodes and layers. This knowledge is not directly correlated to the problem to be solved, and the use of the SME would not help in this case.

4.3.2.2. Transparency

One issue for the learning paradigm is to preserve the features of the CxBR that make it appropriate for modeling simulated agents with human performance. One of those features of CxBR, when applied the traditional way, is that the knowledge stored in the context base is transparent - the knowledge will be stored in source code. Hence, the knowledge resulting in the agent's action can be inspected and evaluated (i.e. the knowledge is transparent). This is important if human features such as episodic memory and communication are to be incorporated into the model.

If the learning algorithm, used to build the knowledge automatically in the CxBR structure, is transforming the search space in any way, it will be more difficult to interpret the knowledge. ANNs are mostly regarded as an opaque algorithm where the knowledge stored is very difficult to interpret. In the case of GP, the learning algorithm is transparent because GP evolves source

code statements. As source code is easy to understand, and with a close resemblance to written language, it is rather easy to interpret. Notable about source code evolved with GP is that it is not structured as well as normal source code and might include *non-coding regions*. Non-coding regions are unexplored code that is never tested during training or does not affect the output when training input patterns are presented to the individual. Non-coding regions should not be correlated to the uncontrolled growth of the individuals (i.e. bloat) [47]. Even if the code is not affecting the results during training, it has been shown that information embedded in the non-coding regions actually improves the learning process [24],[54],[82]. Analogies to this can be found in animals' DNA structure. There exists chunks of information in our DNA that is not used for anything by the individual, but it can store valuable information used in individuals several generations later [45]. Even if the source code is unstructured and includes non-coding regions, it is interpretable and can be evaluated (automatically or manually) in a comprehensible manner.

4.3.3. Using GP to Implement Learning in CxBR

From the discussion, we can see that GP has advantages such as a transparent and non-transforming algorithm. GP has also been used in a wide variety of machine learning areas and the choice in this research is to extend CxBR with learning using GP. The new approach to building human behavior models by observation is called Genetic Context Learning (GenCL).

An advantage with GP is that if a-priori knowledge is available, it could be easily incorporated as

a starting point. This is an approach that ANN is not capable of accomplishing. The strategy used in this research, regarding a-priori knowledge, is further discussed in section 4.6.1. The GP-device would then be allowed to refine the performance and behavior of the CxBR modules. If we have basic knowledge on the behavior of a car driver, we could let some of the individuals in the population start with those features and let them evolve to fit the behavior of the observed human better. Even a very small amount of knowledge would likely improve the performance and accelerate the learning. However, it could be more beneficial to begin with some rather detailed models of behavior, and let them personalize through learning by observation with this approach. Let's assume that we would like to create several car agents with different types of human behavior in a simulated environment. We could begin with a generic car model that we would develop after different human observations to create a set of agents with different behavior. The ability to refine already existing knowledge also shows that the learning algorithm is capable of doing learning by experience. If the original creation of the simulated agent uses learning by observation to incorporate human behavior, the learning strategy could remain within the agent so it could later improve its performance. Hence, intelligent agent behavior could be enhanced by learning during operation in a simulation or in the real world.

The integration of CxBR and GP also provides the opportunity to choose alternative representation of the function set. Several researchers have argued that the use of Fuzzy Set Theory and Fuzzy Logic would enhance the humanness in human behavior models [12], [39], [43]. Fuzzy Sets deal with real-world problems where instances have a degree of membership in sets, as opposed to normal crisp sets. Fuzzy Logic is then the logic operator that could be applied

to the Fussy Sets. A detailed description of Fuzzy Logic and Fuzzy Set Theory could be found in Zadeh [85]. If this was a feasible approach, the GP device could be set up to evolve rules built on Fuzzy Sets and Fuzzy Logic. The function set in GP will then consist of Fuzzy Logic and a Fuzzy Set will describe the terminal set. The possibility of incorporating Fuzzy Logic is an advantage of using GP as a learning approach. The implementation issues and function set used in this research is further discussed in section 5.3.1.

4.4. Employing CxBR and GP towards Learning by Observation

The choice of learning strategy to implement in CxBR is, of course, GP. The advantages described above indicate that GP would be a useful and flexible algorithm to evolve contextual human behavior knowledge within CxBR.

Instead of creating the contexts manually in CxBR, the GP process can be used to build the contexts. The GP's evolutionary process can provide and build or refine the CxBR's context base with appropriate contexts, functionality and sentinel rules (see Figure 15). The individuals in the genetic population represent parts of the context base, and the simulation acts as the evaluator in the learning process. Since there is CxBR knowledge to be evolved by the GP, the Micro Simulator within GenCL is also designed according to the CxBR paradigm so that the individuals operate in the same environment during learning as in their final use. During the learning process, each individual within the GP module represents some behavior knowledge, and each individual is loaded into the Micro Simulator module to be executed. This execution

results in a simulation with the knowledge contained in the GP individual. The results of this simulation can then be compared to the performance of the human being modeled and a fitness value for this specific individual can be computed. In effect, the human performance observed serves as the fitness function against which to compare each individual's performance. This procedure is conducted for all individuals in the GP population. When all individuals have received their fitness value, the GP can evolve a new generation.

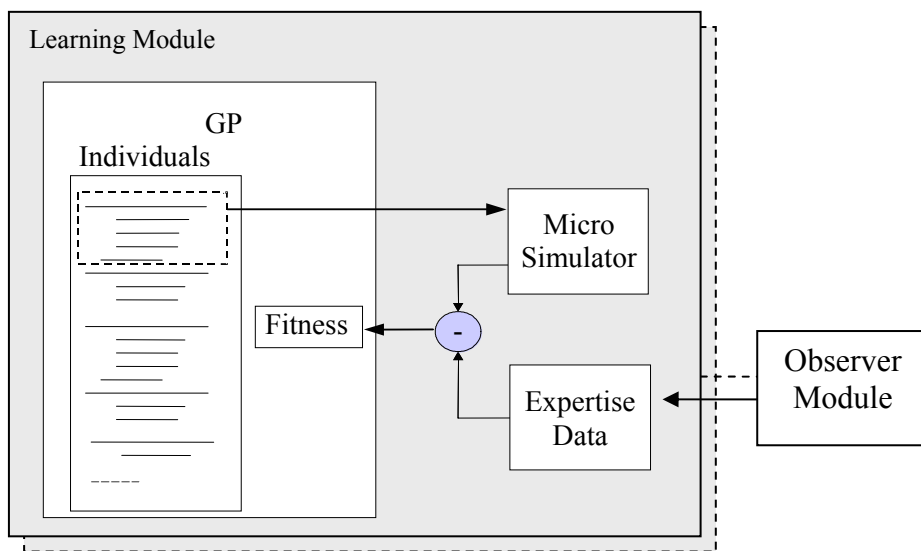


Figure 15: Learning by Observation with GenCL

An example of the learning process can be found in the automatic creation of CGFs. The GP individuals could represent a part of the CGF's action within a specific context (e.g. attack). As the knowledge evolved in the individual is simulated in the Micro Simulator module, it can be compared to the action taken by the SME at the same situation. The simulation must be able to

simulate the context within the same environment as the SME experienced. Hence, the discrepancies between the individuals in the GP population and the action taken by the SME will be reduced as GP evolves new generations of individuals. Note that the performing entity being observed need not always be an expert. If the objective is to create an application that predicts the force movements in different situations, the observed entities could be trainees, enemies or other entities with arbitrary skill levels.

In short, the GP creates individuals that represent the behavior pattern for the context in focus. Each individual in the population is an implementation of the action or sentinel rules for this specific context, and is a candidate for the final solution. To incorporate each one in the simulation and compare them with the human performance, a ranking among the individuals could be established to the degree that they reflect the behavior of the observed human. Hence, better-fit individuals are more likely to evolve the next generation.

The simulation part in Figure 15 is essential in order to ensure the correct functionality of the knowledge learned. The simulation restricts the new GenCL approach to learning knowledge that could be described and evaluated in a simulator. However, the well- defined simulator function makes it easy to customize the algorithm to model and build different simulated agents (e.g., ethnic or political groups, army troops, pedestrian, aircraft or submarine) that inhibit human behavior patterns.

The observer module in Figure 15 is the sensing and perception interface of this new learning by

observation architecture. Here, the observations from the environment are processed to fit the learning strategy. This module must be configured for the specific learning environment. If the learning from observation task is performed in the real world, this module needs to process the different sensor readings and prepare the incoming data in an applicable way. The other option is that a human operates a simulator, and then the observation module needs to handle the data from the simulator. In the observation module, the data also need to be partitioned and filtered. The configuration of this module needs to be aligned with the configuration of the GP module. The terminal set of the GP module requires the observation module to feed the correct data set. All sensor values or available data might not be used in the learning process. All of the available data might not be used during the training process either. If the sampling rate, for example, is 10 Hz, then the amount of data fed to the system will grow very fast. Hence, if the human to be modeled operates in hours, the problem space will be very large. This data needs to be filtered and partitioned in an applicable way.

To achieve a complete learning by observation device, several learning modules could conceivably be operating in parallel to produce a complete context base for an agent. If the observer module partitions the data in an appropriate manner and feeds each of the learning modules with suitable data, it would be able to perform on-line learning. In on-line learning, learning is done at the same time as data collecting, so the filtering and partitioning of the data in the observer module must be done in real time. This is left for future research.

The observer module has not yet been automated and instead relies on the specification from the

knowledge engineer. The objective in this research was to investigate the feasibility of integrating CxBR and GP to achieve our stated objectives. Since the computational task was too complex to employ the complete data set, the filtering and partitioning of the data has been done manually in this research. This research investigates the learning module to see if CxBR and GP have the ability to learn by observation when the learning module has fed it with the appropriate data.

4.5. Empirical Studies – CxBR and GP

Previous sections showed the basic ideas on how GP and CxBR could cooperate during the learning phase to implement learning by observation. Here we describe the merging of CxBR and GP in a more formal way so their cooperation is thoroughly presented.

4.5.1. In Depth Study of CxBR

The knowledge constituting the agent's intelligent behavior in CxBR is stored in the different contexts. The format of the knowledge is not fixed in CxBR; only the knowledge structure is defined. When building human behavioral agents with CxBR, the most common knowledge representation paradigm consists of functions and IF-THEN rules (i.e. source code). These functions could be predefined in the programming language package or be user-defined command sequences. A special form of a command sequence is a context at a lower level (e.g., sub-context). Note that in this section the use of *sub-context* refers to a generic context at the

next lower hierarchical level in CxBR. This is not to be misunderstood as a specific Sub-Context that refers to a context at the level underneath the Major-Context. Hence, a sub-context (Major-Context or Sub-Context or Sub-sub-Context, etc.) could also have a sub-context.

If we do not distinguish between functions and operators, we could define a function set, $F = \{f_1, f_2, \dots, f_m\}$, that processes the information to deliver actions. The function set could consist of the following types (we use C/C++ notation for our examples):

- Arithmetic operations: $+, -, *, \dots$
- Relations: $<, >, ==, \dots$
- Mathematical functions: $\cos, \sin, \text{pow}, \dots$
- Boolean operators: $\&\&, ||, \dots$
- Command sequences: $\text{distance_to}(), \dots$
- Conditionals: $\text{if}, \text{else}, \dots$
- Iterations: $\text{do}, \text{while}, \text{for}, \dots$
- sub-contexts: $\text{TrafficLightDrivingContext}(), \dots$

The function set is only useful if it can process information. Information has value and is stored in variables or constants. These information storages are called terminals and the information available is then defined in a terminal set, $T = \{t_1, t_2, \dots, t_n\}$. Note that some of the values in the terminal set could originate from sensor reading and are therefore time or instance dependent.

The function and terminal set are defined to highlight the integration with the GP.

The Mission Context, MC , mainly defines goals, G , and plans, P . Plans are realizations to reach the goals for one or many intelligent agents. The action specified in the Mission Context is mainly used to let the plans invoke Major Contexts in an appropriate sequence so the goals can be fulfilled. The active context is the context that controls the agent (i.e., the function set of the active context, F_{ac}) and defines the agent's action. At least one default Major Context is specified as a part of the sentinel rules in the Mission Context. With the sentinel rules, R , a specific mission, MC_y , is now defined as:

$$MC_y = \{G, P, R\}$$

The contexts from the Major Context level down in the hierarchy all have the same structure.

The knowledge contained in a Context could be described as a set of action rules, A , and a set of sentinel rules, R . The action set describes how the entity will behave within the present situation (i.e. active context). The sentinel rules determine whether the active context should remain active or if control is to transition to another context. Now a specific context, C_x , could be described as:

$$C_x = \{A, R\}; \quad A = \{F, T\}, R = \{F, T\}$$

Both the action set, A , and the set of sentinel rules, R , contain a function set, F , and a terminal set, T . The set of sentinel rules, R , also contain a list of valid context transitions, which actually

could be viewed as a specific implementation of a terminal set (i.e. an array of context pointers).

4.5.2. Formalizing GP

The formal description of the GP functionality presented here is derived from Holland's formal description of GAs [34]. An adaptive evolutionary system such as GP always operates in an environment, E . It is only through the interaction with the environment that we can measure the performance of the learning system. This is a key factor in GP, where the genetic process evolves new individuals based on their performance. GP represents its individuals as a finite set of structures, $S = \{s_1, s_2, \dots, s_n\}$, often referred to as genotypes, that are represented as programs (e.g. source code individuals) in GP. A structure, s_x , consists of a function set, $F = \{f_1, f_2, \dots, f_m\}$, and a terminal set $T = \{t_1, t_2, \dots, t_n\}$. Note that these could be the same terminal and function sets as in CxBR. It is in this commonality of terminal and function sets that we find the integration of CxBR and GP so synergistic. The first step in each generation is to transform the genotypes into phenotypes, $P = \{p_1, p_2, \dots, p_m\}$. This is a requirement for interaction with the environment. This is done with a genotype-phenotype mapping function μ such that:

$$p_x = \mu(s_x)$$

In GP, the mapping function μ is the compilation or interpretation of the source code (i.e. the genotype individuals). The interaction of the phenotype with the environment is, in GP, the execution of the phenotype:

$$\varphi_E(p_x)$$

The next step in the GP process is to evaluate the performance of the individual through the fitness function μ_E :

$$\mu_E(s_x, \varphi_E(p_x))$$

Note that the fitness function μ_E could be a function of both the performance of the phenotype in the environment and the genotype, s_x . This could, for example, encourage smaller individuals with unnecessary or less complicated source code. The fitness function returns a fitness measure based on how well this individual's combined performance was. Depending on the fitness measure of the individual, the selection scheme, α , selects an operator, $\omega \in \Omega$. Ω is the set of all operators where:

$$\Omega = \{\omega_x: S^m \rightarrow S^n\}$$

An operator ω maps m structures into a set of n possible modified structures. Individuals with a better fitness measure have a greater chance of being selected. Each operator type is a stochastic operator that will be triggered with a predefined probability. If neither stochastic operator indicates applicability, the selected individual will survive (i.e. being cloned) to the next generation without any modifications. After the selection scheme has selected individuals to breed and appropriate operators have been applied, a new population now exists, partly with new

individuals and partly with individuals from the last population. The two most common operators in GP are *crossover* and *mutation*. The process will continue to map the genotypes into phenotypes, execute them and evaluate their fitness, and again apply operators to the selected individuals (see Figure 15). This continues until some stopping criterion is met. According to the Schema theory [34] or the Building Block Hypothesis [25], this procedure will evolve better and better individuals until a global or local optimum is reached.

The only problem-specific function in the CxBR function set is the sub-context (i.e. any context at any level except the Mission Context level), but it is only a special form of a command sequence. GP has the ability to evolve and develop command sequences, also referred to as Automatically Defined Functions (ADFs) by Koza [40]. Hence, besides the ability to build the knowledge within the contexts, GP also has the ability to develop and destroy sub-contexts. This means that the GP possesses the ability to evolve appropriate sets of contexts within the context base. In other words, GP has the ability to create the context hierarchy and the knowledge within the contexts from scratch. As Contexts are more abstract than functions and inherits more features, that makes it a context and not only a function, the ADF functionality of GP needs to be customized to enable Context creation and destruction.

Since GP uses the same function and terminal sets as CxBR, GP 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 from scratch.

4.6. Integrating CxBR and GP

The basic idea and the formal way to integrate CxBR and GP have been presented. So far, the generic approach has been described that could be applied in many different ways in different applications. We have not yet placed any restrictions on what function set to use (e.g. fuzzy or crisp logic) or if the GP module should create the knowledge base from scratch or refine some initial structure.

In this research, the CxBR and GP integration was evaluated by applying it to automatically evolving simulated cars with human behavior. Five different drivers used a Driving Simulator. The task for our new approach was to evolve five different simulated car agents with behavior patterns that reflect each of the five drivers. The complete description of the experiments conducted are later described in *CHAPTER 5: THE MODEL BUILDING PROCESS*. From now on, the decision plans and strategies discussed are influenced by our research objectives as well as by the application we selected to verify the GenCL approach.

4.6.1. Plan for the Integration

When it comes to most machine learning algorithms, the learning could be classified to be somewhat of a refinement process or a construction process for the knowledge. One could view it as a continuous scale where one side is solely a refinement procedure and the other side is a constructive procedure.

Far out on the refinement side, the model's processing elements have already been pre-constructed and most of the knowledge and the learning procedure are merely an adjustment of the model to fit the problem at hand better. On the other side of the scale is the constructive end. Here, no knowledge or structure is available a-priori on how the knowledge is to be represented. The learning process then involves both creating the knowledge structure as well as the knowledge itself. Note that the closer we get to the constructive side of the learning scale the more difficult the learning task becomes (i.e. larger problem space). However, a constructive process is more valuable than refinement, as it requires less manual effort to build the model.

The flexibility of GP then allows many different approaches to learning. The learning could begin with known knowledge and refine the model to better suit the behavior of the observed human. If we take the example of a driver operating a car simulator, the learning system could already consist of a general human car-driving model. The GP could start off with this model and refine it during the observation of the human in the simulator to evolve a model more comparable to the current driver. On the other hand, GP could be set up only with an appropriate set of functions and terminals and it could construct the knowledge from scratch by observing the driver. The latter is more desirable when investigating learning by observation, since it would reduce the implementation effort even more. However, restrictions need to be enforced since the GP has to evolve knowledge according to the CxBR paradigm. The GP process could be allowed to evolve the context sets in all the hierarchy levels, the action rules in each context and all the appropriate sentinel rules in all the contexts.

The No Free Lunch theorem [83] and [84] state that if we apply any search or optimization algorithm over all possible problems we will not get a better average performance than a random search. The reason that certain algorithms perform better than other algorithms on some specific problems is because there are problem-specific features or knowledge incorporated in the algorithm that suit the problem. In other words, the more problem-specific knowledge incorporated in the algorithm, the better its chance for success.

If GP is to be applied to build the context base in a constructive manner, some sort of organized structure needs to be enforced in the learning process to increase the probability of success. Even if this approach seems more complicated than a refinement procedure, it is the aim of this research to investigate these constructive features of GP since it would reduce the development effort more than a refinement strategy. One of the biggest advantages with a system that could learn only by observing someone's performance would be the reduction of development cost in modeling the behavior. If a major part of the behavior still needs to be modeled before learning can take place (i.e. refinement), the advantage of introducing learning to CxBR would probably not be significant. Therefore, it is more desirable to investigate GP as a constructive learning paradigm.

The research in this project does not investigate the GP's ability to construct and delete contexts. Instead, the number of contexts and their position in the CxBR hierarchy will be pre-defined and fixed and will serve as a structural framework to support learning capabilities. The contexts will initially be empty and will not contain any knowledge whatsoever. This represents a tolerable

compromise that provides a nearly constructive learning process without the computational complication of a fully constructive method. It will also give GP a structure to operate within that could improve the learning performance. Additionally, it will dramatically reduce the development cost and effort in creating intelligent agents with human behavior. Still, a strategy needs to be defined on how the learning procedure is to be conducted. The knowledge to be learned is still structured in different levels in the hierarchy and the different parts of the knowledge might be dependent on each other.

4.6.2. Strategy for Learning within CxBR

The task of implementing a learning strategy applicable at all levels of the CxBR hierarchy is very complex. The problem domain ranges from very low-level decision-making (e.g., how to apply brakes) to high-level tactical decision-making (e.g., cautious low fuel-consumption driving). Many of the parts constituting tactical human behavior are also correlated with each other. There is correlation with other aspects of tactical behavior when, for example, driving in city traffic and approaching an intersection with a red traffic light and there are cars already waiting at the light. If we have one sub-context that handles traffic lights and another that handles cars in the same lane, those two Sub-Contexts are interdependent in that situation. If direct transition between contexts is used, those two contexts are correlated (i.e. the learning of these two instances could not be performed mutually exclusively). Hence, we can identify two main problems concerning the learning implementation:

1. Hierarchical complexity
2. Interdependency among contexts and context parts

The first problem points to the need for some sort of structure for the implementation of learning in the different contexts. Evolving human behavior structure in all different hierarchical levels at the same time is complex. The search space would probably be large. Different approaches have been adopted to attack this problem. One approach is to reduce the search space by letting the GP evolve higher-level behavior by using a meta-language, where the function set is a set of low-level behavior routines [46]. This restricts the problem solving in many ways but also contradicts the idea of learning by observation. If we used this technique, we would need to manually construct the behavior in the lower context levels that we do not try to learn. This means that there is a major part of the CxBR structure that must be manually created. Another approach would be to construct a very complex and sophisticated fitness function to control the learning in the different parts of the context base. This moves the complexity and manual design from the low-level behavior to the fitness function, and consequently, the learning task is considerably more complicated. Moreover, this still implies a significant amount of manual coding.

Hsu and Gustafson [36] propose an alternative GP learning approach where the high-level behavior would be broken down into lower level behavior that is first learned and then used when higher-level behavior is being learned. The approach is called Layered Learning GP (LLGP) and breaks up the complex problem into a hierarchy of sub-problems. Hsu and Gustafson showed promising results that improved the learning capabilities using the LLGP

strategy. This is a type of bottom-up strategy and it suits our needs well because the problem domain in CxBR already has a hierarchical structure. To apply LLGP to implement learning by observation within CxBR, one simply starts the learning by considering the lowest context level. The learning problems at this level are relatively simple and well defined. When the learning system has been able to establish the contexts at the lowest level, the learning can continue with the next higher context level, and the already learned context could be used as building structures in the GP's function set.

The other problem is the issue of interdependency between the knowledge in the same level of contexts. The interdependency between different action rules in the contexts at the same level might not be that strong, since only one context could be active at the same time (i.e., mutually exclusive). Rather, the dependency is in the combined performance of the action and the context switching (i.e., sentinel rules). When the sentinel rules are implemented in a direct transition manner, the collection of sentinel rules at the same context level is interdependent. For a sentinel rule within a context to be able to activate the context, any other contexts at the same level need to release its activation. The joint performance of all sentinel rules, at the same level, needs to be evaluated in some way. This means that the sentinel rules in the different interdependent parts could not evolve separately. Since the performance in one part is dependent on the performance in the other part, these parts need to be evolved together simultaneously. An approach to this problem is to use the Cooperative Co-Evolutionary strategy [60]. In this strategy, it is possible to have different populations evolving solutions to sub-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 and measure their performance. Mendes, et al. [49] used Cooperative Co-Evolution to successfully evolve fuzzy classification rules. They used GP to evolve the different classification rules and a simple EA to evolve the membership function definitions. In a similar way, we allow the different contexts' sentinel rules to evolve in parallel, and evaluate their joint performance when the fitness value is calculated. We use LLGP and co-evolution as our basic GP strategy for learning human behavior in CxBR from observation.

4.7. The GenCL Algorithm

Figure 16 shows the GenCL algorithm. Here, the GP part of the algorithm is sparsely described, since the common GP algorithm was described in section 3.3.5. As in most GP applications, the first task is to create a random population of individuals from the function and terminal set specified. The algorithm needs to be able to calculate the fitness value for all the individuals. Each individual is now placed into the Micro Simulator and executed for a number of simulator clock cycles. The individual is first initialized in a position and speed at a predefined location of the human driver's recording. Now the individual has the same initial condition that the human driver had when he or she was operating the Driving Simulator on which his or her actions were observed. The individual then experiences the same environment as the driver but controls the simulated car by itself. This continues for a few simulator cycles until the next evaluation point (e.g., 60 ms later).

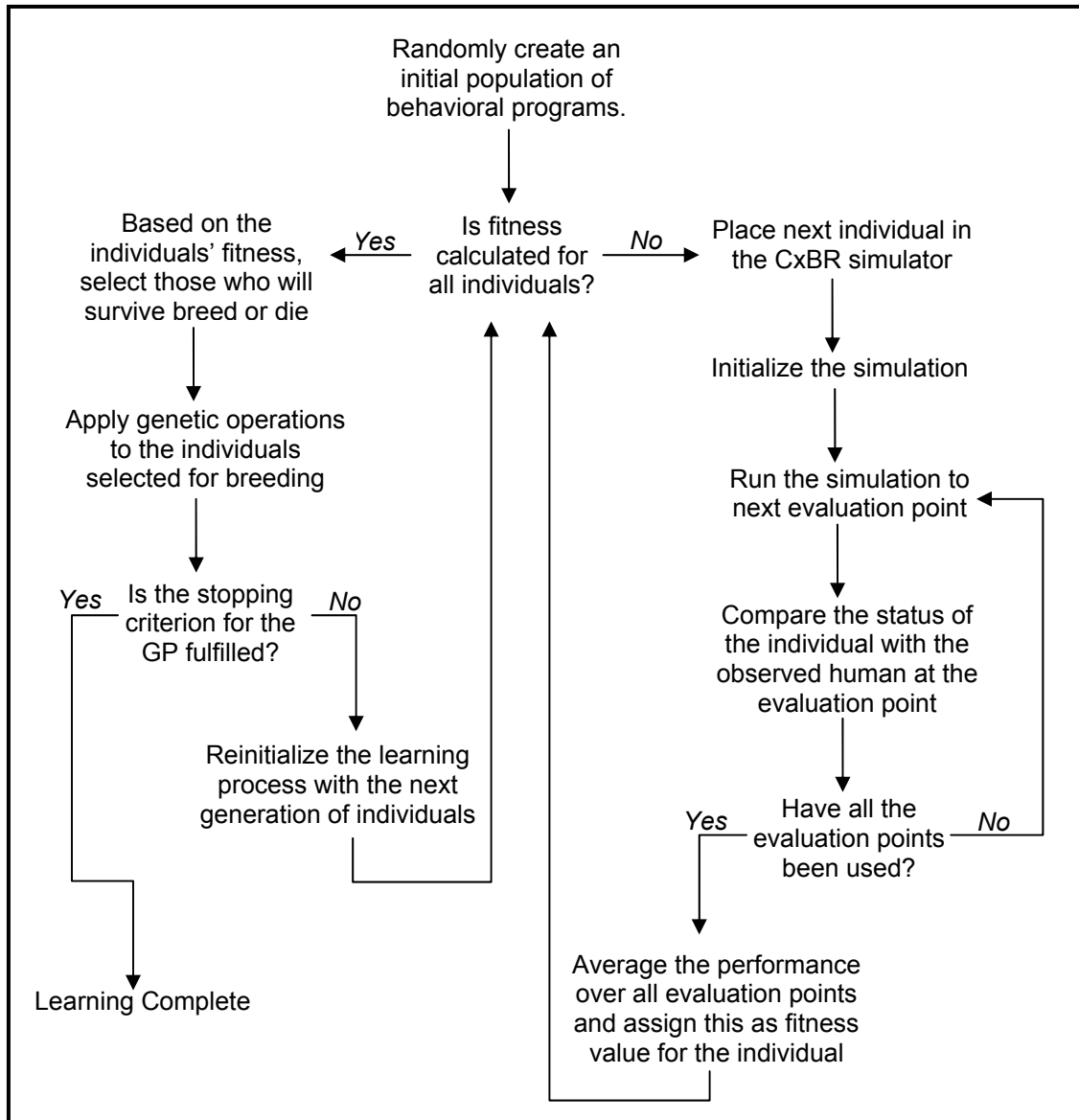


Figure 16: The GenCL algorithm

Here the performance of the agent is compared to the performance of the SME. Next, the simulation continues until the next evaluation point and the deviation there is calculated. This continues until all evaluation points of the learning scenario are covered. How to select the

evaluation point is problem dependent and differs from case to case along with issues of the observer module (see Figure 15). In this research, the evaluation points were selected manually. This is further described in section 5.1.1.

Note that at each of the successive evaluation points, the deviation is probably larger than the prior ones, since no reset of the individual's speed and position is done at each evaluation point (i.e., errors accumulates). This is actually preferable, since it will punish the algorithm if it collects accumulated deviations. When the individual has passed all evaluation points, the average deviation is calculated, representing an individual's fitness value.

When the fitness values are calculated for all the individuals in the population, the GP module steps in and selects individuals to breed the next generation of individuals to the next generation. An individual's probability of being selected is greater the lower the fitness value is. Note that this is because the fitness value in our case is the average deviation. Hence, the lower fitness value is better. Then the algorithm applies genetic operators to the selected parents and new individuals are created. The newly created individuals and the ones surviving from the last generation now constitute the next generation of individuals. Individuals not selected will not contribute to the next generation and are discharged. The learning procedure continues to calculate fitness for all the individuals in this new population with the help of the Micro Simulator until the GP stopping criterion is met.

4.8. The Synergistic CxBR – GP Integration

Using CxBR as a structure to store knowledge gives GP a sound framework in which to operate, and it gives GP the prerequisites to enhance it with the LLGP strategy. GP could be used without the CxBR structure, but to evolve complex behavior patterns would imply complex fitness functions, and the search space would be very large. CxBR limits the search space when searching for the stored knowledge in the context base, and it also reduces the search space during the creation of the knowledge (i.e. learning). The hierarchical nature of the contexts divides the problem into smaller sub-problems of a suitable size. By using the expert knowledge to pre-define the valid contexts and sub-contexts, a coarse knowledge framework is established for the GP to evolve more complex behavior. In this research, the only pre-defined structure used prior to learning was five valid contexts at three levels (further described in section 5.2). All the contexts were empty and contained no knowledge. Only the context frames were defined and their hierarchical relationship. This will provide a suitable framework for GP to operate in, but it will not restrict the behavior model in those situations. The GP-learning algorithm will still be able to map an unlimited number of different human behavior patterns within those situations. Remember that the objective here is not to model the best behavior or the average human behavior. The intent is to capture the specific behavior of the current human performance, no matter how good or bad his or her performance might be. The only restriction presented is the number of situations the model will be able to handle.

4.9. Summary

In this chapter, the CxBR and GP were validated as advantageous approaches to merge into a learning engine for learning by observation. The basic structure of the new approach, called GenCL, was described with the extension of configuration-specific issues related to the objectives in this research. The GenCL algorithm was presented along with sufficient theoretical analysis made to continue with practical experiments. The next chapter will describe the configurations of the model building process used to validate the practical use of this new algorithm.

CHAPTER 5: THE MODEL BUILDING PROCESS

This chapter describes the application, data collection and configuration of the experiments used to verify the new approach to learning by observation (i.e. the GenCL algorithm). As stated before, the objective was to prove the feasibility of combining CxBR and GP (i.e. the learning module in Figure 15). The application area was automobile driving behavior. A commercial driving simulator was used to collect the data. Since this simulator was only available for the data collection and the observer module has not been implemented yet, the learning was conducted off-line in these experiments.

The final use of the knowledge evolved by the GP is the context base of a CxBR model. This model is used in a simple Traffic Simulator, developed with the CxBR framework [55]. Five different drivers were used to drive the commercial driving simulator in simulated city traffic. The experiments examined whether GenCL could model the drivers' behavior during normal operating conditions merely from observation.

Note that three different simulations are mentioned in this dissertation, which can be confusing. One "simulator" is the commercial driving simulator used to collect data. We refer to this as the *Driving Simulator*. Another "simulator" is the small simulator used within the GenCL artifact to

evaluate the individuals during the learning phase. This “simulator” is referred to as the *Micro Simulator*. The third “simulator” we discuss here is the final use of the fully evolved agents within a CxBR traffic simulation. This simulation is built from the CxBR Framework and it will be used later to evaluate the performance of the evolved agent’s performance. Hence, we refer to this as the *Traffic Simulator*.

5.1. Data Collection

The data for the experiments was collected in the Virtual Technologies’ driving simulator in Linköping, Sweden. The simulator is a full-scale driving simulator where the driver sits in a car cabin and the simulated environment is projected on three walls. See Figure 17.

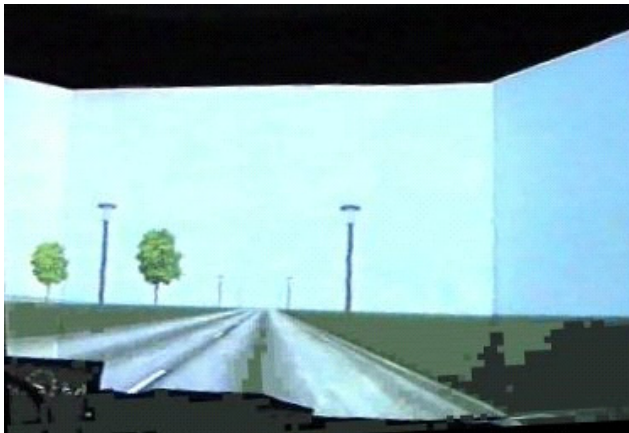


Figure 17: Virtual Technologies’ Driving Simulator in use

This commercial Driving Simulator is used for simple driving simulations. It is not a “top of the

line” simulator, as it lacks motion capabilities. However, it provides a sufficiently realistic driving experience to permit adequate testing of our learning by observation approach. The use of a commercial simulator has an advantage in creating the realistic environment, but it suffers from some drawback on the training tasks. The simulator was only available during the data collection phase and not accessible to use as a test-bed for the fitness function. Hence, a small simulator module (i.e. the Micro Simulator) had to be created to emulate the commercial Driving Simulator’s car model. This introduces a small deviation in the learning module. See further discussion on this issue in section 5.3.1.3.

The data collected consists of two sets. The first set was used as a base for training the five agents to be created (one agent for each driver). The second data set was used for validation and was not included in the training set. The validation set consists of new but similar situations used to evaluate the agents. The validation set was collected four months after the training set. The time between the collections of the two sets was purposely designed to give the drivers a lack of preparation when entering the Driving Simulator the second time, which was the case when they performed the training runs. The design of the scenarios included two types of driving, city driving and rural driving. The drivers spent approximately two thirds of their time in the city and the rest on a rural road. The collection of each training set took approximately 30 minutes, while the validation runs were a bit shorter - 20 minutes.

The five drivers were randomly selected students from Linköping University, Sweden. Before doing the actual drive where the data were collected (the observed drive), the drivers were able

to take a short test drive (approximately 15 minutes) in the simulator to get used to the environment. After the observed drive, the drivers were asked to comment on their experience during the drive.

The scenarios were designed to cover several interesting situations. When in a city, each driver was presented with traffic lights and intersections. Another scenario presented to the drivers was a set of hazardous situations of varying severity, both in city driving and on the rural roads. The least severe situation might not be recognized by the driver as a hazardous situation while the most severe one needs the driver's reaction to prevent an accident. During the drive to collect training data the driver passed 11 traffic lights and experienced seven hazardous situations of varying severity. The rural part and the hazardous situations in the data sets were not used in this research and not further discussed here. The city driving data contained more personalized behavior and was chosen to be used for this first evaluation of the GenCL algorithm. A complete description of the entire data set is found in *APPENDIX C: COMPLETE DESCRIPTION OF THE DATA SETS*.

The environment for the experiments was set up to ensure that the behavioral patterns of the drivers were neither predictable nor trivial. As Banks and Stytz [5] stated, one feature of human behavior is unpredictability. An example of how to trigger this behavior from the drivers is found in a traffic light changing from green to yellow and then to red. If this change takes place at an appropriate distance from the car, the drivers will make a decision on whether to slow down to a stop when the light turns yellow, or continue and pass through the light while yellow. The

distance to trigger this diverse behavior among people seemed to be when the car is 30 meters prior to the light when driving in Swedish city traffic. When the training data sets were collected, we were able to capture this difference in behavior patterns. Even if the lights always change from green to yellow (to red) at 30 meters ahead of the light, there was a significant difference in the experts' behavior. Depending on the environment in the light's proximity and the current speed of the car, the same driver would react differently at different lights. Also, a difference among the drivers could be detected in their behavior, as described in Table 1. S stands for stop and R for running the light while yellow. When the driver does not stop, his speed is usually so high that he would pass the light when it's still yellow. One driver drove carefully and stopped at all yellow lights, while others stopped at some yellow lights and ran others.

Table 1

Behavior of the five drivers at the six traffic lights that changed from green to red.

	Light 2	Light 3	Light 5	Light 6	Light 7	Light 9
Driver A	S	R*	S	R	R	S
Driver B	S	S	S	R	R	S
Driver C	S	S	S	S	S	S
Driver D	S	S	S	R	R	S
Driver E	R	S	S	R	R	S

* After the observation run, driver A explained that he became stressed and by accident ran the red light. This was not his normal behavior and this particular incident was regarded as an outlier and removed from the training set.

Four of the 11 lights faced change from red to green; six from green to red and one is constantly green. All the changes initiate when the agent is at a distance of 30 meters before the light.

Swedish traffic lights have a state of yellow both in the transition from green to red (3 seconds)

and in the transition from red to green (1.5 seconds). As a transition of the traffic lights, either from green to red or from red to green, is described in this dissertation, the state of yellow is always part of the transition even if it is not mentioned.

Because the aim of these experiments was to show the validity of this new approach to automatically build individualized context knowledge, the success of the experiments is when the deviation between the agent and the driver whose observed performance was used to learn is small.

Figure 18 shows the city driving part of the test-bed for the collection of the training data. Note that the rural part is not shown in the picture. The red line shows the major path through the city. Green circles represent traffic lights and blue circles mark where hazardous situations occur. Some of the traffic lights were passed several times in different directions. This is why the number of traffic light occurrences is eleven. The hazardous situations and rural driving are not described here because they were not used in this research. Normal city driving data had more individual behavior pattern and it gave enough results for the research objectives. A complete description of the collected data can be found in *APPENDIX C: COMPLETE DESCRIPTION OF THE DATA SETS*.



Figure 18: City driving training data. Green circles are traffic lights and blue circles are hazardous situations.

While the human subject operates the car in the Driving Simulator, data from the simulation are captured at a rate of 10 Hz. Data points captured include position, heading, pitch, roll, steering wheel angle, throttle pedal pressure, break pedal pressure, speed and distance to closest “Hot Spot”. Hot Spots are non-visible objects at important locations in the simulated environment, such as traffic lights or road maintenance areas. The Hot Spots are implemented in the environment to assist the analysis of the simulator run. An example of the captured data can be found in *APPENDIX C: COMPLETE DESCRIPTION OF THE DATA SETS*.



Figure 19: The setup of the city driving during validation scenarios. Green circles are traffic lights and blue circles are hazardous situations.

During the collection of the validation data that took place four months after the collection of the training data, the same drivers took another route and experienced new scenarios. Figure 19 shows the city driving part of the validation scenarios. The validation drive was a bit shorter than the one used for collecting training data. Once again, the drive was partly conducted in city traffic and partly on rural roads. The black line shows the route through the city, green circles are traffic lights and blue indicates hazardous situations.

Now in the collection of validation data, the driver was exposed to seven different traffic lights and seven hazardous situations. One light is changing from red to green as the driver approaches the traffic light while the rest change from green to red. The timing of the traffic light was now

set to change at different distances instead of only when the driver is 30 meters before the light to see whether the agents were able to generalize their behavior. The setup for the different traffic lights is shown in Table 2.

Table 2

Changes of traffic lights during the validation data collection

	Activation Distance [m]	Change
Traffic light #1	30	Green > Red
Traffic light #2	35	Green > Red
Traffic light #3	30	Red > Green
Traffic light #4	40	Green > Red
Traffic light #5	35	Green > Red
Traffic light #6	30	Green > Red
Traffic light #7	35	Green > Red

Table 3 shows the behavior of the five drivers as they pass the six lights turning from green to red in the validation setup. If we compare their action in this simulator run with their first run (the training) four months earlier described in Table 1, we can see that driver E behaves inconsistently. In the training simulator run, his driving style was rather aggressive and he ran more yellow lights than anyone else. In the validation run he was rather careful and, together with driver C, was the one who stopped most frequently at lights turning red.

The different behavior in the two runs is one of the drawbacks (in terms of machine learning) that could occur by having a gap in time between the two data collection occasions. If the machine learning algorithm is to capture the drivers' "normal behavior" but the drivers do not

exhibit this, it will be difficult to capture. The driver might have been in different emotional states in the two runs and might have shown a behavior in one of the occasions that could not be regarded as his normal driving behavior. The objective of the time gap between the two runs was to ensure that the drivers were unprepared for the two simulator runs each time. Driving in a simulator differs a bit from driving a real car. As the driver gets used to it, anticipating the scenarios played, the driver might adjust his or her driving behavior. In the simulator, you know that the scenarios are directed and that something is being tested, but in the real world, you never know when or what is going to happen (e.g. hazardous situations). Those situations are not desirable to test in a real car, but in the simulator, they can be introduced rather frequently. However, test drivers notice if hazardous situations become too frequent and are more careful than normal. This problem with inconsistency in human behavior is anticipated but not desirable when it comes to capturing normal behavior patterns. Hence, this is the problem of designing and conducting accurate and desirable observational scenarios.

Table 3

Behavior of the five drivers at the lights in the validation set, changing from green to red

	Light 1	Light 2	Light 4	Light 5	Light 6	Light 7
Driver A	R	R	S	R	R	S
Driver B	R	R	S	R	R	R
Driver C	S	R	S	S	S	S
Driver D	R	S	S	R	S	S
Driver E	R	S	S	S	S	S

S stands for stop and R for running the light while it's still yellow.

5.1.1. Selecting and Partitioning Training Data

The nature of learning by observation is to let the human subject perform his or her task as realistically as possible and to monitor and extract his or her true behavior. To examine the potential of learning by observation, no extra knowledge or information is added to the data. As an example, the agent will not be penalized extra during the learning process if it runs a red light. It should only compare its behavior to the human whose behavior was used to evolve it. The experimental test-bed was designed for the experts to drive a route combining city driving and rural driving. During these 30-minute drives, the expert did not experience two identical situations. Human behavior is not always consistent, and the human driver might react differently to similar situations. It might be tempting to present the same situation 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 and might not behave naturally. Conversely, letting the human act in a realistic environment with similar, but not identical, situations introduces disorder to the training data. In the worst case, contradictory data might exist in the data set. If learning by observation is to be rigorously implemented, this is an important issue. The learning conducted in this research left any disorderly data within the data set to investigate how well the CxBR and GP approach handles this.

The data used in these first experiments were restricted to city driving. During city driving, the most diverse and unpredictable behavior was noticed.

When it comes to observing tactical human behavior, the perspective of the behavior pattern is observed from outside the car. The car and the driver are looked upon as a single unit with some human behavior characteristics. The agents (i.e. simulated cars) to be automatically created are also only observable from the outside. The issue is not to model the human behavior in terms of emotions, stress, sickness and other reactions imposed from car driving that will not be observable from outside of the car.

When we observe the driver and the car as one unit, there are three actions that affect the car's behavior: steering, acceleration and braking. The first experiments conducted are limited to evolving basic city driving behavior. As long as the car is in city traffic, the steering is of minor importance since no differences in human behavior patterns are detectable with regards to steering. The driver always keeps the car close to the middle of the lane. Hence, the steering was not part of the learning task.

In this research, the filtering and partitioning of the data were made manually. The amount of data was large because the Driving Simulator sampled the environment and the action of the driver at a rate of 10 Hz. This results in a data set close to 15,000 samples as the driver completes the drive. To get a reasonable response time from the learning system, this data set was reduced to less than 300 data samples to constitute the training data for each agent. The training data used is described in *APPENDIX D: TRAINING DATA*. In addition, inputs that were of no interest for the learning paradigm, such as pitch, roll and steering wheel angle were also removed from the data set. The data used during training were throttle pedal pressure, break pedal pressure, speed

and distance to intersection or traffic light. Additionally, two Boolean variables were derived from the simulator's Hot Spot outputs. Those two variables indicate the presence of traffic lights or turning at intersections. Note that no preprocessing of the data was done in terms of computing the average behavior or other processing of the data in specific situations after data were selected.

The data were partitioned to contain similar amounts of data from the different scenarios to be used in training. This was done to ensure that the search pressure would not favor any particular situation. The data points were selected randomly from typical scenarios (e.g. within 100 meters before a traffic light or an intersection). The scenarios are further selected to complete the behavior in the context to be learned. As an example, if **Traffic-Light-Driving** is to be learned, data from several different scenarios (e.g. stopping at light turning red, running yellow lights, passing traffic lights when making an intersection turn, etc.) need to be included in the training data. In some cases, the results from learning indicated that the selection pressure favored some specific scenario. This was why the data set had to be adjusted. Data were also partitioned to fit the CxBR structure. If the training, for example, was designed to learn normal city driving, no data were used from traffic lights or intersections. Note that before the agent is fully developed, data from each scenario has to be presented to ensure that the agent can determine the right context used in all possible situations. The data points for each scenario selected need to be picked with a constant time frame (e.g. 0.4 seconds), so GenCL knows when to stop the micro simulation and compare the individual's performance with the human's performance.

5.2. Configuring the Context Base Prior to Learning

The frame of empty contexts, where GP evolves knowledge, is shown in Figure 20. This was the only data about the problem that was created manually during this research.

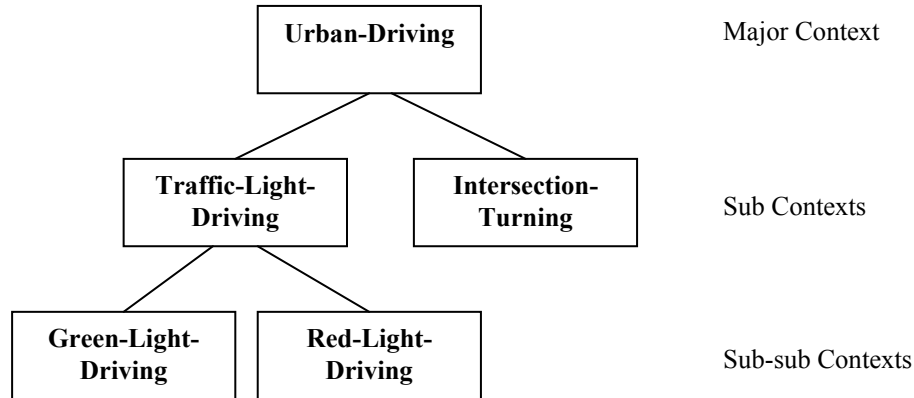


Figure 20: The context hierarchy of basic city driving.

If the model is to cover the basic city driving, it must be able to handle traffic lights and intersections besides normal driving on a straight road segment. The predefined structure for learning is described in Figure 20. The actions (i.e. the knowledge in the action rules) within the context **Urban-Driving**, the Sub-Contexts **Traffic-Light-Driving** and **Intersection-Turning** and the Sub-Sub-Contexts **Red-Light-Driving** and **Green-Light-Driving** are evolved by the GP algorithm. Additionally, the system evolves the rules controlling the activation of contexts (i.e. sentinel rules).

5.3. Experimental Test-Bed – The GenCL Artifact

The experimental test-bed consists of the GenCL artifact that works according to the GenCL algorithm described in section 4.7. The implementation and configuration of the GenCL artifact will be discussed in this section, especially the details of the GP module.

5.3.1. GP Implementation

Fuzzy Logic representation was not used in the experiments within this research. This was done because the integration was of primary interest and the function set was kept as simple as possible in this initial investigation. The use of Fuzzy Logic seems to have features advantageous to modeling human behavior and could be of interest for future research.

The source code representation in the GP individuals is chosen to be implemented in a tree structure, earlier described in 3.3.5. The knowledge evolved by the GP module in the GenCL artifact is syntactically correct C code. The code evolved by the GP module makes it easy to incorporate the knowledge evolved into the Traffic Simulator, developed from the CxBR framework originally implemented in C++ by Norlander [55].

In order to investigate the applicability of the GP to evolve knowledge in the context bases, different GP features were implemented in the GenCL artifact as various genetic operators and selection methods. The impact of adjusting these features will be investigated in 6.6. Two genetic operators were implemented: crossover and mutation. The crossover and the two types of

mutation were implemented in accordance to the description in section 3.3.5.

5.3.1.1. Population Initialization

When an individual in the GP population is created, an instruction tree is randomly generated. If the nodes are selected completely at random among both the function set and the terminal set (i.e., GROW initialization method), the risk is that many individuals will be rather small and only contain a few functions and operators or even only a single value or variable. If this is the case, the GP learning process has very little information upon which to base the evolution. One initialization method commonly used is called RAMPED initialization. Here the individual randomly picks either the initialization method GROW or FULL with equal probability. If the individual is initialized with the FULL method, the instruction tree is forced to grow to the maximum depth defined. The maximum depth needs to be defined so the instruction trees do not grow overly large during evolution. Using the RAMPED initialization method will then produce an initial population where approximately half of the individuals are of full depth and the other half are of different depth sizes. RAMPED initialization is used in this research for the initial creation of the individuals. Note that the GROW method is used when a new sub-tree is created during regular mutation.

5.3.1.2. Selection Methods

Four different selection methods were developed within the GP module: Fitness Proportionate, Ranked Fitness, Tournament Selection and Fitness Uniform Selection. These four different

selection methods were presented in section 3.3.5 and will put different selection pressure on the learning process. A higher selection pressure will favor exploitation of the interesting areas of the search space where individuals with high fitness reside. The selection method here with the highest selection pressure is Fitness Proportionate Selection. If one individual has an exceptionally good fitness value, the probability is that this individual will be chosen most frequently and the population will focus its search in this area. With Ranked Selection, the difference in fitness value is smoothed out when it comes to selection. Even if there is a great difference between the best and the second best individuals' fitness value, their probability of being selected is minor. The selection method with the weakest selection pressure is the Tournament Selection, since the group size often is much smaller than the population size. The Fitness Uniform Selection presents another type of search pressure. Here, the very best individual, like the worst individual, has little chance of being selected (see Figure 10). Rather, the selection pressure is somewhere in the middle of the population with one important exception. This selection method penalizes individuals grouped together with similar fitness values. If half of the population, for any reason, is grouped together around one value, all of the individuals in the group, except the two on the group border, will have a small probability of being selected. In this manner, this selection method has a built-in feature that withholds the diversity in the population.

How the selection pressure affects the learning in GP is highly problem dependent and the optimal configuration can only be gained by testing. The complete configuration of the GP during the experiments is further discussed in section 5.3.2.

5.3.1.3. The Model of a Car Model

Since the Driving Simulator was not accessible during the learning phase, a car model was developed. The configuration was set to represent the output from the individuals to describe the accelerator and brake pedal pressure. To be able to perform a comparison to the actual drivers' speed and position, the model of how pedal pressure affects the car's change in speed was developed. The implementation of the Driving Simulator's car was not accessible. The physics behind the model that connects the pedal pressure to the change in speed is a rather complicated system of differential equations. Furthermore, it is strongly dependent on the features of the actual car, such as air resistance, engine and transmission dynamics and wheel configuration. A model of a real car would probably be different from the model used in the Driving Simulator.

Instead of looking at a correct vehicle model, an alternative engineering approach was taken. A rough model was established by inspecting the recorded data. The rough model establishes a relation between the speed of the car and the pedal pressure. This model was then optimized with a simple GA. The basic GA is very easy to use for optimizing function features. The rough car model was placed in the fitness function. The fitness function compared the output from the model (i.e., speed) with the speed of the car the driver operated during the data collection drive. The average deviation was then the fitness value. The resulting model looks like this:

```

double calcSpeed(double acc, double prevSpeed, double deltaT)
{
    double tmpSpeed;

    tmpSpeed = (1.34319425020673/(acc+1.34319425020673))
                *63.435064487992*acc*deltaT+prevSpeed*0.97839448706081;
    if(prevSpeed > 15.0)
        tmpSpeed += 0.0934197333815892;
    if(prevSpeed > 58.6784578243764)
        tmpSpeed += 0.583733277998991;
    if(prevSpeed > 94.0581146329115)
        tmpSpeed += 0.63169388263119;

    if(tmpSpeed < 0)
        tmpSpeed = 0.0;

    return tmpSpeed;
}

```

The highlighted numbers are those optimized by the GA in the model.

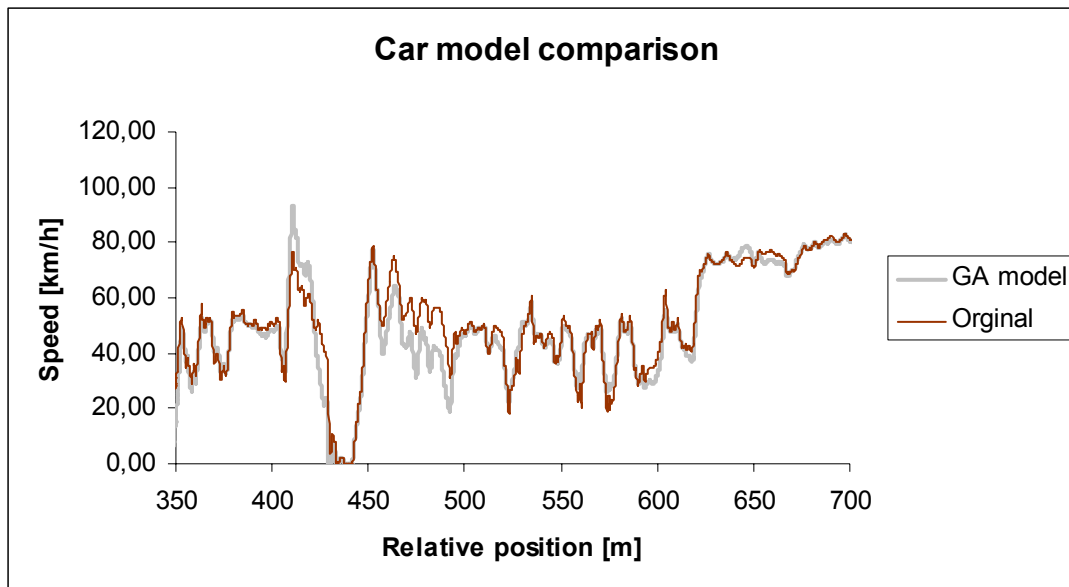


Figure 21: The model of a car model

The average deviation between the rough model optimized by the GA and the original data over the entire test run was 4.2 %. Figure 21 shows a piece of the calculated deviation. Even if the

deviation is fairly small, it might be a concern when designing the fitness function.

5.3.1.4. The Fitness Function – A Micro Simulation

To be able to compare the individual's performance with the observed human and to calculate the fitness value, the source code of the individual was executed. This is done in the Micro Simulator incorporated within the GenCL artifact. See Figure 15.

The Micro Simulator in GenCL was reduced to minimal operational requirements to save computational effort. Evolving parts of the simulated agents in the GP part and running those in a simulator is computationally expensive. The Micro Simulator was stripped down only to be operable in a restricted environment. All the pattern matching and data retrieval from different fact bases were removed. The context base was kept from the original CxBR framework described in Figure 3 and incorporated in the GenCL artifact together with a minimal simulator engine. This is enough for the agent's operational behavior to be compared to the human driver's recorded behavior.

Because of the time complexity of compiling, linking and loading the C code in the individuals, an interpreter was developed in the Micro Simulator module to interpret the C code. Each individual from the GP is fed to the Micro Simulator module to perform a micro-simulation from where the result can be compared to the human's performance. In this micro-simulation, the C code that represents an individual in the GP population is interpreted and run for a number of

simulation cycles. Eventually, when the complete agent is evolved, the source code will be placed in the CxBR framework, compiled and executed in the Traffic Simulator (i.e. the final use of the agent). This Traffic Simulator is used during evaluation of the agents.

The fitness value is the average combined speed, distance and throttle deviation. During the calculation of the fitness value, the Micro Simulator within the GenCL runs the individual for an appropriate number of time steps so that the speed, distance and throttle/brake pressure can be compared to the next training data sample. The Micro Simulator then continues to the next sample and new deviations are measured. This continues to cover all data samples within the current training sample set and the average deviations are then calculated as the fitness value. Figure 22 shows an example of one training scenario when an individual is compared to the actual driver's behavior approaching a traffic light turning red. At the start of the micro-simulation, the individual is initiated with the same position and speed as the real driver.

As the micro-simulation is running, the deviation between the individual's behavior and the driver's behavior is measured at defined time steps. In this example, the time steps are 0.4 seconds apart (i.e. each vertical line in the chart). Actually, the time steps for the actual driver are a set of data samples (18 in this example) to which the performance of the individual's micro-simulation is compared. In Figure 22, only the speed is given as an example, but the same comparison was also made with throttle/brake and distance.

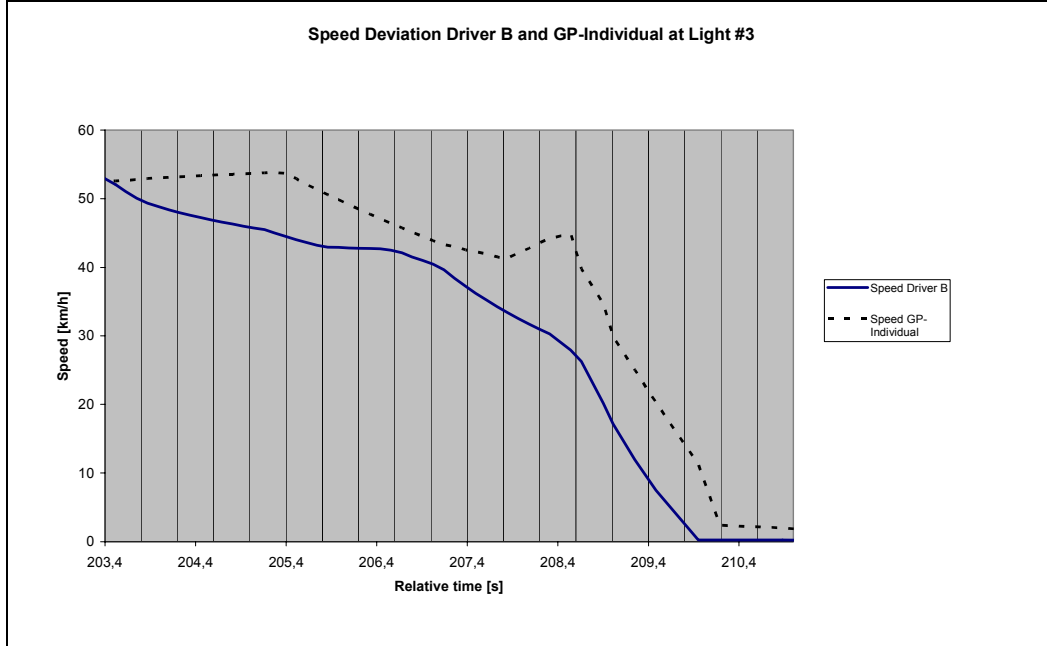


Figure 22: Comparison of individual's micro-simulation and Driver B

The fitness value is calculated as

$$f_i = \frac{1}{M} * \sum_{k=1}^M \left(\frac{\sum_{j=1}^{N_k} (|v_{d_j} - v_{i_j}| + |s_{d_j} - s_{i_j}| + |th_{d_j} - th_{i_j}|)}{3 * N_k} \right)$$

where v is the speed, s is the distance, th is the throttle/brake pressure, M is the number of scenarios presented (e.g. four traffic lights) and N_k is the number of samples used in scenario k . The subscript d refers to the real driver and i to the individual whose fitness is calculated. This will give an average measure of the combined speed, distance and throttle/brake pressure deviations. Note that this fitness value is a summation of units and therefore the result is unit-

less. As the speed part of the fitness function measures the momentary deviation, the distance serves to collect the accumulative deviation. This can be compared to the short and long term reward in reinforcement learning.

At the end of the experimental phase, throttle comparison was removed from the fitness value. There are two factors that made the throttle comparison a bad measure on the performance. First, a car model does not react on high-frequent changes in the throttle pressure. The car model works as a low-pass filter between the throttle and the resulting speed of the car. In other words, if the driver makes a lot of rapid small changes to the throttle it will not affect the speed of the car, but it might affect the fitness value. Another problem was that the code from the Driving Simulator car was not available and a model of the car was created as described in section 5.3.1.3. Even if this newly created model has a close resemblance to the Driving Simulator's car, it is not perfect and will be an error source if the throttle/brake pressure is part of the fitness value. When there is a discrepancy between the two car models it might bring in a negative correlation between speed and throttle/brake pressure in the fitness function. If the GP evolves individuals that lower the discrepancies in speed, it might worsen the discrepancies in pedal pressure.

5.3.2. GP Configuration

Before starting the evolutionary process, the GP needs to know the function set F and the

terminal set T . The following terminal and function sets were used during evolution:

- Variables/Input
 - `double distance, speed;`
 - `enum Light;`
 - `bool intersection, LightPresent;`
- Variables/Output
 - `double throttle;`
 - `bool activateContext;`
- Constants
 - `double [-100, 100]`
 - `bool [false, true]`
- Operators
 - `mathOperator: +, -, *, /`
 - `trigOperator: cos, sin, exp, log`
 - `polyOperator (degree 2 – 5): pow`
 - `compOperator (if–else statements): ==, !=, >, <`
- Functions (Sub-sub contexts)
 - `lightFunc: redLight(), greenLight()`

Not all of those functions and terminals were used at all times. Rather, these were varied during the experiments to determine whether any function set would suit a specific problem better than

another and to investigate the learning algorithm's sensitivity to different parameter settings.

After initial testing, some of the parameters of the GP were kept consistent during the experiments while others were varied. The settings used in the experiments are shown in Table 4.

The mutation rate was always set to 0.1 while the *mutation type rate* was varied. We define the mutation type rate as a probability of which type of mutation (i.e. sub-tree mutation or node mutation) will occur if an individual has been selected for mutation. The closer the value gets to 0.0, the higher probability that the mutation type is sub-tree mutation. If the value gets closer to 1.0, the higher the probability of a node-type mutation.

Table 4
GP configuration during the experiments

Initialization method	RAMPED
Population size	2000
Generations	2000
Crossover rate	0.55, 0.65, 0.75, 0.85, 0.95
Mutation rate	0.1
Mutation type rate	0.2, 0.5, 0.8, 1.0
Elitism	25
Selection method	Fitness Proportional, Ranked, Tournament, Fitness Uniform

A single value indicates that this configuration was used for all experiments while multiple values indicate a variation between the different experimental runs.

The population size and the number of generations were not varied during the different experiments. The *elitism* number refers to the number of best individuals preserved from one

generation to the next. This ensures that the best individuals are always kept in the population. Crossover rate and selection methods were varied during the experiments. Additional to the features described in this table, the function set was also varied during the different experiments. How these different configurations affect the outcome of the learning phase was investigated and the results are presented in section 6.6.

5.3.3. Evolving the Models

In accordance to Layered Learning GP (LLGP), the first modules to evolve were the **Red-Light-Driving** and the **Green-Light-Driving** Sub-sub-Contexts shown in Figure 20. The next step was to evolve the **Traffic-Light-Driving** Sub-Context. During the evolution of this Sub-Context, the two Sub-sub-Contexts were available as part of the function set that the GP could use. In this manner, the Sub-sub-Context was accessed as part of a competing context transition scheme [65]. The **Traffic-Light-Driving** Sub-Context evolves an action rule consisting of operators, functions and Sub-sub-Context calls. The activation of the Sub-sub-Contexts then becomes part of this action as the action rules of the Sub-Context evolve.

When the **Traffic-Light-Driving** context was completed, the action rules in **Intersection-Turning** and **Urban-Driving** were evolved by the GP. The final part was to evolve the sentinel rules of **Traffic-Light-Driving** and **Intersection-Turning**. Those sentinel rules were evolved according to the direct transition methodology. Those two sets of rules were evolved simultaneously using the Co-Evolutionary approach. The set of sentinel rules within the two

Sub-Contexts **Traffic-Light-Driving** and **Intersection-Turning** actually evolved after the Major Context **Urban-Driving**. This is a necessary adjustment (or the set of sentinel rules can be viewed as another dimension in the hierarchy) in order to evolve the most complex task of this training process. The sentinel rules for these two Sub-Contexts use the direct transition approach. Direct transition makes the learning somewhat more complex for the GP. Now, the two sets of sentinel rules are interdependent. If one of the contexts makes an activation query and the other already is activated, the active context needs to release the activation before the other context can be activated. Hence, the evolution of the two sets of rules needs to be done in parallel so the fitness is also penalized if it prohibits the other more appropriate context to be activated. Also, note that all the prior evolved contexts need to be part of the evolution of these two sentinel rule sets. In order to determine if the Major Context **Urban-Driving** should be active or if any one of its two Sub-Contexts should be activated, the complete functionality of all those three contexts need to be operable to be able to compare the action with the human's action. When a data sample is presented during training and the sentinel rules could activate one of the Sub-Contexts, then the action rules within this sub-context will be executed. If neither of the two sentinel rule sets is activated, the **Urban-Driving** context is executed. When the execution is done, the output is compared with the action performed by the real driver and a fitness value is calculated. When this is completed, the GenCL artifact has evolved an agent capable of handling basic city traffic, traffic lights and making turns in an intersection.

Intersection Turning - Sentinel rules

```
(lightPresent?  
  distance>35.319071?  
    (lightPresent?  
      (speed>35.319071?  
        (distance<98.083437?  
          (!intersection?  
            : (speed<distance?0:1))  
          : (!intersection?  
            (speed<98.083437?0:1)  
            : (speed>14.383373?  
              (speed>57.991272?  
                (!intersection?  
                  (!lightPresent?1:0)  
                  :0)  
                  : (!lightPresent?0:1))  
                :1)))  
            : (lightPresent?0  
              : (!intersection?0:1)))  
          : (speed>50.532548?0  
            : (distance>50.215155?0  
              : (intersection?1:0))))  
      : (lightPresent?0  
        : (!intersection?0:1)))  
    : (speed>50.532548?0  
      : (distance>50.215155?0  
        : (intersection?1:0)))));
```

Intersection Turning - Action rules

```
((sin(73.021637))<distance?  
  pow((distance<speed?  
    (pow(speed,5)>(33.744316>(distance<speed?speed:71.636097)?  
      pow(pow(distance,3),3)  
      : pow(59.166234,2))?  
    (distance>speed?(cos(-52.122562)):(sin(pow(distance,5))))  
    :13.791314)  
  :speed),2)  
: (log(72.826319)));
```

Figure 23: Example of evolved code – Intersection turning, agent C

An example of the evolved code is presented in Figure 23. The code is here presented on several lines and in a somewhat structured way but the code is untouched. All the evolved code is

presented in *APPENDIX E: EVOLVED CODE*.

5.4. Summary

This chapter presented the application scenarios used in the experiments. Furthermore, the collection of data and the configuration of the learning algorithm and its prerequisites were described. This will then constitute a sufficient basis to validate the new GenCL algorithm. The next chapter describes how these experiments were conducted and also presents the results and conclusions.

CHAPTER 6: RESULTS AND CONCLUSIONS

This chapter presents and discusses the results of the experimentation. The objective is to determine how well the learning strategy was able to learn and generalize the personalized human behavior. Note that all the behavior knowledge within the contexts is learned by the GenCL system.

6.1. Evaluation Criteria

A model's performance when compared with training data is by itself not sufficient to determine the success of the new approach. The key objective is to facilitate the creation of simulated agents with human behavior and to open up the possibility to capture implicit knowledge. The knowledge learned should not only mimic the behavior, it should also generalize the behavior and create reliable agents. Our evaluation of the GenCL approach will consider the following criteria:

- Learning capabilities
- Generalization
- Long-term Reliability

- Usefulness
- Ease of use

Learning capabilities experiments simply measure how well the system learns human performance. The model developed during training is simply compared with the training data collected from the human's driving performance (i.e., same comparison data is used as for the training). Learning capabilities experiments compare the model output with the human's action based on the observed data. Since the application of this research focuses on building autonomous agents able to operate in a simulated environment, this validation will not be sufficient. Even if these experiments show small deviations, the agent could, during operation, accumulate errors, so the agent might not work correctly as the simulation continues over an extended period. Except for the learning capabilities, all other validation experiments evaluate the agent continuously in a simulated environment, and make a long-term evaluation of its performance. The agent operates autonomously in the environment while presented with a number of situations that need to be acted upon, and its deviation from the expert's actions are monitored.

Generalization measures how well the agent can handle new situations not seen in the training data. The agent simply operates autonomously in the simulated environment and is compared to the recorded driver's behavior. The first generalization experiment was done with the training data set. Not all the of the training data set was used during training, so by letting the agent operate in the entire training data set environment, it will be exposed to partly new situations and

partly semi-known situations. Even if the agent here passes a scenario environment used during training, it will not be exposed to the same data as during training. Now the agent is approaching this scenario with a different speed and location since it is autonomous and has been in the environment for a while. Hence, we call those situations semi-known. The performances of the agents are compared with the driver's performance in these new situations to measure how well the agents were able to generalize the problem.

The second generalization test was carried out in a new environment. This new environment is the same in which the real drivers were exposed during the collection of the validation data set. Now the environment is different and the behaviors of the traffic lights are also different from the training data.

By letting the agent operate freely in the environment for an extensive period, the *Long-term Reliability* of the evolved model will be measured. The consistency of the agents' behavior is tested here.

To compare the *Usefulness* of the new approach, the model created of the CxBR and GP learning will be compared with models independently created in the traditional fashion by a knowledge engineer and a programmer interviewing and riding with the driver. If the results from the new approach are comparable to the results from agents developed with traditional means, this validates the feasibility of the new approach.

Finally, the *Ease of use* compares different settings of the GP-learning algorithm to evaluate how sensitive the learning algorithm is. This measure indicates how easy the learning algorithm is to use. If the learning algorithm needs much tweaking and fine-tuning, the knowledge engineer needs to be replaced with a machine learning expert and the workload of creating a simulated agent might remain the same.

6.2. Test of Learning Capabilities

When testing the learning capabilities, the objective is to get a measurement on how well the algorithm was able to conduct learning of human behavior. The fitness value is the measurement that tells the GP algorithm how well each individual performs against the human's performance (i.e. the fitness function). The first determination on how well the algorithm conducted the learning phase could be to look at the fitness values of the best individuals.

Table 5 shows the fitness values for each evolved agent in its different modules. Fitness for **Traffic-Light-Driving** is the combined fitness when the three contexts **Red-Light-Driving**, **Green-Light-Driving** and **Traffic-Light-Driving** are working together. This is simply a result of the CxBR structure, shown in Figure 20, where the **Traffic-Light-Driving** cannot yield a complete and accurate output without accessing its two Sub-Contexts. Similarly, the fitness values of the sentinel rules in **Traffic-Light-Driving** and **Intersection-Turning** are calculated when all the context parts are accessible. If an accurate output should be obtained during the micro-simulation, whether any of the two Sub-Contexts (**Traffic-Light-Driving** or **Intersection-**

Turning) are activated or if the Major Context (**Urban-Driving**) is activated, then all those contexts' actions need to be fully operational.

Table 5

Fitness values for the five evolved agents

	Urban Driving	Red Light	Green Light	Traffic Light	TLD sentinel rules	Inter-section	ID sentinel rules
Agent A	2.03	1.32	1.67	4.37	2.32	2.33	1.98
Agent B	3.22	1.23	1.29	2.15	3.04	1.58	3.55
Agent C	0.824	6.04	2.07	5.61	3.34	1.63	2.96
Agent D	1.18	5.40	3.18	2.24	2.93	1.17	2.44
Agent E	1.76	6.08	3.57	6.08	4.32	3.35	6.04

The fitness value is not a totally meaningful measure of learning effectiveness, but it is the key factor in the GP algorithm to distinguish a better individual from a worse one (i.e. low value is less deviation - better fitness value). It gives us the combined average deviation, but it could be that one of the compared variables (i.e. speed, distance or throttle/brake pressure) could deviate significantly while the other two are good. The values for the sentinel rules, which together constitute the performance measure of the whole agent, in Table 5 indicate that agent E is performing worse than the other agents. Actually, the values in different rows and columns could not be fairly compared with each other since different data sets have been used. Different data are used when, for example, **Urban-Driving** and **Intersection-Turning** is learned. The speed range in the data used for **Urban-Driving** could be only 10 km/h but for **Intersection-Turning** the range could be 65 km/h. Therefore, a value of 1.18 might not necessarily yield better

performance than 2.93. In the same manner, different data sets are used when training different agents and their respective measures cannot be fairly compared. The only thing that matters to the learning algorithm is when training is conducted with a specific data set, lower values indicate better performance.

To get a measure on how well the agent learns its tasks, data used in evolving the two sentinel rule sets (i.e. sentinel rule sets of **Traffic-Light-Driving** and **Intersection-Turning**) was fed into the complete model and compared with the outputs of the corresponding driver. Since the evolution of these two sentinel rule sets involve all other contexts, this data set could be used as a measure on the learning capabilities of the complete agent. Table 6 shows how well the agents learned their tasks.

Table 6

Learning capabilities

	Speed deviation		Speed
	RMS [km/h]	RMS [%]	Correlation
Driver A/Agent A	2.98	4.88%	0.988
Driver B/Agent B	3.68	6.41%	0.983
Driver C/Agent C	2.57	4.74%	0.990
Driver D/Agent D	2.41	4.19%	0.989
Driver E/Agent E	8.10	13.2%	0.852

The comparison made here is simply obtained by feeding the inputs experienced by the real driver to the agent. Their deviation is then measured at the time of the next training sample. A few simulation steps are performed (less than one second) to be able to make the comparison at

the next data sample. The initial inputs for each micro-simulation are not collected as the agent is acting in the simulated environment. Instead, the inputs come from inputs that the actual driver experienced during his run. Hence, no real simulation is performed and the deviation in position is minimal. Since only the outputs from the agent are compared to the driver's performance at a specific location and the result of the car is of interest (not really the pedal pressure), the only sound measurement is speed. The speed deviation in Table 6 is calculated with the same formula as the fitness function, except that the distance and throttle/break pressure was not used. The correlation coefficient in Table 6 is calculated as

$$\rho_{V_d, V_a} = \frac{Cov(V_d, V_a)}{\sigma_{V_d} * \sigma_{V_a}} ; -1 \leq \rho_{V_d, V_a} \leq 1 ; Cov(V_d, V_a) = \frac{1}{n} \sum_{j=1}^n (v_{d_j} - \mu_{v_d})(v_{a_j} - \mu_{v_a})$$

where μ is the mean value and σ is the standard deviation. If the correlation coefficient is close to 1.0, there is a high correlation between the two data series. If it is close to zero, there is little correlation. If it is close to -1.0, the correlation is inverted.

We can see in Table 6 that the agents have small deviations and high correlation to their respective driver. Agent E shows a slightly worse performance, where its deviation is higher than that of the other agents and the correlation is also a bit worse. As discussed earlier, looking at the data collection, the behavior of driver E in the second simulator run has little resemblance to his first run. Further investigation of driver E's behavior shows a slight inconsistency within the training set. Light 2 (see Table 1) is located 30 meters 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 in the environment and he did not spot 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 if his approaching speed is higher. Note that the training data set is missing a lot of information about the environment, as buildings and other objects besides the road.

The results from the test of learning capabilities present low discrepancies between the agents and their respective driver. The speed deviation is low and the correlation is high. A high correlation, close to one, means that the agent modifies its speed in the same manner as the real driver. Hence, GenCL show strong learning capabilities.

6.3. Test of Generalization

Generalization takes place when an agent trained with one specific set of data can extend its correct behavior to handle similar but different situations correctly. To measure generalization, the agent operates in the simulated environment within the Traffic Simulator and compared with the recorded driver's behavior in new situations.

As an autonomous agent is being evolved, it is necessary to test its ability to perform in its normal environment (i.e. running in a simulation). It is not sufficient to only test input vectors

and compare them with the anticipated output vectors since this will not tell us anything about the agents' accumulated errors and long-term reliability. We need to ensure that the agent is autonomous and that the agents' behavior in this simulated environment actually is comparable to the drivers' behavior, even after minutes, hours and days. Except for the Learning Capabilities evaluation, previously presented in section 6.1., the results are gathered when the agent operates autonomously within the simulated environment. At each simulation cycle, the behavior of the agent is compared to the behavior of the driver at the same position (e.g. 35 meters prior to a traffic light). The deviation in behavior could primarily be measured in speed and time deviations. Since the comparison is made at specific locations, it would have taken the agent and the driver some time to get there from the start of the simulation. Hence, there will always be a time deviation between the agent and the driver at a certain location in the simulator environment. The time deviation is harmless to the agent's behavior, but it gives a measure on the agent's long-term deviation. The speed deviation measures how the agent's behavior deviates from the real driver's behavior at every specific moment. Even if this deviation is small, it could accumulate over time (e.g. if the agent always is slightly slower than the driver is) but it will then show in the time deviation.

When comparing the agent and the driver at specific locations, a problem occurs if the speed differs greatly between the agent and the driver at one location. The problem is that in such a case the comparison could be made with the same sample more than once (see Figure 24). If the agent is moving slowly and the real driver is moving fast at some part, the closest recorded driver's sample will be the same sample for several of the agent's samples, since the sample rate

is 10 Hz for both the driver data and the agent. This is not sound when doing a statistical analysis of the data, as when computing the correlation coefficient. Hence, all those occurrences of duplicate sample comparisons were removed from the comparison data. In the example in Figure 24, only two of the five agents' samples can be used, since otherwise the same driver's sample needs to be used several times for comparison.

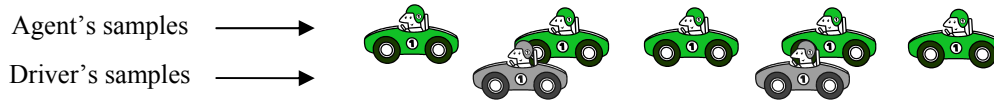


Figure 24: Comparison at closest position with same sample rate but different speed

6.3.1. Generalization in the Training Environment

Less than 300 samples of the approximately 5,000 samples collected in city driving were used during the learning phase of the agents. Hence, not all of the data or the scenarios in the training data set were used during training, so the first generalization test was done on the complete city training data set. Generalization measures how well the agent can handle new but similar situations. Nevertheless, even if some of the data used during training were collected from the same parts that the agent now experiences during generalization testing, the agent will never experience the same input pattern as it did during learning. This is because the agent is now autonomous and by its action will generate its own input state for the next simulator cycle.

Figure 25 shows the gray areas of city driving where training data samples were used during training of the agent. Note that not even all the samples within the sections were used. We can

further see that the evolved agent almost never has the same speed as the driver at a certain position in those segments.

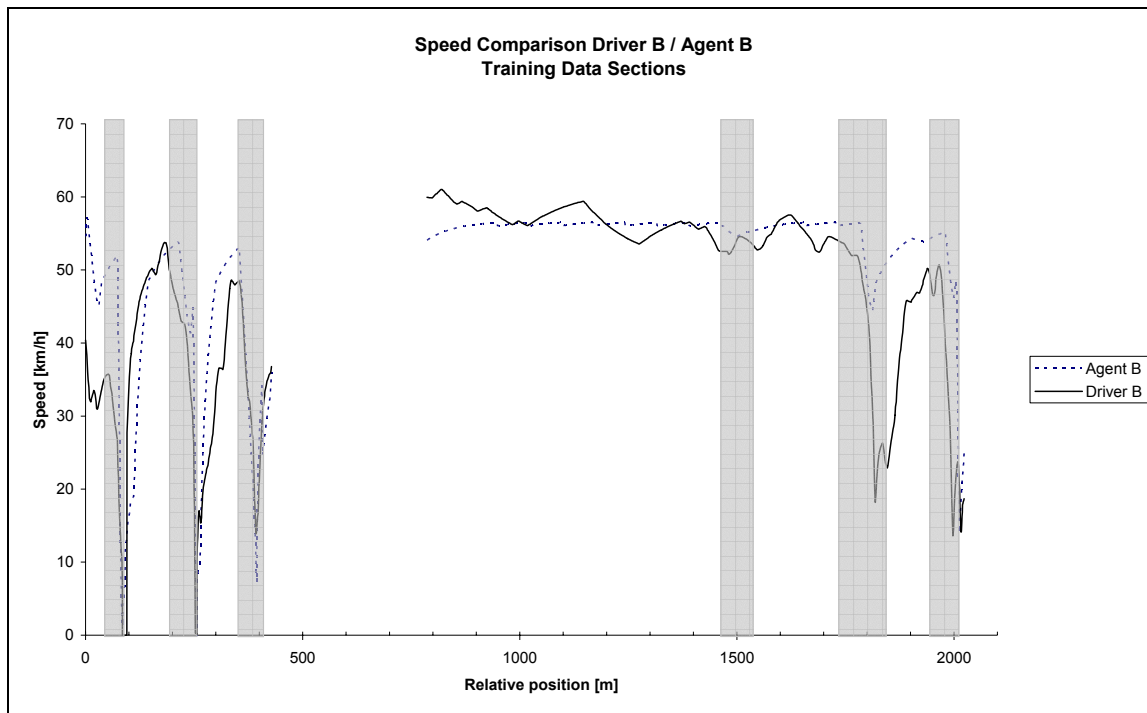


Figure 25: Behavior of agent B and Person B in the training environment

There is a gap in the graph because the agent is taking an alternative route and not completely following the driver's route. This is because the real driver is leaving the city and entering rural traffic driving that is not implemented in the agents. Therefore, no comparisons are made in those locations where they are taking different routes.

In this generalization run, the agent will also experience at least four completely new traffic

lights and three intersection turns never exposed during training. Yet, the data used in the comparison are from the training data set. Each agent will run the same path, which will take between 140 to 170 seconds to complete depending on the agent's behavior. This equals 1,400 to 1,700 data samples to be compared to the drivers' behavior. Remember that the agent was only trained with less than 300 data samples.

Table 7

Qualitative validation of the agents in the training environment

	Light 2	Light 3	Light 4	Light 6	Light 7	Light 8	Light 3b	Light 4b
Driver A/Agent A	S/S	R*/R	OK	R/R	R/R	OK	OK	OK
Driver B/Agent B	S/S	S/S	OK	R/R	R/R	OK	OK	OK
Driver C/Agent C	S/S	S/S	OK	S/S	S/S	OK	OK	OK
Driver D/Agent D	S/S	S/S	OK	R/R	R/R	OK	OK	OK
Driver E/Agent E	R/R	S/S	OK	R/R	R/R	OK	OK	OK

* After the simulation run, driver A explained that he became stressed and by accident ran the red light. This was not his normal behavior and this particular incident was regarded as an outlier and removed from the training set.

Table 7 shows a qualitative comparison regarding the traffic lights that the agents pass. Lights 2, 3, 6 and 7 change from green to red. S stands for stop and R for running the light while it's still yellow. Lights 4, 8 and 4b change from red to green, and OK here means that the agent performs in accordance to the driver (i.e., slows down when the light is still red and picks up speed when it turns green). Light 3b is constantly green, and OK refers to the agent performing in accordance to the driver (i.e. almost no noticeable change in driving behavior).

Here we can see that the agents act in harmony with the human drivers. The only question is

agent A at traffic light 3. Here the agent actually runs the yellow light. Driver A ran this light while red but in his statements after his simulator run, he stated that it was not according to his usual behavior. Therefore, this traffic light occurrence was regarded as an outlier and removed from the training set.

As an example on the comparison between an agent and a driver, Figure 25 shows the speed deviation of agent B and driver B at different positions in the training environment. The similarity in behavior is shown in Figure 25, where the agent and the driver slow down and stop at the same positions in the environment. Even if it is not visible in the figure, those occasions are traffic lights and intersection turning.

By looking at the graph, we can see that the agent presents a behavior pattern that is clearly comparable with its corresponding driver. The notable differences are that the agent seems to filter out some of the fluctuations and presents a slightly smoother driving style. This is quite notable in most of the agent/driver comparisons.

The results representing the differences between the agents and their corresponding driver are presented in Table 8. The distance covered during the comparison is 1.6 km, and it takes approximately three minutes to complete. The average deviation between an agent and a driver is probably close to zero since some of the deviation will be positive and some negative. Therefore, a better measurement of the deviation is the Root Mean Squared error (*RMS*) where the sign of

the error, e , is irrelevant.

$$RMS = \sqrt{\frac{1}{n} \sum_{j=1}^n e_j^2} \quad \sigma_e = \sqrt{\frac{1}{n} \sum_{j=1}^n (e_j - \mu)^2}$$

Note that the standard deviation (σ_e) is a measure of the variability of the error and is based on the population's variation from the mean and not from the *RMS*. Hence, the standard deviation is a variability measure regarding both the positive and negative deviations.

Table 8

Relationship between the agents and the drivers in the training set environment.

	<u>Speed deviation [km/h]</u>		<u>Time deviation [s]</u>		Speed Correlation
	RMS	Std.Dev.	RMS	Std.Dev.	
Agent A vs. Driver A	8.09	7.35	5.81	4.11	0.825
Agent B vs. Driver B	8.32	7.92	3.13	2.78	0.893
Agent C vs. Driver C	6.74	6.72	2.10	2.06	0.920
Agent D vs. Driver D	8.46	8.45	3.13	3.12	0.842
Agent E vs. Driver E	9.29	8.42	4.49	3.72	0.783

The speed deviations and the speed correlation shown above can be compared to those presented earlier in the learning capabilities (Table 6). The results in Table 8 are not as good as those of table 6, but that's also expected since the agent now is exposed to new data not used during training. The agents also act autonomously. The results show low deviation and the correlation is still high. The exception, however, is agent E, which shows a slightly worse performance than the other agents.

6.3.2. Generalization in the Validation Environment

The validation environment refers to the new environment the drivers experienced during their second simulator run. When doing comparisons in the validation environment, the agents experienced the same city environment as the drivers did in their second simulator run. The agents will now experience totally new situations with a slight difference in environment behavior. The traffic lights will change their states at various distances as described in Table 2.

The total distance where deviation could be measured between the agents and the drivers is approximately 2.2 km and it takes the agent 3 - 3½ minutes to cover that distance depending on their behavior patterns. During the drive, the agent will pass five traffic lights that change from green to red and one that changes from red to green.

First, the qualitative behavior at the different traffic lights was examined. Table 9 shows the agents' behavior at the traffic lights. Here it shows that agent A runs light 7 while driver A stops at that light. Agents B and C perform exactly as drivers B and C, respectively. Agent D, however, runs both light 6 and light 7, but driver D actually stops at those two lights. If we look at agent E and driver E, we can see that the behavior often differs at the lights. As discussed in section 5.1, driver E actually performs differently during the two simulator runs. In the first run, he was reckless and ran more yellow lights than the other drivers. On the other hand, while in the validation environment, he was very careful and stopped at almost every light.

Table 9

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 the light turning green.

If we look at the speed and time deviations of the validation run in Table 10, the A, B and C agents perform well. However, a slightly worse performance can be observed from agent D.

Table 10

Speed and time deviation during the validation testing

	Speed deviation [km/h]		Time deviation [s]		Speed Correlation
	RMS	Std.Dev.	RMS	Std.Dev.	
Agent A	7.47	7.44	1.47	1.47	0.880 (0.924)
Agent B	7.14	6.19	2.56	1.75	0.896
Agent C	7.12	7.11	3.60	2.80	0.926
Agent D	10.5	9.23	9.10	6.78	0.712 (0.860)
Agent E	17.0	12.0	38.4	30.3	0.550 (0.664)

The values within the parenthesis represent the correlation when the occurrences are removed where the agent's traffic light behavior differs from the corresponding driver's behavior.

Agent E, on the other hand, is not performing well at all. That agent E is not performing well in comparison to driver E is easy explained by the irregular behavior of driver E between the original training run and the subsequent simulation run. When the stopping behavior at a traffic light differs between the driver and the agent, the deviations will be affected rather dramatically

after the light, since the time for the light to turn green will be added to the time deviation. Speed deviation will also be affected since differences at some of the samples will be high. The agents' behavior, when the lights where they misbehaved were removed from the comparison, is shown within the parenthesis. This is done as an indication of how much the lights affected the comparison.

The results for agent A are good even when its misbehavior at one of the traffic lights is kept in the comparison. The misbehavior of agent D is more interesting to analyze since agent D's behavior is not good. The one thing that differs from the drivers' first and second simulator run is the traffic light behavior. In their first run, all the lights changed when the driver was 30 meters ahead of the light. This was done to trigger unpredictable driver behavior. In the second run, the different lights had trigger distances listed in Table 2. One question to be asked is whether the constant behavior of the lights in the training data set would imply lack of richness in the training data set.

6.3.2.1. Richness Analysis of Training Data

To more thoroughly investigate the misbehavior of agent D and the relation to the possible lack of richness in the training data set, agents B and D were presented to different light scenarios and their behavior was recorded. Since the knowledge learned is transparent, the evolved source code was also investigated to better investigate agent D's misbehavior.

Table 11

Behavior of drivers B and E at the traffic lights changing from green to red

	Light 2	Light 3	Light 5	Light 6	Light 7	Light 9
Driver B	S	S	S	R	R	S
Driver D	S	S	S	R	R	S

S stands for stop and R for running the light while it's still yellow

Agent D performs poorly in the validation scenarios when the validation data set is used. To evaluate the performance of agent D further, the performance of agent B is also investigated here. Agent B is of interest since it performs well during validation, and driver B and D had the same qualitative behavior in the training scenarios (i.e. they stopped and ran the same lights) (See Table 11).

To investigate the agents' performance, they were presented with two different traffic lights: one where they are going straight and one where they are about to make a turn. Furthermore, each light changes its state when the agent is at six different distances to the light: 100, 50, 40, 30, 20 and 10 meters from to the light. At these distances, the light goes from green to yellow; three seconds later, they turn red. Note that there are six runs shown in each diagram in Figure 26 and Figure 27, but many of them are overlaid because the behavior of the agent is the same at certain times, even if the light changes at difference distances.

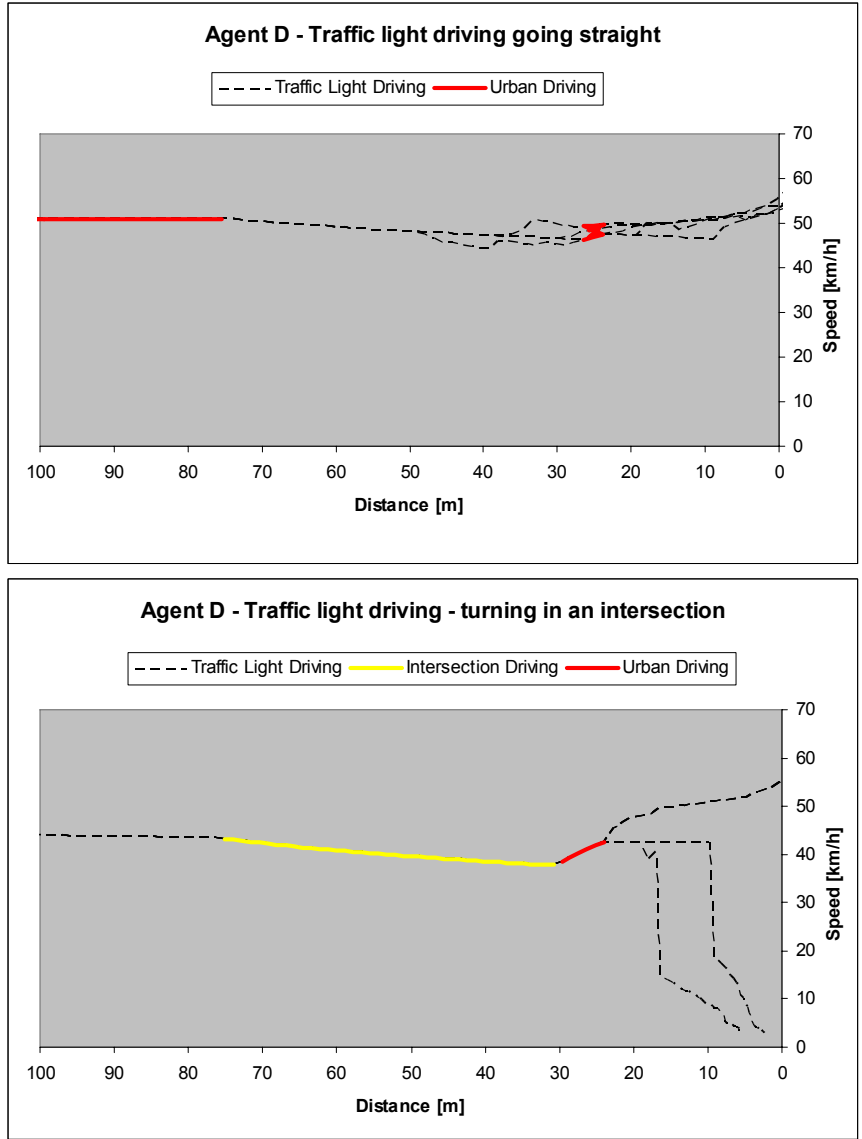


Figure 26: Agent D’s behavior at traffic lights

Black dotted lines indicate that **Traffic-Light-Driving** context is active, red lines that **Urban-Driving** is active and yellow that **Intersection-Turning** is active. The diagrams do not show how the Sub-sub-Context’s **Green-Light-Driving** and **Red-Light-Driving** are accessed. When

Traffic-Light-Driving is accessed, it will in some manner access its Sub-sub-Contexts.

Figure 26 shows the performance of agent D. When turning at an intersection, **Traffic-Light-Driving** activates and decreases the speed until agent D is closer than 76.5 meters to the light. Then, the **Intersection-Turning** context takes over and lowers the speed even more, which after a short time of **Urban-Driving** will revert to **Traffic-Light-Driving** again. (A Sub-Context needs to release the activation to enable another Sub-Context at the same level to be activated. Hence, the Major Context will temporarily get activated.) Now, the agent will come to a stop at the light, most of the time. When agent D is going straight, the **Traffic-Light-Driving** will not be activated before 76.5 meters prior to the light. The **Traffic-Light-Driving** will not lower the speed enough to make a stop in time. In *APPENDIX B: SENTINEL RULES OF AGENT B AND AGENT D*, the evolved sentinel rules that activate **Traffic-Light-Driving** for agent B and D are presented. The code there is presented in a structured manner and has been manually cleaned of unnecessary code. The code generated by GenCL is on a single line, but to better understand the code, it was restructured. Branches of code that could never be reached (because of conflicting conditions) were also removed. The code shows that if agent D has spotted a traffic light and plans to make a turn, the **Traffic-Light-Driving** context immediately activates. If agent D is going straight, nothing is taken under consideration unless agent D is closer than 76.5 meters.

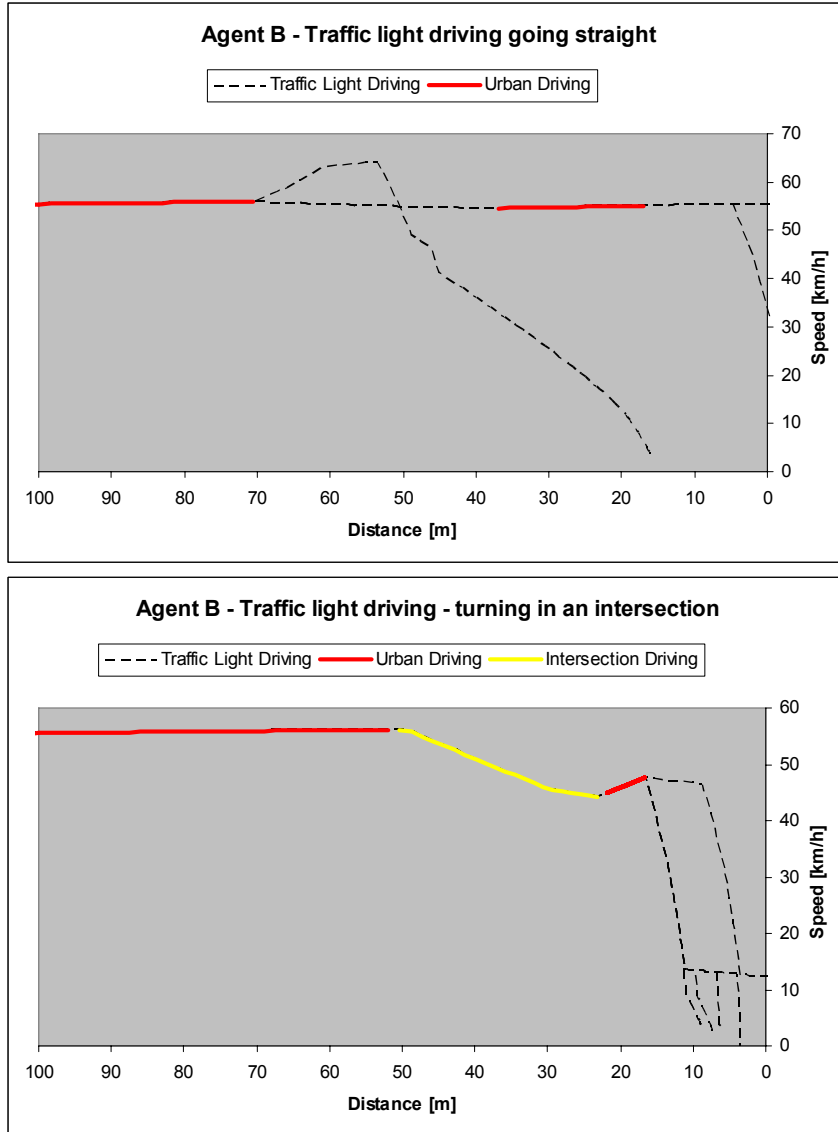


Figure 27: Agent B's behavior at traffic lights

Figure 27 shows agent B's behavior. When agent B is going straight, the **Traffic-Light-Driving** context activates at 70.6 meters prior to the light, and if the distance is far enough from the light when it turns yellow, agent B will make a stop. When making a turn, the **Intersection-Turning**

context activates at approximately 45 meters prior to the light and the **Traffic-Light-Driving** is not activated until agent B is closer than 17.4 meters as shown in the source code in *APPENDIX B: SENTINEL RULES OF AGENT B AND AGENT D*. This makes room for the **Intersection-Turning** context to be activated, which reduces the speed to a level where the stopping rate of the agent significantly increases. Observe that the sentinel rules for **Intersection-Turning** are not presented in *APPENDIX B: SENTINEL RULES OF AGENT B AND AGENT D* since the misbehavior is related to traffic light driving.

The interesting result of this analysis is that the poorly behaving agent D doesn't trigger on the obvious 30 meter mark. Rather, the action triggers on whether agent D is going to make a turn or not. If we compare this to the drivers' behavior in Table 11, we see that the drivers stopped at all lights turning red except light 6 and 7. Their driving path was so laid out that the drivers were making a turn at all lights except light 6 and 7. Hence, there is a correlation between stopping at the traffic light and whether the driver is making a turn or not. This shows that there is a lack of richness in the training set. If the training set were equipped with a light where the driver stopped but did not make a turn, it would have been more complete.

This analysis actually shows that there was a lack of richness in the training data set, but the lack of richness was not related to the constant distance that triggered the light change as expected. It shows that agent D actually learned that the driver stops at a traffic light if he is going to make a turn at the light. If not, he actually runs all the lights as shown in *APPENDIX B: SENTINEL RULES OF AGENT B AND AGENT D*. The GP found this relation and learned it. Note that there

are probably other relations in the training sets, such as approaching speed at different distances to the light. These other relations could have been the reason why agent B manages to avoid the relation to turning at the intersection and generalizes better than agent D. The stochastic nature of GP might also influence which correlations in the data set that the learning mechanism will trigger. Also note that the problem addressed here is the lack of richness in training data not how to cope with messy training data.

If the knowledge about the lack of richness in the training data had been present at the initial stage of the training phase, the most reasonable conclusion would be to partition the two data sets and create a new training set partially from the original training set and partially from the original validation set. The remaining two data sub-sets would then be merged to constitute the validation set. This would have overcome this richness problem since there existed occurrences in the validation set where the driver stopped at a light even if he would not turn in the intersection. Another problem present was the fact that the two data sets were collected four months apart. If both sets had been available at the beginning of the training phase, the chance of discovering this problem might have been greater.

6.3.2.2. Capturing Individual Behavior

The correlation between the agents' speed and the drivers' speed could provide another quantitative measure of the agents' performance. By comparing the correlation between all the drivers and agents in the validation environment, we can investigate whether the agents have

been able to capture driver-specific features. Table 12 shows the correlation between the drivers and the agents. Note that the table is not symmetric because each agent operates in the environment, and comparative data depends on its behavior. This means that the same data is not used when agent x is compared to agent y as when agent y is compared to agent x. If the coefficient is close to one, the correlation between the variables is high. Conversely, if the value is low there is a low correlation between the variables. A specific row in Table 12 shows the correlation between a specific agent and the five different drivers' data. The correlation is calculated over the entire test run and is based on a sample set between 1,500 – 2,300 samples.

Agents A, B and C show the best correlation to the correct driver, while agent D and E correlates better to other drivers. Some agents misbehave at some traffic lights and the correlation coefficients inside the parenthesis shows the coefficient value when those occasions are removed from the validation data set.

Table 12

Speed correlation between the agents and the drivers

	Driver A	Driver B	Driver C	Driver D	Driver E
Agent A	0.879 (0.924)	0.840	0.831	0.708	0.667
Agent B	0.819	0.896	0.711	0.690	0.540
Agent C	0.853	0.644	0.926	0.857	0.913
Agent D	0.859	0.853	0.694	0.717 (0.860)	0.602
Agent E	0.794	0.855	0.738	0.675	0.550 (0.664)

The values within the parenthesis represent the correlation when the occurrences are removed where the agent's traffic light behavior differs from the corresponding driver's behavior.

Driver C is the driver who shows a careful driving behavior both in the training set and in the validation set and that reflects in the behavior of agent C. The correlation between agent C and driver C is high while the correlation to the other drivers is substantially lower. Another interesting observation is that agent A correlates better with driver A than driver B, even when the agent misses the last light and seems to be more qualitatively consistent with driver B in Table 9. This might indicate that the agent actually learned a more detailed behavior pattern than the qualitative measure shown in Table 9 suggests.

To validate the results of Table 12, it is necessary to know how similar the training data set and the validation data set were. If two of the training sets are similar, the corresponding agent would probably also be very similar in its behavior. If two of the validation sets are similar, the same agent should get the similar correlation values compared to those two drivers. The correlation between the different training sets is shown in Table 13. The correlation that is regarded rather high is marked with a bold red color.

Table 13

Correlation between the different training sets

	A	B	C	D	E
A	1	0.801	0.773	0.823	0.692
B	0.792	1	0.869	0.882	0.906
C	0.776	0.870	1	0.796	0.836
D	0.825	0.881	0.797	1	0.772
E	0.635	0.916	0.803	0.777	1

In Table 14, the correlation between the different validation sets are shown and the values with a high correlation are bold red. Table 13 and Table 14 are not symmetric since duplicate data are removed from the data sets to avoid producing a weighted correlation. In the same manner as when an agent is compared to a driver and duplicate data can appear, the same thing might appear when two drivers are compared. Not exactly the same data might have automatically been removed since it will depend on which data is the independent data (i.e. which car is the base for the comparison).

Table 14

Correlation between the different validation sets

	A	B	C	D	E
A	1	0.776	0.795	0.813	0.712
B	0.776	1	0.662	0.723	0.652
C	0.794	0.661	1	0.789	0.775
D	0.814	0.723	0.784	1	0.800
E	0.710	0.650	0.768	0.800	1

There is a significantly high correlation between some drivers in the training set and in the validation set. This means that the different drivers' behavior did not differ significantly. The result in Table 12 becomes even more interesting when the poor performance of agent E (inconsistency of driver E) and agent D (lack of richness in training data) could be explained. All other agents are able to capture the individual behavior of their corresponding driver. Agent A correlates best with driver A even if its qualitative traffic light driving better suits driver B in Table 9. Notable is that three of the agents were able to find the small variations in the

drivers' behavior and learn those so they will best match their behavior with their corresponding driver.

In conclusion, the agents have shown their ability to generalize behavior, and are able to conform to new but similar situations. Three of the agents have also shown ability to learn personalized behavior patterns even if the training data and validation data exhibit rather strong correlation between different drivers. Explanations to the misbehavior of the other two agents can be found with the lack of richness in the training data and inconsistent behavior of driver E.

6.4. Long-term Reliability Test

The long-term reliability test was conducted to investigate whether the agents exhibit consistent behavior even after a substantial amount of time in a simulation run. Given that GP will produce code not accessed during the training phase (i.e. non-coding regions), and that the function set within GP contains conditional statements that introduce discontinuities, the test of long-term reliability is important.

Here the five agents were allowed to operate within the simulated environment for 40 minutes, passing more than 60 traffic lights and 25 intersections. Now the agents were exposed to a variety of traffic light scenarios where, each one different from the other. To be able to compare their stability and long-term reliability, their behavior was recorded when the traffic light ahead of them was either yellow or red, since this is one of the occurrences where different behavior

can be detected. The logging function (i.e. recording of the agent's behavior) starts when the agent is closer to the light than 100 meters and continues 100 meters after the light (if the light is either yellow or red).

If an agent was not stable and invoked **Intersection-Turning** when making a turn, the agent will approach the turn too fast and will actually end up beside the road. If this were the case, 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 minutes proves robustness in terms of **Intersection-Turning**.

The data from this evaluation test is far too extensive to completely present here, but an example is shown in Figure 28. All data from this evaluation is presented in *APPENDIX A: LONG-TERM RELIABILITY*.

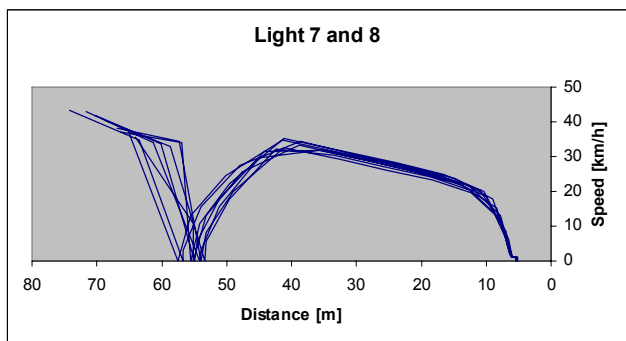


Figure 28: Behavior of agent C when repeatedly approaching traffic lights 7 and 8

Figure 28 shows the behavior of agent C as it approaches traffic light 7 and light 8 where light 7 is located 50 meters ahead of light 8. The lights turn from green to red at different timings. We can see that even if the light changes at different times, the agent stops at nearly the same spot each time. As the lights turn green, the agent continues to light 8 that turns yellow and eventually red as the agent approaches the light.

Since the traffic lights now change their states at different distances (i.e. the lights are time scheduled and not related to the agents' distance) and agents might approach the lights at different speeds, it is difficult to make an exhausting statistical analysis of their behavior.

Regardless, Table 15 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.

Table 15

Agents' Long-term behavior

	Light turning Red			Light turning Green
	Stopping	Avg.Dist	Std.Dev	Correct behavior
Agent A	20/20	34.7	12.9	20/20
Agent B	22/22	8.04	1.95	22/22
Agent C	25/25	5.89	1.03	8/8
Agent D	31/34	4.50	1.31	6/6
Agent E	22/22	13.5	0.551	11/11

A qualitative measure could be performed of the agents' action when the lights turn red. The stopping column in Table 15 shows how many lights the agents stop at, compared to the total

number of lights passed turning red. All the agents, except agent D, stop at all lights turning red. Agent D runs three lights when they turn red late (i.e. the light actually turns red before the agent passes through the light). Investigating the results more, it shows that if the lights turn red when the agent is further away than 27 meters, the agent will stop. On the occasions where the lights turn red when the agent is closer than 23 meters, the agent will run it. Even if the agent occasionally runs the light, it is consistent and acts the same in similar situations.

As the agents come to a stop at the red lights, a comparison could be made on their different stopping distances. Table 15 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. The diagrams in *APPENDIX A: LONG-TERM RELIABILITY* show that agent A stops at different distances almost every time. The surprising fact is actually that the other four agents manage to generalize so well that they stop at approximately the same distance, even if the time of light change is different. Remember that in the data presented to the agents during learning, lights changed their state when the driver was 30 meters prior to the light. Hence, all the agents stopped consistently at the same distance during training (approximately thirty meters after the light turns from green to yellow). Therefore, the most obvious action is not for the agents to generalize as well as they have, but rather to stop approximately 30 meters after the light changes from green to yellow. One remark can be stated about agent D's behavior at light 8: one behavior outlier occurs when the agent makes a very quick stop at 22 meter and then slowly continues towards the red light before picking up speed after the light turns green.

The final observations on the agents' long-term behavior are their behavior when approaching traffic lights turning green. Two observations can be made as the agents approaching a red light about to turn green. The first thing that institutes correct behavior of the agents is that they do not stop at the red light when they are far from the light. The other behavior to investigate 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 15 compares the number of correct behaviors to the total numbers of lights turning green exposed to each agent. All the agents show a correct behavior all the time as they approach a red light about to turn green.

This test has shown that the agents show consistent and stable performance throughout the long-term stability test. Four of the five agents even perform more consistent traffic light driving than could be expected regarding the training data presented during the learning phase.

6.5. Test of Usefulness

In order to determine how useful the automatic creation of simulated agents through GenCL is, two agents were developed by an independent source in the traditional way [9]. Here, a knowledge engineer interviewed and rode an automobile with two drivers acting as SMEs. The two SMEs were drivers C and D that earlier had been driving the simulator runs resulting in C's and D's training and validation data sets. The driving part of the knowledge acquisition lasted for approximately 45 minutes. 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 a bit more complicated than in a simulator. The simulator environment can be configured to focus on specific and interesting events, while in the real world one can try to act so those events might occur but it might be difficult to achieve.

The knowledge engineer knew that his model 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 prerequisites for the knowledge engineer were the same as for GenCL (i.e. the same empty context structure). The task was for the knowledge engineer to collect knowledge through interviews and by observing the SMEs driving a real car. The knowledge engineer 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 implemented to run in the same CxBR framework as the agents developed by GenCL. Note that an independent researcher did the knowledge engineering without any influence from the ones developing GenCL. Now, the two different approaches to build human behavior models, implemented in simulated agents, could be compared. The agents developed by the knowledge engineer were exposed to the same scenarios as those created through GenCL, and their behavior could be compared to the real driver's behavior.

Table 16 and Table 17 compare the agents developed by the knowledge engineer (KE) and the GenCL agents to the driver's behavior in the Driving Simulator. Table 16 compares the agent's

behavior to the driver's behavior within the training environment and in Table 17 the comparison is made in the validation environment.

Table 16

Comparing GenCL and Knowledge Engineer agents in the training environment

	Speed [km/h]		Time [s]		Speed
	RMS	Std.Dev.	RMS	Std.Dev.	Correlation
KE agent C vs. Driver C	7.94	7.81	4.35	4.35	0.894
GenCL agent C vs. Driver C	6.74	6.72	2.10	2.06	0.920
KE agent D vs. Driver D	8.83	8.88	9.55	9.01	0.852
GenCL agent D vs. Driver D	8.46	8.45	3.13	3.12	0.842

Table 17

Comparing GenCL and Knowledge Engineer agents in the validation environment

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

Comparing GenCL agent C and KE agent C with driver C, the agent evolved by GenCL performs consistently better than KE agent C. Comparing D agents to driver D, the GenCL agent performs better in the training environment and slightly worse than KE agent D in the validation environment. The interesting result here is that the GenCL agent is able to perform almost as well as an agent developed by traditional means even if the GenCL agent D was affected by the

lack of richness in the training data.

Another interesting observation to make is that both the GenCL and the KE agents C perform slightly better than the D agents do. This might infer that agent D's behavior might be more difficult to model than agent C's behavior.

Bergström [9] states that 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 workload. If the GenCL was generic enough and available, this approach could probably reduce the development time significantly. 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 hours. This is an approximation, since all of the experiments were done with a screen saver that was active when no user used the machine. The development of the screen saver is described in section 7.1.3. There was no log on how much CPU time the screensaver gained access. Time needs to be added for data collection, some preparation and a small effort to get the evolved code into use. As complexity and size of the problem increase, the use of an automatic creation of the agents would probably reduce development time even further. An example would be hazardous situations where the knowledge engineer approach would have to rely on interviews. Developing models for such situations would probably improve performance of the agents developed by learning by observation compared to those developed in a traditional manner. To reduce this development time further would be to research and develop

the Observer Module (see Figure 15).

This test shows that the learning and generalization capabilities of GenCL are able to create an agent performing at least as well as an agent developed through traditional means.

6.6. Ease of Use Evaluation

The objective of this evaluation was to investigate how difficult the GenCL algorithm is to use, in terms of how sensitive the performance is on different GP settings. If the performance is very sensitive on the GP configuration and parameter settings, it could be regarded as difficult to use, since a deep knowledge about GP is required.

These results present a collection of the results gained throughout the experiments. As described in section 5.3.2, the GP settings varied during different experimental runs as described in Table 4. Besides different settings of the GP parameters, the function set was also varied throughout the experiments. The different function sets used are also described in section 5.3.2. The sub-set *lightFunc* was only used during the evolving of Sub-Context **Traffic-Light-Driving**. Evolving the action rules in all the contexts, all other function sets were varied. When evolving the sentinel rules for **Traffic-Light-Driving** and **Intersection-Turning**, the only function set used was the *compOperator* set. This reduction of the function set was done because the knowledge within these sentinel rules is a matter of fulfilling a set of conditions to activate a specific Sub-Context (i.e. classification problem). This reduction of the function set is trivial using SME knowledge

(e.g. when a traffic light is spotted, another behavior pattern is used). Hence, no specific GP knowledge is needed for this task.

Table 18 through Table 22 describes the fitness values for agent A, B, C, D and E in their different contexts and sentinel rules evolved in the different experiments. Here, the fitness value is presented as a percentage. To recalculate the fitness to a percentage, the range of the input variables used in a specific training scenario (e.g. City Driving) is placed into the fitness formula. By dividing the actual fitness with this value, it will be described as a percentage. In these tables, the best, worst and average fitness is presented as is the standard deviation in fitness between the different experimental runs. Additionally, the number of different experimental runs is also presented in the tables. Note that not all possible combinations of configurations were evaluated, and in some scenarios with many test runs, there could be experiments with the same configuration. Even if two experiments have the same configuration, the result will not be the same because GP is a stochastic search algorithm where the outcome is based on probabilities. Hence, different seeds to the random generator will yield different outcomes.

The most interesting part of the tables is standard deviation. It shows a low variation among the different settings. The variation between the best and worst performing results are small in most of the cases. These results indicate that the algorithm is insensitive to different GP configurations. By inspecting some of the results with identical configuration, they can be found among the top performing experiments and among the worst performing ones, because of GP's stochastic nature.

Table 18

Fitness variations for agent A with different GP settings

	Average Fitness	Standard Deviation	Best Fitness	Worst Fitness	Test Runs
Green Light	2.86%	0.79%	1.49%	4.57%	170
Red Light	2.04%	0.23%	1.41%	2.47%	94
Traffic Light Driving	7.71%	1.53%	4.32%	10.32%	74
Intersection Turning	4.92%	0.58%	3.69%	7.83%	164
City Driving	10.15%	2.62%	5.70%	23.94%	68
Sentinel Rules (Traffic Light)	2.63%	1.01%	1.58%	4.28%	14
Sentinel Rules (Intersection)	1.49%	0.16%	1.29%	1.83%	14

Table 19

Fitness variations for agent B with different GP settings

	Average Fitness	Standard Deviation	Best Fitness	Worst Fitness	Test Runs
Green Light	2.41%	0.41%	1.80%	3.18%	94
Red Light	2.26%	0.40%	1.74%	3.61%	97
Traffic Light Driving	4.27%	1.13%	2.56%	7.47%	275
Intersection Turning	7.16%	1.26%	4.11%	11.23%	195
City Driving	18.28%	1.39%	12.56%	21.50%	143
Sentinel Rules (Traffic Light)	2.32%	0.47%	1.57%	3.60%	85
Sentinel Rules (Intersection)	2.55%	0.53%	1.71%	5.32%	85

Table 20

Fitness variations for agent C with different GP settings

	Average Fitness	Standard Deviation	Best Fitness	Worst Fitness	Test Runs
Green Light	3.09%	0.28%	2.40%	3.76%	315
Red Light	7.89%	0.56%	6.20%	9.11%	55
Traffic Light Driving	8.38%	1.10%	6.52%	10.49%	57
Intersection Turning	2.32%	0.15%	2.06%	2.70%	39
City Driving	6.64%	0.66%	4.55%	7.86%	22
Sentinel Rules (Traffic Light)	4.25%	0.47%	3.28%	4.93%	8
Sentinel Rules (Intersection)	3.05%	0.17%	2.91%	3.31%	8

Table 21

Fitness variations for agent D with different GP settings

	Average Fitness	Standard Deviation	Best Fitness	Worst Fitness	Test Runs
Green Light	4.90%	0.62%	3.63%	6.95%	137
Red Light	7.39%	0.73%	5.70%	9.50%	107
Traffic Light Driving	3.05%	0.28%	2.68%	3.33%	6
Intersection Turning	2.29%	0.69%	1.18%	3.27%	76
City Driving	5.77%	1.31%	3.68%	7.93%	28
Sentinel Rules (Traffic Light)	2.79%	0.53%	2.35%	6.12%	243
Sentinel Rules (Intersection)	6.91%	5.10%	1.91%	15.26%	243

Table 22

Fitness variations for agent E with different GP settings

	Average Fitness	Standard Deviation	Best Fitness	Worst Fitness	Test Runs
Green Light	5.63%	0.62%	3.37%	7.41%	232
Red Light	8.90%	1.01%	7.55%	13.53%	50
Traffic Light Driving	11.03%	1.25%	7.58%	14.65%	254
Intersection Turning	7.22%	1.03%	5.29%	10.65%	26
City Driving	8.94%	1.30%	6.00%	10.16%	11
Sentinel Rules (Traffic Light)	4.61%	1.50%	3.61%	12.17%	39
Sentinel Rules (Intersection)	8.89%	3.89%	4.24%	18.46%	39

Since no evaluation of all possible combinations of configurations was made, no detailed statistical analysis was made on this problem. Feldt [22] presented results from a thorough investigation of different GP configurations. In his research, he performed factorial experiments, investigating different settings' influence on the results. His results show that the two most influential parts of the genetic process is the population size and the number of generations the

evolutionary process was allowed to run. Feldt further showed that GP systems are insensitive to parameter variances. In this GenCL research, the configuration with 2,000 individuals operating in 2,000 generations might be substantial enough for the other configuration options to play a minor role in the evolutionary process.

The part, not related to GP that had even more influence of the success of the evolution was the data reduction process. As described earlier and shown in Figure 15, the task of the observation module was, in this work, done manually. Reduction of data was still needed since the data set was far too big to be practical for the experiments. When data is reduced, the importance and influence to the evolution of each sample increases. This affects the search pressure of different sub-tasks within the learning process. As an example, if the behavior of the agent making a turn at an intersection affects the fitness value to a much higher degree than traffic lights, the agent might never learn how to handle traffic lights in a correct manner.

The results presented here in the ease of use test show that the learning module within the GenCL algorithm is robust and not sensitive to different GP settings. Note that one of the better performing individuals was used in the final implementation of the agents but not necessarily the best performing individual. Because there is a hierarchy of contexts which are in the process of being built, some individual were put into use to enable the evolution of the next level (according to the LLGP scheme) of contexts before the evolutionary process at the lower level were completed.

6.7. Experiments: Summary and Conclusions

In these experiments, we have shown that the new approach to learning human behavior, GenCL, is able to learn and generalize a given problem. The agents evolved also show stable, long-term reliability. Further, the performance of the GenCL algorithm is fully comparable to agents developed by the knowledge engineering approach. Finally, the learning within the GenCL algorithm is insensitive to variations in the setting. Hence, the algorithm is easy to use and SME knowledge can be used to enhance the performance.

The results presented here are highly encouraging for further development and research concerning this new learning methodology. It has given significant encouragement for developing the Observation Module (see Figure 15) which completes our approach to learning human behavior by observation.

CHAPTER 7: SUMMARY, CONCLUSIONS AND FUTURE WORK

As described in the introduction and background chapters in this dissertation, developing human behavior models are complex and time consuming. Human behavior patterns can be very difficult to express, and the human might not even be aware of his or her actions. In order to reduce development costs and open new possibilities to incorporate implicit knowledge in the models, learning by observation has been investigated. In this research, we have presented a new learning methodology for automatic human behavior modeling from observed data. The new approach GenCL merges GP and CxBR. CxBR is an intuitive, efficient and effective methodology for creating tactical human behavior models applied to simulated agents. By merging GP to this methodology, a learning engine is created that is able to automatically create knowledge according to the CxBR paradigm. GP creates the knowledge in source code that makes it a flexible approach and easy to complement with additional knowledge after learning. Alternatively, a small model could be initialized with knowledge from SMEs or doctrines and then be refined through the evolutionary process.

The experiments conducted have shown good learning capabilities, generalization, long-term stability and insensitivity to GP parameter settings. The new methodology has further shown a capability to create simulated agents with at least the same performance as agents created

through traditional means. The results from the experiments have also shown that the hypothesis stated in section 2.2 was accomplished.

7.1. Research Observations

Two observations were made while performing this research that were not related to the objectives of the research but are, nevertheless, worth mentioning. The first is the positive effects of using a learning algorithm that produced transparent knowledge structure. The contextual knowledge produced by GenCL is represented in source code statements. To verify the lack of richness in the training data, the knowledge evolved by GenCL was investigated. Because the knowledge is transparent, the behavior of the evolved agent could be thoroughly investigated and new insights were made into its behavior. In a similar manner, the transparent knowledge could be investigated, after learning was conducted, to better analyze the observed human's behavior patterns. This can open new application areas where the knowledge not only could be used to implement human behavior into simulated agents, but also to analyze the action taken by the observed humans (e.g. after-action reviews).

The second observation concerns the complexity of feeding applicable data to the learning algorithm. One result derived from the experiments shows that the data used for learning highly influences the learning and generalization performance of the algorithm. It is highly important that the data used for learning have an equal pressure on the different tasks being learned. This is probably a general machine learning dilemma, but it highlights the importance and complexity of

the observational module. Creating a generic observational module is difficult and challenging. To create one for a specific problem might be feasible, however.

7.1.1. The Relationship to Reinforcement Learning

As described earlier, reinforcement learning consists of four sub-elements: a policy, a reward function, a value function and a model (optional). The policy that defines the behavior at any given time, serves a similar purpose as contexts in CxBR. The reward and value functions' intention is to direct the learning towards better performance as the fitness function in GP. The value function estimates the long-time desirability of different state changes while the reward function gives an immediate response from the environment. This is something that normally could not be found in evolutionary systems. In the case where a simulation is used to evaluate the performance, both the short-time rewards and the long-time rewards are taken into account when designing the fitness function. The difference lies in the estimation process of the long-time reward. In the GenCL algorithm, the value function does not need to be predicted since the simulation could be run to measure the long-time reward.

Another aspect that differentiates the evolutionary approach from reinforcement learning is that the evolutionary approach deals with many individuals evaluated, while in reinforcement learning there is usually only a single learning agent. The model in reinforcement learning is a model of the environment that could be used for planning. This is an expansion of the reinforcement learning where the learning system also learns to model the environment and uses

the model for planning. This is comparable to when the GP device builds and updates the action rules or the mission's plan in the context-base in CxBR.

The conclusion is that there are many similarities between CxBR + GP and reinforcement learning. There are many similarities, inspirations and theories from the reinforcement learning community, even though it is not learning by observation that could assist in the development and progress of this new approach.

7.1.2. Initial Stability Problems

The first set of experiments conducted did not use a simulation within the learning paradigm as described in Figure 16. The learning was conducted only using input to output mapping. In an input-output mapping scheme, the learning is conducted merely by mapping the correct outputs to the given inputs. The learning seemed to work well, but when the evolved models were inserted in the Traffic Simulator to work autonomously, some severe problems were discovered.

In the initial experiments conducted, the fitness value in the GP module was configured to measure the accelerator and brake pedal pressure deviation between the GP individual and human. This is a common machine learning approach where the inputs are mapped to a benchmark output (i.e. a given set of inputs should result in corresponding correct outputs). Figure 29 shows how the learning was conducted by mapping the input variable's speed, distance and light to the appropriate output value (i.e. throttle/brake pressure). The throttle/brake

pressure will affect the individual's simulated car and produce a new speed and a new position. Hence, this will be adequate to model the human's behavior if the learning is successful. Even if this method of learning seemed to work well, with a small deviation between the evolved agent's pedal behavior and the human's, a stability problem of the evolved agent occurred when it was operating autonomously in a simulated environment.

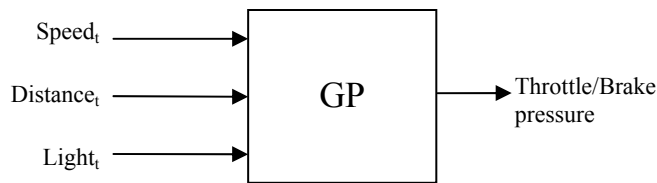


Figure 29: Learning through input – output mapping

When the agent is trained with input-output mapping, the correct output is learned for the specified inputs presented to the learning algorithm. When the agent, after training is over, operates in a simulated environment, the agent in reality will not experience what is described in Figure 29, since its own action is the cause of the speed and position at the next simulator time instance. In the training configuration of the input/output mapping scheme, all the inputs came from the recorded human performance. Actually, there is a feedback loop in the system that will store the accumulated error produced by the agent's own behavior (i.e. when the agent, by its own actions, produces the input in the next simulator step). What complicates the issue even more is when IF – THEN statements are part of the GP's function set. The use of such conditional statements makes the learned behavior function discontinuous. IF – THEN can be

described as a discontinuous step function. If the remaining error accumulated in the agent's behavior becomes too large, it can trigger a branch of the GP-evolved code tree never tested during training, and its behavior becomes totally unpredictable.

Actually, the accumulated error might not at all be particularly large for this to occur. During learning, all the inputs come from the recorded human and is therefore well aligned with a known correct behavior. When the agent, after learning is complete, operates autonomously in the simulated environment, a small drift in the agent's performance might trigger code (due to the use of discontinuous functions (such as IF-THEN) stored in the agent but never tested during training (i.e. non-coding regions)). It is a known fact that the GP's learning process involves non-coding regions. One approach to attack this problem might be to remove non-coding regions when learning is complete.

Another approach adapted in this research is to make the learning process look similar to the environment of the evolved agent. The solution is to let the individuals run small micro-simulations and be autonomous for a short duration during the learning process. The dynamics of the learning system are now shown in Figure 30. Now, only the initial input values are gathered from the recorded human performance as described in section 5.3.1.4. The GP individual is initiated and then set to operate in a simulated environment for a restricted duration. As the individual is passing predefined evaluation points, its performance (i.e. speed, position and pedal pressure) is compared to the recorded human performance, and the deviation constitutes the fitness value of the individual. Now the accumulated deviation is part of the learning process.

Furthermore, if the individual drifted away from the human driver's behavior (e.g. speed increases) after a number of simulation cycles, and the individual is starting to behave differently, it will affect the fitness value significantly.

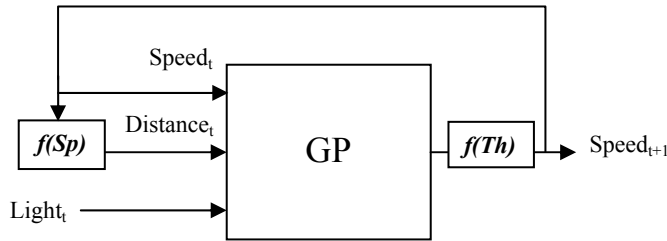


Figure 30: Learning through micro-simulation

The two boxes $f(Sp)$ and $f(Th)$ are actually part of the Micro Simulator described in Figure 15. The $f(Sp)$ box is calculating a new position as a function of the current speed and current position. The $f(Th)$ is the car model that calculates the speed of the car as a function of the throttle/brake pressure and the current speed. Even if the GP module of the GenCL artifact still learns the throttle/brake pressure, the fitness function takes both speed and distance, besides pedal pressure, into account when it is calculated.

7.1.3. Implementation to Maximize Computational Power

The process of evolving GP is computationally complex and often a time-consuming task. In this research, complex knowledge structures evolve at different hierarchical levels. To be able to run the experiments within an acceptable time frame, either an extremely fast computer or some sort

of parallel execution must be available. Inspired by the SETI@home project [2] at the Space Sciences Laboratory of the University of California at Berkeley, a screensaver was developed that incorporates the GenCL artifact. Implemented in a screensaver, the GenCL artifact could be executing the evolution process on many computers while not used by others. The screensaver is a multi-threaded application containing two major threads. One is running the GenCL artifact that evolves the contextual knowledge (i.e. GP individuals). The other thread is running entertaining animation but also regularly presents the GenCL learning status (i.e. fitness and source code of the currently best individual). The configuration with multiple threads ensures the graceful degradation of the GenCL artifact without disturbing the user with a slow ending screensaver.

Each computer with the GenCL screen saver also had a list of experiments to run. When the computer is done with one experiment, it reports the results via email and continues with the next experiment in the list. If a user enters the computer, the experiments are saved and set aside to permit the user to use the computer. They are resumed as the computer comes to a rest. In this manner, many experiments were efficiently conducted in parallel (45 Pentium 4, 1.8 GHz computers were used) with minor effects on computing resources.

7.2. Conclusions

This research has shown that individualized human behavior models can be created automatically from observed human performance. All human behavior knowledge, except the

context structure, in the agents created within this research were automatically created by the GenCL algorithm. The tests have also shown that those agents were able to generalize the knowledge and provided a stable performance with comparably high standards.

The results presented here have shown the ability of GP to produce knowledge in all the different parts of the CxBR's context base. GP has been able to evolve knowledge in the action rules of the contexts and the knowledge in different types of sentinel rule implementation (direct transition and a variant of competing context transition). Using the structure of CxBR also improves the learning capabilities of GP. In this research, the CxBR gave GP a frame in which to conduct learning. By dividing the general problem into sub-problems according to the CxBR structure, a learning strategy is formed, one not very different from LLGP. The work with LLGP has shown that this is a way of boosting the learning performance of the GP algorithm. The good results from this research indicate the same result. The conclusion is that the newly developed GenCL algorithm can learn, generalize and build stable models of tactical human behavior.

7.3. Future Work

Even if the experiments presented in this dissertation have shown positive results, some corrections could be made in the learning phase to improve the performance of GenCL. First, the training data and validation data could be restructured, as described in section 6.3.2.1, so the training data would be richer and cover more of the humans' behavioral patterns. The other thing that could improve the results of training is the removal of throttle/brake pressure in all the

fitness functions. If there is a discrepancy in the car model used in GenCL and the car model used during data collection, the combination of pedal pressure and speed in the fitness function will not conform. If the learning algorithm evolves something that improves the speed correlations, it might be that it would increase the pedal pressure deviation when there is a discrepancy in the car models.

Furthermore, it could be worth investigating how non-coding regions produced by the GP might affect the performance of the evolved agents. If the code not addressed during evolution is removed, would that improve or degrade the agent's performance? This is an interesting topic for future research.

To explore the automatic creation of human behavior models further, the introduction of *Fuzzy Logic* would be interesting. As mentioned in section 4.3.3, GP could use Fuzzy Sets and Fuzzy Logic as terminal and functions sets. Hence, the evolved models will then have a CxBR context base with Fuzzy representation. One interesting aspect of this would be to see whether the agents would show less discrepancy to the real driver and whether the individual knowledge in the agents could be enhanced by using fuzzy logic.

The results presented in this dissertation have only applied the new GenCL to one application – human driving behavior. The results have been encouraging, but to explore the usefulness of the algorithm fully, it should be applied to more and different application areas.

To make learning by observation complete, the development of the observer module is necessary. In this research, the extraction of data was done manually. If the observer module could be developed, it would reduce the development time of human behavior models even further and open up new application areas. With a fully working observer module, learning by observation could be done on-line and in near real time. One critical but difficult issue is to develop a generic observer module. As the problem space differs from time to time, the most problematic task for the observer module is to be applicable to many different problems. If the module needs to be crafted for the problem at hand, the reduction of development time might not be significant when compared to manually reducing the data set.

Another interesting approach towards completing learning by observation is to minimize the computational effort of the observer module. This would imply the investigation of the new GenCL algorithm to a huge data set with a greater problem space to tackle. The two main issues would have to be the computational power and the time for learning if this were to be successful. One way to approach the computational complexity would be to implement the GenCL algorithm into the Massively Parallel GP Engine developed by Eklund [20]. This GP engine is designed for hardware implementation and is an efficient and portable solution that, combined with GenCL, could be developed to be a solution that facilitates the prerequisites for on-line, near real time, development of human behavior models [23].

The objective of this research was to facilitate and improve the creation of building agents with human behavior. The new GenCL algorithm used in this research to build human behavioral

models automatically could easily be modified so that the evolved agents could be equipped with learning capabilities. The only adjustment that would be necessary in the learning module is to rewrite the fitness function. Instead of letting the fitness function compare the action of the agent with the SME, the fitness function could be arranged to encourage certain behavior or even behavior that after a substantial time leads to better results. By making this small adjustment to the fitness function, the learning module could be incorporated into the simulated agent and the creation of an intelligent agent could be formed. An additional module would need to be incorporated in the intelligent agent that actually recognizes when the behavioral knowledge has been improved enough to put into use.

APPENDIX A: LONG-TERM RELIABILITY

During the *Long-Term Reliability* test, the traffic lights in the simulation were set to have a cycle time of 25 seconds (i.e. the time to transit from green->yellow->red->yellow->green->yellow). The agents were tasked to do a circular loop in the city where they passed six different traffic lights (numbered 2, 3, 4, 6, 7 and 8). It took the agents between 226 to 301 seconds to complete the loop, depending on their individual behavior. It seemed that most of the agents fell into some pattern where they stopped at the same lights at each loop in the city. The agents' behavior was only recorded when they were within 100 meters of the light and the light was either yellow or red. If the agents were further away from the light or if the light was green, their behavior was not recorded. Some of the lights were passed when they were constantly green and their behavior was not recorded. The recorded lights were different from agent to agent. Note that Swedish traffic lights have a state of yellow in-between red and green.

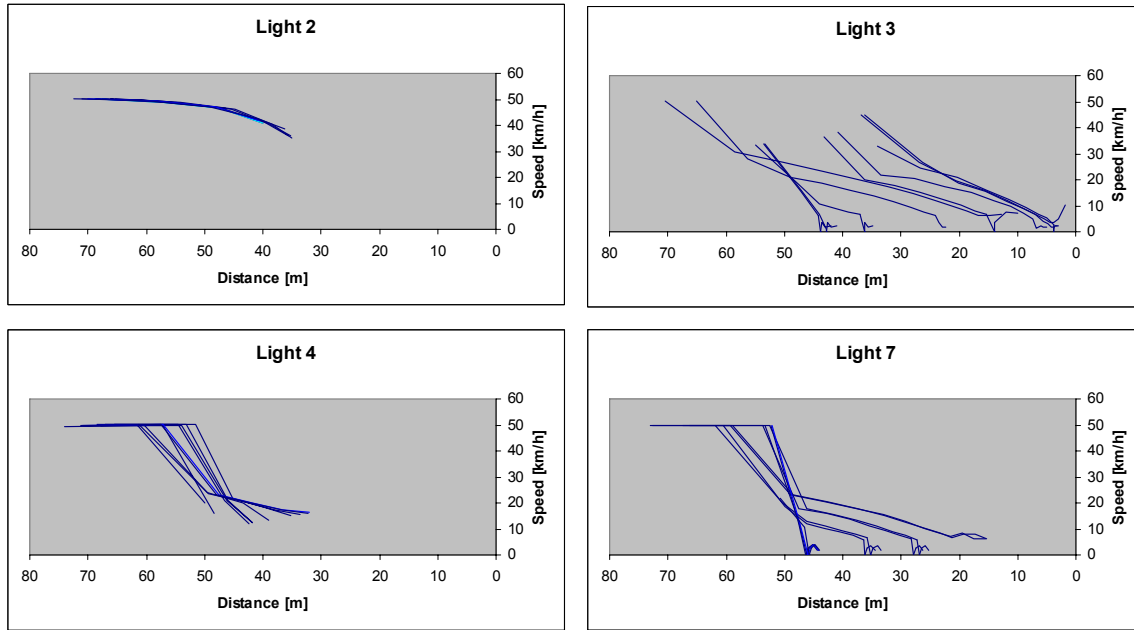


Figure 31: Long-term behavior of agent A

Figure 31 shows the recorded data of agent A. Lights 2 and 4 change from red to green when driver A approaches the light. Agent A is consistent and slows down when the light is red, but never comes to a complete stop while lights 3 and 7 turn red as the agent approaches the lights. Here we can see that when the light turns from green to yellow and then to red at different times, agent A has different stopping patterns and will stop at different locations almost every time. Agent A is consistent and stops every time a light turns red, and the different distances can be related to the training data configuration. All the lights within the training data changed their state when the car was 30 meters from the light. Hence, the anticipated stopping distance would be approximately 30 meters after the light changed from green to yellow.

Figure 32 shows the behavior of agent B. Lights 3 and 8 turn red as the agent approaches the lights, while lights 4 and 6 turn green. When the lights turn red, the agent comes to a stop at almost the same location every time, except when the light turns red extremely late. When the light is red (about to turn green), the agent slows down and picks up speed again when the light goes from red to yellow. At one occasion, the agent slows down early but never comes to a complete stop before the light turns yellow (to be green). The consistency of agent B is strong and stable.

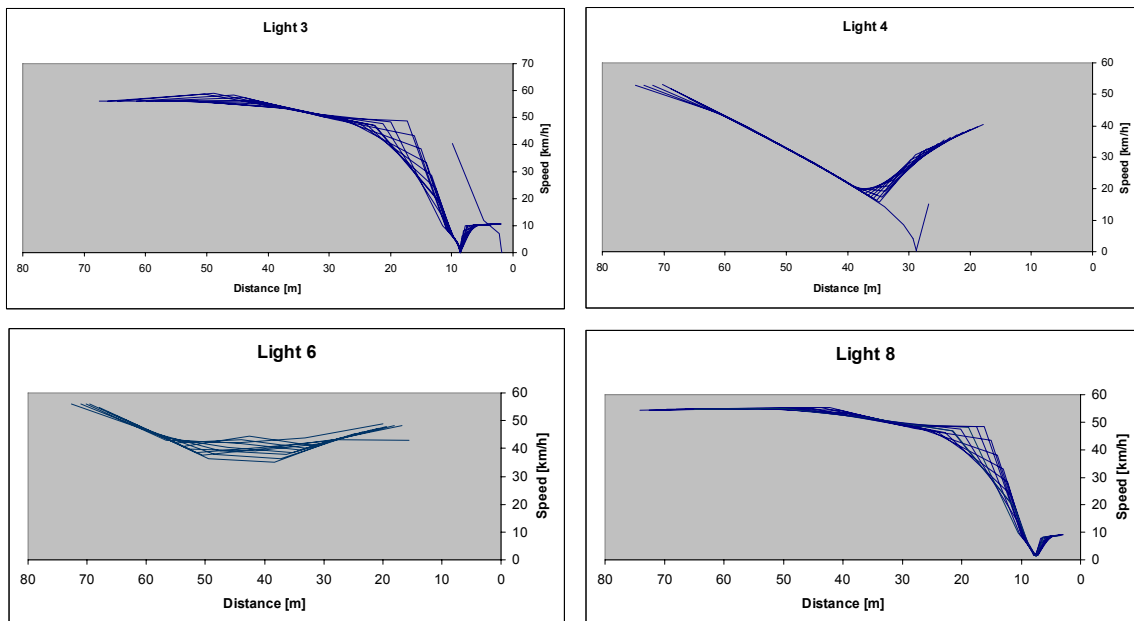


Figure 32: Long-term behavior of agent B

The long-term behavior of agent C is shown in Figure 33. Agent C is the most careful agent. The agent comes to a stop at five of the six lights, where the sixth light was constantly green

each time agent C passed it. Note that lights 7 and 8 are shown in the same diagram. Agent C is consistent, as the agent comes to a complete stop at almost the same spot each time.

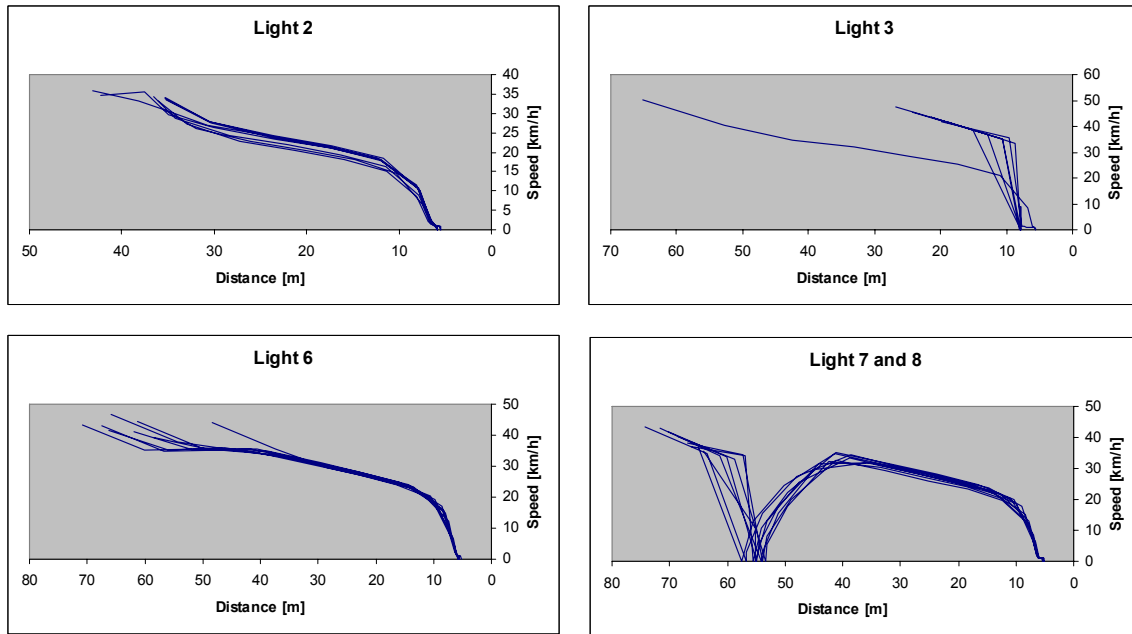


Figure 33: Long-term behavior of agent C

As agent D passes the lights, it is only light 4 that turns from red to green. Actually, on one of the occasions light 4 turned red as the agent approached the light. In some cases, agent D's behavior is especially interesting to observe. Squares in the diagrams indicate when the lights turn red or when the light is about to turn yellow (about to be green). When the agent comes to a complete stop, it stops at almost at the same spot each time. At light 2, it seems like the stopping distance is varied, but the agent actually slows down to a very slow speed and then comes to a complete

stop. The variance of all the stops at light 2 is only 2 meters.

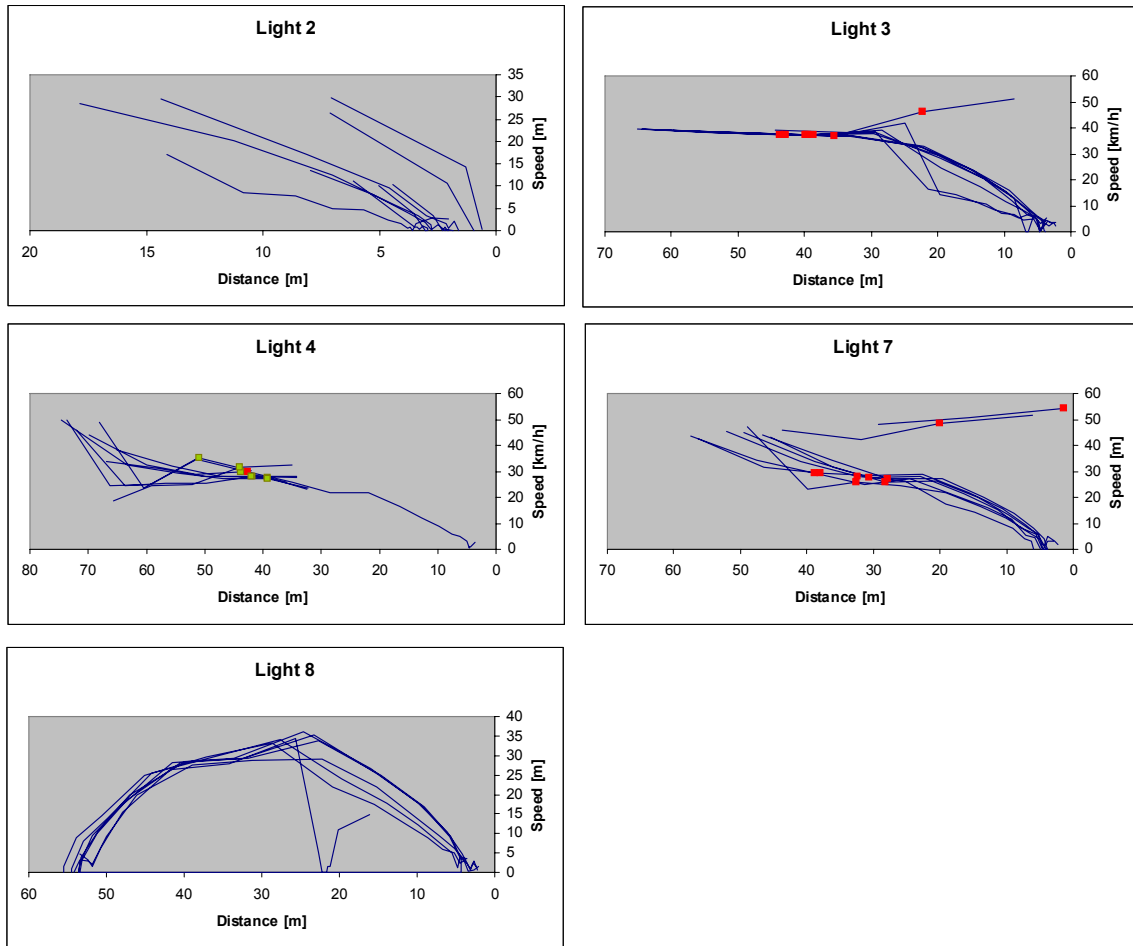


Figure 34: Long term behavior of agent D

The only outlier is at light 8 when the agent actually makes a quick stop 22 meters prior to the light but then slowly increases speed, even if the light is still red. Agent D performs normal acceleration first when the light turns yellow. In two occasions, at light 7 and one occasion at light 8, the agent actually runs the red light. We can see that if the light turns red when the agent

is at 27 meters or earlier, the agent stops. However, if it turns red when the agent is at 23 meters or later, the agent runs the light. Even if the behavior is not desirable, it is still consistent.

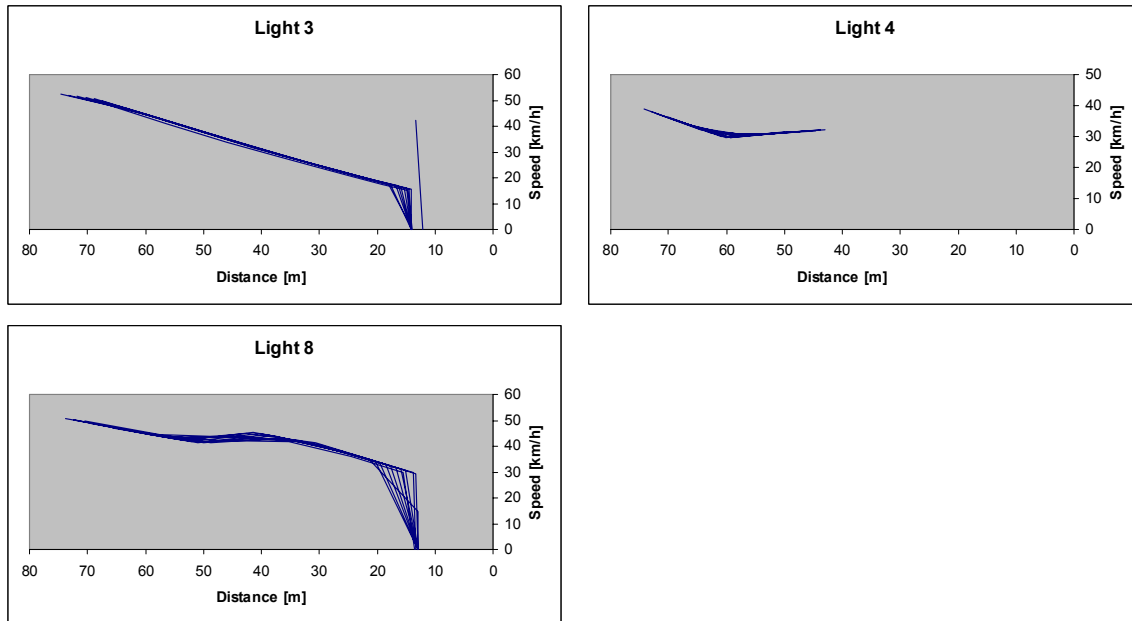


Figure 35: Long-term behavior of agent E

Agent E’s behavior is shown in Figure 35. The light was either yellow or red in only three of the lights that agent E passed. Light 3 and light 8 turned red and the stopping distance is consistent except for one occasion where the agent brakes hard to come to a complete stop when the lights change late. Light 4 changes from red to green and agent E is very consistent when slowing down as the light is red and picks up speed when the light is turning yellow.

All the agents show consistent behavior when approaching the traffic lights. Agent A is the only

agent that stops at different distances when the light turns red. It is surprising that the other four agents are able to generalize that well and always stop close to the lights. This is because all the traffic lights during the training scenarios changed their state from green to yellow when the driver was 30 meters from the light. Hence, the most obvious behavior expected in the agents would be a stopping distance approximately 30 meters after the light changes from green to yellow. The fact that all the agents are still running after 40 minutes shows stability when it comes to intersection turning. If the agent would not slow down when taking a turn, the simulated car will not be able to take the turn and end up beside the road. There are no towing cars in the simulation, nor any recovery algorithms implemented, so the car would be stuck. Hence, all the agents show consistency to apply the **Intersection-Turning** Sub-Context accurately.

APPENDIX B: SENTINEL RULES OF AGENT B AND AGENT D

Agent D, Sentinel rules - Traffic Light Driving

```
if(distance<76.485488)
{
  if(!lightPresent)
    return 0;
  else
  {
    if(distance<=24.814600)
      return 1;
    if(distance>24.814600 && distance<=27.884762)
      return 0;
      if(intersection)
        return 0;
        if(mySpeed<24.250008 && distance>37.684255)
          return 0;
          else
            return 1;
  }
}
else
{
  if(intersection && lightPresent)
    return 1;
  else
    return 0;
}
```

Agent B, Sentinel rules - Traffic Light Driving

```
if(mySpeed>19.278542)
{
  if(!intersection)
  {
    if(distance>=70.607623)
      return 0;
    if(lightPresent)
    {
      if(distance<17.438276 || distance>39.167455
        || distance>mySpeed)
        return 1;
      else
        return 0;
    }
    if(distance>49.220252)
      return 1;
    else
      return 0;
  }
  if(intersection && distance<17.438276)
    return 1;
  else
    return 0;
}
if(lightPresent && mySpeed<=19.278542)
  return 1;
else
  return 0;
```


APPENDIX C: COMPLETE DESCRIPTION OF THE DATA SETS

The data sets were collected on two different occasions. The data from the first occasion is referred to as the training set and the second data set as the validation set. In both sets, the five different drivers were exposed to city driving and rural driving. The training data set took the drivers approximately 30 minutes to conduct, and the validation set about 20 minutes. The driver spent approximately two-thirds of the time in city traffic and one-third in rural traffic in both occasions. In both occasions, one data set was collected from each driver (totally 5 + 5 data sets).

During the collection of the training data, the drivers started in the city center and drove to an office. From the office, the drivers go to a gas station and then home. Figure 36 shows the main part of the city route. Figure 36 through Figure 38 are extracted from the requirement description (with Virtual Technologies approval). The numbered circles in Figure 36 and Figure 37 note interesting places in the training environment:

1. City center
2. Office
3. Roundabout
4. Gas station
5. Home

Green circles in the figures indicate traffic lights and blue circles indicate hazards.

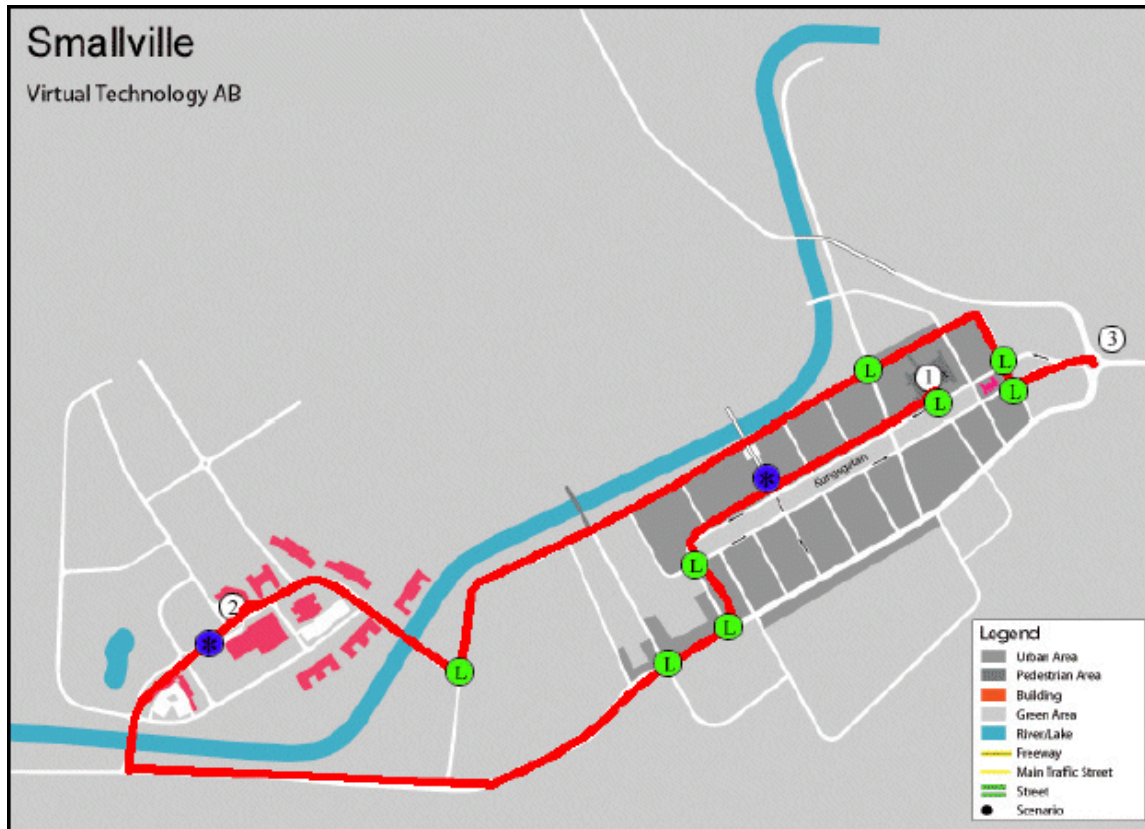


Figure 36: The city driving during the collection of the training data set

During the training scenarios, the driver passed 11 traffic lights. In Figure 36 and Figure 37, there are only 9 traffic lights showing, but the driver passed two of them during its second time through the city, so the total number of traffic lights passed were 11. Six of the lights change from green to red, four from red to green and one was constantly green. In the training set scenarios, all the traffic lights changed their state when the car was approximately 30 meters prior to the light. Lights 1, 4, 8 and 4b (second pass of light 4) changed from red to green, light

3b was constantly green while the rest of the lights changed from green to red.

The hazardous situations during the training set collection were:

- Car from side road does not stop and turns into the same lane just in front of the driver.
- Road maintenance blocking the lane.
- Car standing still in the lane while a car is approaching in the opposite lane.
- A slow moving tractor occupies the lane.
- Moose at the side of the road.
- Children at a pedestrian crossing with a green traffic light.
- Bus leaving bus stop without signaling.

Observe that all the hazardous situations occur on roads with opposite traffic in the other lane.

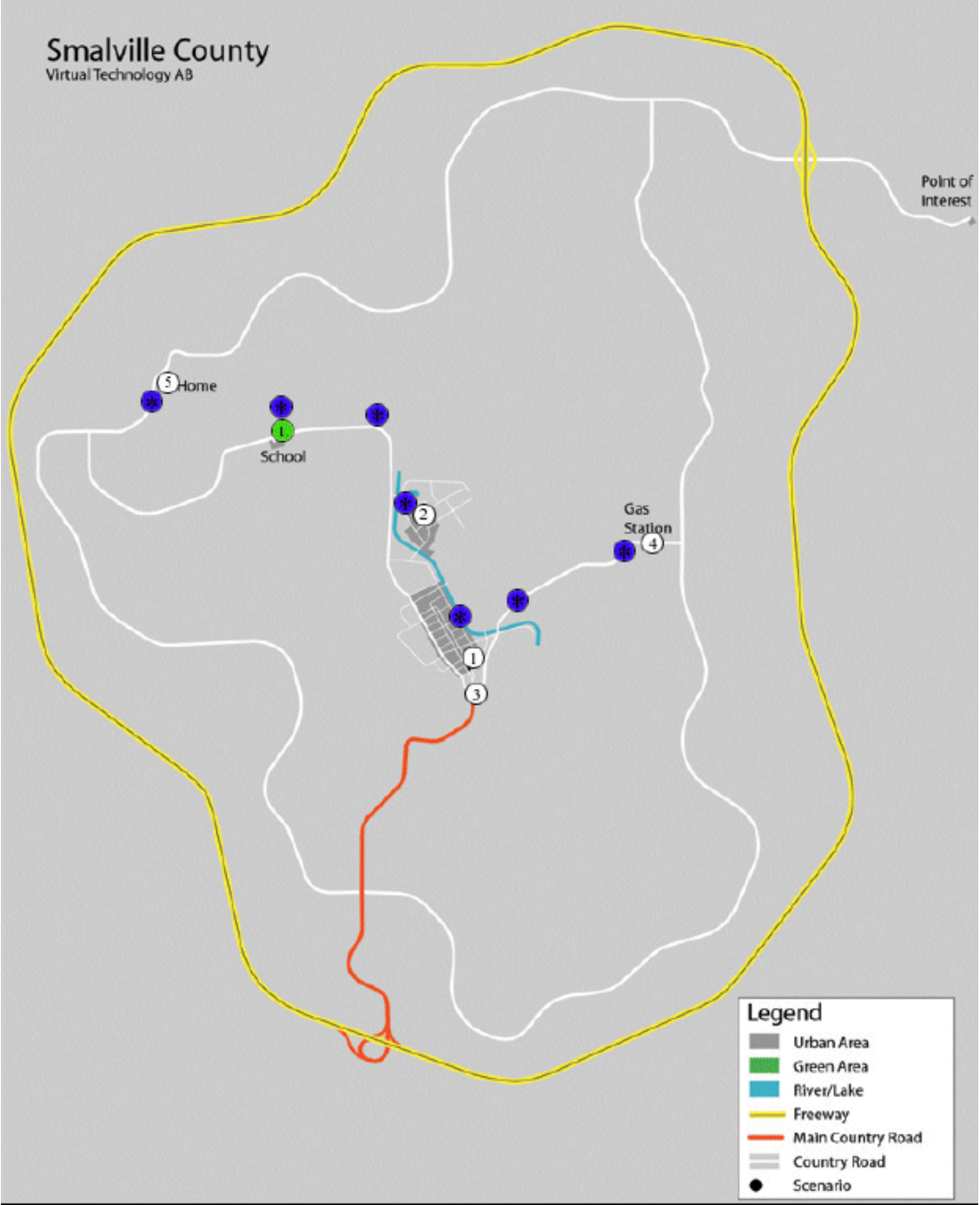


Figure 37: The rural driving during the collection of the training data set

During the collection of the validation data, the driver again starts at the city center marked as position 1 in Figure 38. This time, the route through the city is different and at position 2, the drivers turn back towards the city and eventually leave the city at position 3 where they make a short rural driving session.

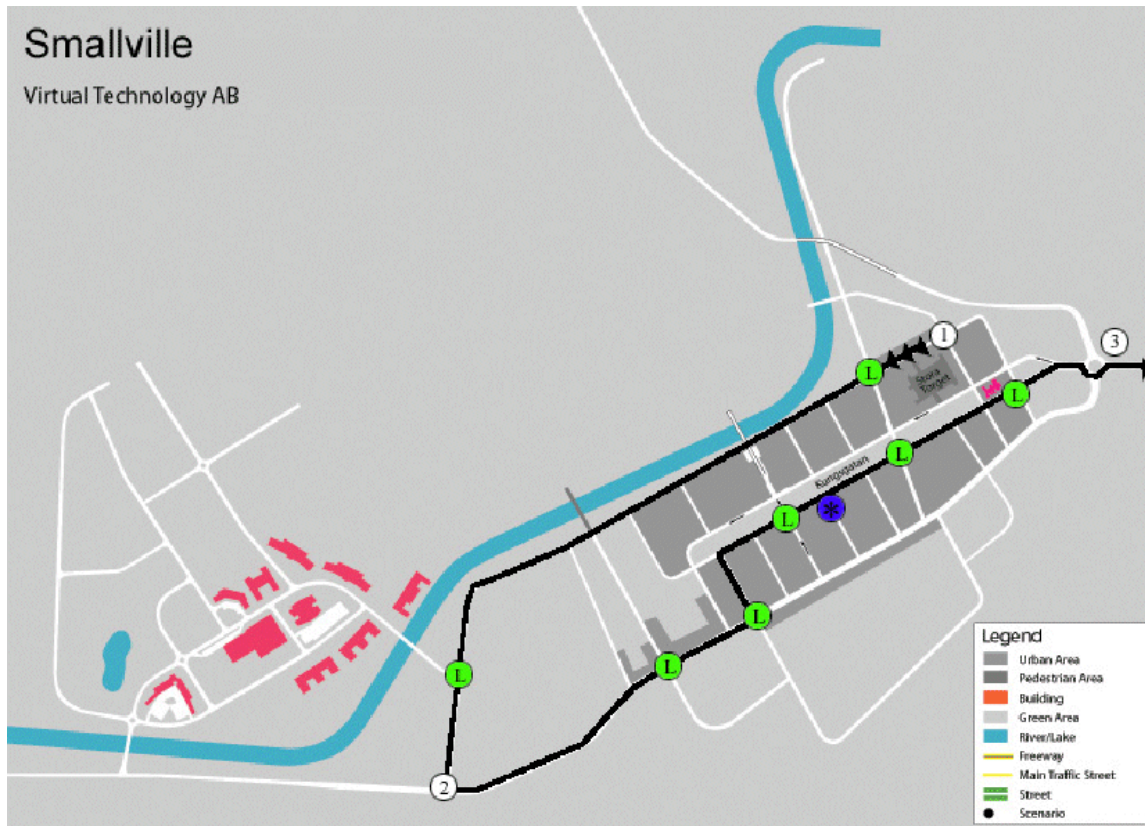


Figure 38: The city driving during the collection of the validation data set

The traffic lights in the validation scenario now change their state at different distances and all of the lights, except one, change from green to red. The characteristics of the seven traffic lights are

described in Table 23.

Table 23

Traffic light characteristics in the validation data set

Traffic Light ID	Activation Distance [m]	Change
TL_01_INS	30	Green > Red
TL_02_INS	35	Green > Red
TL_03_INS	30	Red > Green
TL_04_INS	40	Green > Red
TL_05_INS	35	Green > Red
TL_06_INS	30	Green > Red
TL_07_INS	35	Green > Red

In the validation scenario, there are also seven hazardous situations occurring with different severity. Those occasions are:

- Head-on traffic
- Head-on traffic at the traffic light
- Bus crossing the road at traffic light 3
- Enforced yielding to crossing traffic
- Aggressive car stops very late in an intersection
- Head-on traffic on a rural road
- Slow moving traffic on rural road

Last in this appendix, there is an example of data from the scenarios. Each row in the data sets

represents one sample. The first column shows the time from the start of the simulation in seconds when the sample was taken. The next three columns show the position of the car. The units of the coordinates are meters. Heading, pitch and roll are measured in degrees. Steering wheel angle is a linear, unit-less measurement of the wheel angle in the range from -1 to 1 (left to right). Throttle and brake pressure are unit-less measurements of the pedal pressure in the range of 0 to 1, where 0 is no pressure and 1 is the pedal fully pressed. Speed is simply the current speed of the car measured in km/h. HotSpots are indicators in the simulated environment placed there as helpers to interpret changes in the environment. All HotSpots tell us where and when changes take place. When a HotSpot pops up in the data set, it means that something changed state in the simulated environment (e.g. the traffic light changes its state from green to yellow or an approaching car comes within visual range). As the HotSpot ID changes, some changes in the driver's environment occur. The last column shows the distance to the HotSpot. If the distance is negative, the HotSpot is in front of the car. If it is positive, the car has already passed the HotSpot.

Time	X	Y	Z	Heading	Pitch	Roll	Steering Wheel	Throttle Pressure	Brake Pressure	Speed	HotSpot	HotSpot Distance
183.91	6070.68	-1674.85	3.71	69.88	0.08	0.57	-0.19	0.15	0	28.24	CAR_1_INS	262.26
184.04	6069.75	-1674.52	3.71	71.76	0.04	0.58	-0.18	0.15	0	28.4	TL_02_INS	-29.48
184.16	6068.86	-1674.23	3.71	73.49	0.03	0.58	-0.18	0.15	0	28.53	TL_02_INS	-28.54
184.28	6067.9	-1673.95	3.71	75.2	0.05	0.57	-0.16	0.15	0	28.66	TL_02_INS	-27.55
184.4	6066.94	-1673.71	3.71	76.85	0.04	0.55	-0.16	0.15	0	28.78	TL_02_INS	-26.55
184.52	6066	-1673.5	3.71	78.44	0.03	0.57	-0.16	0.15	0	28.88	TL_02_INS	-25.59
184.65	6065.02	-1673.3	3.71	80.06	0.03	0.59	-0.16	0.15	0	28.97	TL_02_INS	-24.59
184.77	6064.08	-1673.14	3.71	81.57	0.03	0.57	-0.15	0.15	0	29.04	TL_02_INS	-23.63
184.89	6063.07	-1673	3.71	83.12	-0.06	0.58	-0.14	0.13	0	28.71	TL_02_INS	-22.61
185.02	6062.08	-1672.89	3.71	84.58	-0.15	0.57	-0.14	0.12	0	28.37	TL_02_INS	-21.62
185.15	6061.14	-1672.8	3.71	85.96	-0.64	0.63	-0.13	0	0	25.67	TL_02_INS	-20.68
185.27	6060.29	-1672.75	3.71	87.2	-0.98	0.53	-0.13	0	0	23.51	TL_02_INS	-19.83
185.4	6059.5	-1672.71	3.71	88.39	-0.63	0.35	-0.13	0	0	21.71	TL_02_INS	-19.03
185.52	6058.79	-1672.7	3.71	89.55	-0.32	0.32	-0.14	0	0	20.24	TL_02_INS	-18.32
185.64	6058.11	-1672.7	3.71	90.72	-0.33	0.3	-0.15	0	0	18.95	TL_02_INS	-17.64
185.76	6057.5	-1672.71	3.71	91.78	-0.39	0.25	-0.14	0	0	17.88	TL_02_INS	-17.03
185.89	6056.9	-1672.73	3.71	92.8	-0.33	0.19	-0.14	0	0	16.89	TL_02_INS	-16.43
186.01	6056.33	-1672.76	3.71	93.69	-0.25	0.15	-0.13	0	0	16.01	TL_02_INS	-15.86
186.14	6055.8	-1672.79	3.71	94.45	-0.22	0.11	-0.12	0	0	15.25	TL_02_INS	-15.33
186.26	6055.28	-1672.83	3.71	95.07	-0.22	0.07	-0.09	0	0	14.54	TL_02_INS	-14.81
186.38	6054.82	-1672.88	3.71	95.59	-0.2	0.05	-0.09	0	0	13.94	TL_02_INS	-14.34

APPENDIX D: TRAINING DATA

This appendix lists the data samples used during the training of the agents. One data sample contains the current speed of the driver, the distance to a HotSpot and the throttle/brake pressure (labeled speed, dist and throt). The capital letter in the list names indicates the driver (A, B, C, D and E). The single number in the name refers to traffic light ID's. The notation i refers to an intersection close to, or in conjunction with, a traffic light. When data was used to evolve action within Urban Driving context, no intersections or traffic lights were present. The numbering of the data set refers to a section between two traffic lights (e.g. speedA56[10] is 10 speed samples of driver A in-between lights 5 and 6).

The table on the next page shows which data were used and when they were used. Each column refers to a data set (e.g. 8 refers to data as the driver approaches traffic light 8). Each row describes the action rules or sentinel rules in each context evolved (A: action rules, S: sentinel rules, UD: urban driving, IT: intersection turning, TLD: traffic light driving, G: green light driving and R: red light driving). Each cell in the table then describes which data samples were used for a specific scenario (e.g. evolving the agent A's sentinel rules for **Intersection-Turning** used, among other things, data samples 19 through 35 at traffic light 2 from driver A).

DriverA	2	4	5	6	7	8	12	34	56	7i	8i
S_IT	19-35	0-28					0-13			0-24	0-8
S_TLD		0-28	0-16	0-23			0-13			0-24	
A_UD							0-11	0-8	0-8		
A_IT	0-17*									15-28	0-8
A_TLD	20-38	0-28	0-28	0-28	0-18						
A_G		23-28	0-9	0-13		16-28					
A_R	29-38	0-15	20-28			0-7					

DriverB	3	4	5	6	8	2s	7i	i2	i3	i8	56
S_IT	0-18	0-24					0-24				0-13
S_TLD		0-24		0-23			0-24				0-13
A_UD											0-99
A_IT								0-9	0-9	0-9	
A_TLD	4-28	0-23	0-23	0-23	4-28	0-4					
A_G	0-5	18-23	0-10	0-14		0-4					
A_R	12-28	0-12	20-23		4-10						

DriverC	2	3	4	5	6	8	3i	7i	12	34	56
S_IT	14-28	0-26	0-28					0-13	0-13	0-8	0-8
S_TLD	14-28	0-26	0-28					0-13	0-13	0-8	
A_UD									0-9	0-8	0-8
A_IT	0-12*						0-9	5-13			
A_TLD		0-28	0-28	0-28	0-28						
A_G	0-11	0-14	22-28		0-10	10-18					
A_R	15-27	17-28	0-18		14-23	0-8					

DriverD	1	2	3	4	5	6	7	3i	7i	12	34	56
S_IT		0-10	20-29	0-28					0-14	0-13		
S_TLD		12-28	7-29	0-28						0-13		
A_UD										0-11	0-13	0-13
A_IT		0-9*						0-9	3-13			
A_TLD			5-28	5-28	5-28	5-28	5-28					
A_G			0-12	20-28	0-11							
A_R	0-14	13-27	13-24	0-12	14-25							

DriverE	1	2	3	4	5	6	7	3i	7i	12	34	56
S_IT		20-28	0-28	0-28					0-13	0-13		0-13
S_TLD		20-28	0-28	0-28	0-19				0-13			0-13
A_UD										0-13	0-8	0-13
A_IT		0-17*						0-9	4-11			
A_TLD			0-28	0-28	0-28	0-28	0-28					
A_G			0-14	19-28	0-9							
A_R	0-14		18-28	0-16	13-20							

* When data set 2 is used for Intersection Turning 35 meter is deducted from the distance measure, since the intersection is 35 meters ahead of traffic light 2

```

speedA2[40]={ 35.81,37.53,38.90,37.14,33.10,30.51,31.40,32.51,32.23,30.15,
31.14,34.86,35.46,36.91,37.59,36.96,35.73,35.08,35.51,36.24,
29.66,29.38,27.83,21.99,19.61,17.33,15.23,15.6,15.55,14.30,
12.59,11.44,10.35, 9.45, 8.72, 8.09, 7.36, 4.82, 0, 0};
distA2[40]={ 92.85,90.19,86.13, 82.6,79.93,77,74.23,71.41,68.44,65.58,62.34,
58.91,55.81,52.63,48.65,44.62,40.65,36.83,34.27,31.15,28.02,25.34,23.
18,20.91,19.39,17.71,16.24,14.89,13.22,11.59,10.17, 9.1, 7.96, 6.91,
5.97, 5.09, 4.28, 3.71, 3.53, 3.53};
throtA2[40]={ 27, 26, 25, 13, 8, 15, 21, 21, 13, 12,
27, 25, 21, 26, 22, 19, 19, 19, 24, 21,
8, 13, 6, 0, 7, 0, 1, 8, 5, 0,
0, 0, 0, 0, 0, 0, -4, -43, 0, 0};

speedA4[30]={ 44.29,42.03,40.05,38.91,37.99,36.71,35.19,33.72,32.54,31.81,
30.37,29.08,27.91, 26.8, 23.7,20.63,18.33,16.54,15.09,14.7,
15.92,18.56,22.11,25.48,28.63,31.49, 34.3,36.79,38.84,40.53};
distA4[30]={ 64.05,61.3,58.69,56.17,53.73,51.32,49.04,46.86,44.73,42.71,
40.73,38.83,37.05,35.31,33.65,32.2,30.9,29.78,28.78,27.86,
26.91,25.81,24.55,23.09,21.42,19.58,17.57,15.37,13.08,10.66};
throtA4[30]={ 3, 3, 3, 6, 6, 3, 1, 1, 4, 4,
0, 0, 0, 0, 0, 0, 0, 0, 0, 6,
14, 30, 33, 30, 29, 31, 31, 31, 31, 31};

speedA5[30]={ 48.34,47.04,45.88,44.7,43.65,42.58,41.72,41.14,40.83,40.24,
39.88,39.66,36.89,33.44,26.07,14.18, 6.37, 3.87, 3.69, 5.77,
8.24,10.89,10.07, 9.33, 8.78, 7.64, 5.55, 2.59, 0.15, 0.14};
distA5[30]={ 65.97,62.04,58.04,54.33,50.46,46.88,43.21,39.77,36.58,33.38,
30.19,26.94,23.6,20.44,17.64,15.85,15.02,14.63,14.29,13.87,
13.23,12.36,11.46,10.57, 9.85, 9.16, 8.61, 8.26, 8.16, 8.15};
throtA5[30]={ 0, 0, 0, 0, 0, 0, 2, 3, 14, 7,
12, 7, 0, -39, -90, -100, -75, -1, -1, 15,
15, 8, 0, 0, 0, -14, -36, -36, -36, 0};

speedA6[30]={ 48.23, 48.7,49.13,49.54,49.87,49.97,50.07,50.23,50.69,51.08,
51.50,52.09,52.66,53.18,53.66,54.18,54.58,55.00,55.94,57.51,
59.18,60.58,61.04,59.38,55.49,51.52,48.19,45.56,44.38,43.58};
distA6[30]={ 66.96,64.49,62.01,59.48,56.99,54.51,52.08,49.66,47.36,44.47,
42.13,39.79,36.92,34.02,31.12,27.7,24.79,21.47,18.17,14.87,
11.64, 8.39, 5.06, 1.83,-1.21,-3.99,-6.55,-8.97,-11.27,-13.52};
throtA6[30]={ 19, 19, 19, 19, 18, 17, 17, 20, 20, 21,
21, 22, 22, 22, 22, 22, 22, 22, 30, 36,
36, 30, 23, 10, 0, 0, 0, 3, 7, 11};

speedA7[20]={ 45.86,43.18,40.99,41.48,42.23,42.96,43.84,45.25,46.69,48.1,
49.22,49.73,50.16,50.05,49.65,48.69,48.26,48.71, 49.9,51.38};
distA7[20]={ 58.43,55.95,53.63,51.35,49.12,46.91,44.59,42.26,39.87,37.38,
34.64,31.88,29.28, 26.7,23.76,21.08, 18.4,15.77, 13.1, 10.1};
throtA7[20]={ 1, 0, 6, 19, 19, 19, 23, 28, 28, 28,
20, 20, 19, 15, 14, 13, 16, 20, 29, 29};

speedA8[30]={ 48.27,45.06,42.36,39.99,37.87,36.01,34.32, 32.8,31.43,30.18,
29.02,27.95,26.96,23.82,21.14,19.01,19.07,20.56,22.36,23.99,
24.81,23.47,20.98,19.42,20.39,21.44,22.47,22.99,22.89,22.69};
distA8[30]={ 47.74,44.91,42.36,39.94,37.67,35.51, 33.5,31.56,29.75,28.09,
26.45,24.87,23.35,21.98,20.77,19.67,18.62,17.55, 16.4,15.14,
13.81,12.47,11.24,10.15, 9.05, 7.9, 6.68, 5.41, 4.15, 2.86};
throtA8[30]={ 0, 0, 0, 0, 0, 0, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, 13, 17, 19, 16,
12, 1, -1, 6, 13, 13, 14, 9, 9, 8};

```

```

speedA12[15]={ 49.38,49.65,50.01,49.58,49.56,50.32,50.95,50.6,50.23,49.78,
50.16,50.85,51.48,51.96,51.71};
throtA12[15]={ 15, 19, 16, 16, 18, 19, 19, 16, 16, 16,
19, 19, 19, 19, 16};

speedA34[10]={ 28.48,29.31,31.96,35.54,38.73,42.04,46.37,49.11, 51,51.38};
throtA34[10]={ 16, 16, 25, 26, 29, 29, 26, 23, 21, 20};

speedA56[10]={ 70.89,69.53,68.32,66.51,62.84,59.57,56.84,54.31,52.31,52.66};
throtA56[10]={ 4, 12, 9, 2, 0, 0, 0, 0, 7, 14};

speedA7i[30]={ 45.79,48.19,50.58,52.74,53.42,52.54,51.82,51.18,50.64,49.99,
49.44,48.96,49.14,49.69,49.68,49.43,48.13,46.61,45.62,44.73,
41.22,37.97,35.15,32.75,30.76,29.03,27.55,25.57,23.06,23.06};
distA7i[30]={ 96.95, 93,88.62, 84.2, 79.6,74.98,70.59,66.2,61.96,57.69,
53.56,49.44,45.42,41.4,37.44,33.47,29.57,25.84,22.02,18.28,
14.75, 11.4, 8.2, 5.22, 2.54, 0.04,-2.24,-4.51,-6.59,-8.46};
throtA7i[30]={ 29, 29, 29, 29, 16, 15, 15, 15, 15, 14,
14, 14, 19, 17, 16, 14, 9, 9, 13, 7,
0, 0, 0, 0, 0, 0, 0, 2, 7, 15};

speedA8i[10]={ 20.39,21.44,22.47,22.99,22.89,22.69,22.58,22.92,23.41,24.15};
distA8i[10]={ 9.05, 7.9, 6.68, 5.41, 4.15, 2.86, 1.61, 0.34,-0.94,-2.24};
throtA8i[10]={ 13, 13, 14, 9, 9, 8, 11, 11, 12, 15};

```

```

speedB4[25]={ 48.43,46.16,42.01,37.98, 34.8,32.75,32.14,30.18,28.19,26.49,
20.84,17.34,14.97, 13.9,14.92,16.22,17.62,21.02,24.42,27.57,
30.12,32.09,33.52,34.54,35.24};
distB4[25]={ 71.08,66.24,61.75,57.63, 53.9,50.48,47.24,44.12,41.16,38.38,
35.94,33.97,32.32,30.86,29.44,27.92,26.26,24.35,22.14,19.58,
16.77,13.74,10.59, 7.29, 3.95};
throtB4[25]={ 14, 5, 0, 0, 0, 3, 6, 0, 0, 0,
0, 0, 1, 5, 10, 8, 15, 20, 21, 21,
21, 21, 21, 21, 21};

speedB6[25]={ 52.55,52.55,52.55,52.55, 52.4,52.16,52.21,52.44,52.77,53.14,
53.49,53.85,54.19,54.49,54.57,54.57, 54.5,54.41,54.3,54.22,
54.09,53.93,53.78, 53.6,53.41};
distB6[25]={ 72.44,69.67,66.93,64.18,61.43,58.73,56.01,53.05,50.28,47.48,
44.7,41.92,39.18, 36.1,33.24,30.39,27.25,24.47,21.37,18.58,
15.77, 12.7, 9.66, 6.65, 3.64};
throtB6[25]={ 8, 8, 8, 8, 5, 8, 10, 12, 13, 14,
14, 14, 14, 11, 9, 8, 8, 8, 8, 8,
7, 7, 7, 6, 6};

speedB8[30]={ 47.36,49.27,50.61,49.94,48.44,43.96,39.92,36.54,33.69,31.08,
28.08,23.24,18.55, 15.3,13.59,14.45,16.59, 18.9,20.77,21.84,
23.23,23.74,23.16,20.91, 17.8,15.52,14.11,15.15,16.82,18.08};
distB8[30]={ 67.46,63.55,59.54,55.44,51.32,47.34,43.66, 40.2,37.22,34.45,
31.91,29.58,27.87,26.46,25.28, 24.1, 22.8,21.28,19.65,17.86,
16.00,14.01,12.02,10.12, 8.44, 7.06, 5.82, 4.58, 3.29, 1.78};
throtB8[30]={ 25, 22, 18, 7, 3, -5, -5, -5, -5, -16,
-17, -17, -17, -3, -1, 10, 14, 14, 11, 13,
13, 10, 4, 1, -2, -2, 2, 11, 11, 7};

speedB2s[6]={ 1.23, 2.89, 4.6, 7.49,11.34,15.98};
distB2s[6]={ 5.03, 4.75, 4.26, 3.47, 2.22, 0.39};
throtB2s[6]={ 6, 6, 6, 14, 17, 18};

speedB7i[25]={ 55.09,56.32,57.04,57.41,57.45,56.68,55.94,55.21,54.4,53.18,
52.44,53.33,54.48,54.43,54.09,53.78,53.2,52.14,51.99,51.04,
47.64,44.64,36.81,27.76,18.18};
distB7i[25]={ 239.94,228.65,217.39,206.1,195.01,184.31,173.78,163.25,
152.74,142.39,132.19,122.39,111.89,101.36,90.77,80.19,
69.89,59.34,49.01,38.73,29.21,20.44,12.43, 6.31, 1.94};
throtB7i[25]={ 14, 14, 11, 11, 8, 7, 7, 6,
6, 4, 8, 13, 13, 8, 8, 8,
5, 5, 8, 1, -2, -8, -25, -25, 0};

speedB3[30]={ 47.59,46.29,44.98,43.21,42.78,42.12,39.65,35.22,31.75,27.93,
17.18, 7.47, 0.25, 0.23, 0.22, 0.2, 0.18, 0.17, 0.15, 0.13,
0.12, 0.1, 0.08, 0.11, 0.32, 1.03, 1.92, 2.08, 1.93, 1.78};
distB3[30]={ 64.6,58.43,52.45,46.68,41.08,35.55,30.19,25.25,20.91,16.99,
13.96, 12.4,11.96,11.93, 11.9,11.88,11.85,11.83,11.81,11.79,
11.78,11.76,11.75,11.74,11.71,11.63,11.45,11.18,10.92,10.68};
throtB3[30]={ 12, 12, 9, 9, 12, 8, 3, 0, 0, -33,
-64, -67, -54, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, 0, 0, 3, 3, -1, -1, -1};

```

```

speedB5[25]={ 44.1,41.38,38.29,36.76,36.39,36.97,37.87,38.76,39.52,40.17,
39.87,39.48,38.43,36.32,34.01,32.02,30.25,28.71,26.94,20.48,
14.93,10.42, 6.49, 2.91, 0.31};
distB5[25]={ 67.13, 63.7,60.16,56.87,53.79,50.82,47.61,44.55,41.43,38.41,
35.31,31.99,28.95,26.07,23.33,20.78,18.34,16.08,13.93,12.04,
10.69, 9.71, 9.06, 8.69, 8.59};
throtB5[25]={ 6, 1, 1, 6, 10, 15, 15, 16, 16, 13,
10, 9, 5, 0, 0, 0, 0, 0, -18, -47,
-47, -48, -48, -48, -38};

speedBi8[10]={ 15.3,15.12,20.29,23.23,22.54,16.22,15.15,18.23,20.36,22.67};
distBi8[10]={ 26.46,23.68, 20.2, 16,11.37, 7.5, 4.58, 1.3,-2.41,-6.65};
throtBi8[10]={ -3, 12, 14, 13, 3, -2, 11, 7, 11, 11};

speedBi2[10]={ 41.72,36.02,32.08,32.94,32.98,30.93,32.15,34.04, 35.4,35.57};
distBi2[10]={ 47.41,41.09,35.08,29.47,23.78,18.26,13.38, 7.75, 1.89,-4.15};
throtBi2[10]={ 4, 0, 5, 12, 4, 7, 14, 14, 11, 7};

speedBi3[10]={ 3.93, 7.49,11.05,14.15,16.62,16.49,15.56,15.77, 17.6,20.27};
distBi3[10]={ 12.23, 11.5,10.29, 8.69, 6.69, 4.52, 2.46, 0.46,-1.67,-4.17};
throtBi3[10]={ 16, 16, 12, 12, 9, 3, 3, 7, 12, 11};

speedB56[100]={38.62,41.17,44.05,46.59,48.82,50.23,49.53,48.92, 48.9,48.93,
49.11,50.17,51.69,52.43, 53.6,54.53,54.07, 53.1,52.28,51.57,
50.97,51.03,51.47,52.61,53.73,54.73,55.06,55.29,55.54,55.61,
55.64,55.67,55.68, 55.1,54.25,53.75,53.62,53.51,53.52,53.63,
53.85,54.07,54.33,54.63,54.93,55.16,55.41,55.61,55.82,55.98,
56.13,56.26,56.37,20.79,26.02,30.86,35.42,38.59,39.96,41.63,
43.34,45.21, 46.4,47.42,48.23,48.95,49.57,50.09,49.98,49.33,
50.08,51.55,52.88, 45.1,46.87,48.35,48.55,48.28,48.03,48.08,
48.26,48.42,53.53,54.16,54.58, 54.8,55.09,55.63,56.15,56.65,
56.93, 57.1,57.26,57.41,57.52, 57.5,57.25,56.91,56.57,56.25};
throtB56[100]={ 29, 32, 32, 29, 26, 18, 14, 14, 16, 16,
19, 25, 23, 21, 25, 21, 15, 15, 15, 15,
17, 18, 21, 23, 23, 23, 21, 21, 21, 20,
20, 20, 19, 16, 16, 18, 18, 18, 19, 19,
20, 20, 21, 21, 21, 21, 21, 21, 21, 21,
21, 21, 21, 29, 29, 29, 31, 28, 27, 20,
22, 22, 19, 19, 19, 19, 19, 18, 15, 14,
23, 23, 23, 30, 29, 24, 15, 14, 14, 17,
17, 17, 14, 14, 11, 10, 14, 14, 14, 14,
11, 11, 11, 11, 10, 9, 7, 7, 7, 7};

```



```

speedC2[30]={ 32.51,31.95,30.03,28.53, 28.6,27.85,28.11,28.81,29.41,29.79,
29.29,28.1,28.53,28.97,25.67,18.95,15.25,12.83,11.84,13.95,
12.55,10.92, 9.67, 7.91, 2.19, 0.08, 0.06, 0.04, 0.02, 0.01};
distC2[30]={76.52,72.04,67.67,63.54,59.63,55.75,51.95,48.14,44.29,40.32,
36.27,32.34,28.54,24.59,20.68,17.64,15.33,13.43,11.78,10.01, 8.17,
6.6, 5.21, 3.98, 3.29, 3.22, 3.22, 3.21, 3.2, 3.2};
throtC2[30]={ 18, 16, 13, 13, 15, 13, 15, 16, 16, 16,
13, 15, 15, 15, 0, 0, 0, 0, 6, 9,
0, 0, 0, -20, -44, -50, -50, -51, 0, 0};

speedC3[30]={ 45.89,45.87,45.86,45.85,45.83,45.82,45.81, 45.8,45.79,45.78,
45.78,45.77,45.76,45.76,44.73,41.66,39.04,36.78,34.72,32.96,
31.38,29.94,28.66,27.48, 26.4, 22.5,19.67,17.29,13.19, 8.67};
distC3[30]={ 78.44,75.42,72.45,69.39,66.43,63.43,60.42,57.46,54.38,51.41,
48.43,45.44,42.49,39.51,36.52,33.63,30.92,28.41,26.04,23.87,
21.79,19.79, 17.9,16.09,14.34,12.77,11.41,10.21, 9.18, 8.46};
throtC3[30]={ 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, -16, -51, -63};

speedC4[30]={ 47.35,47.32, 47.3,47.28,46.59,44.56,43.06, 41.2,40.34,39.92,
39.53,39.17,38.85,38.54,38.26, 38,37.83,37.71,37.67,37.67,
37.67,37.67,37.34, 35.4,33.69,32.16,30.78,30.93,32.32,34.16};
distC4[30]={ 77.41,74.57,71.73,68.89,66.06,63.15,60.49,57.93,55.42,53.01,
50.63,48.28,45.95,43.66,41.38,39.11,36.85,34.57,32.31,30.02,
27.8,25.56,23.29,21.11,19.06, 17.1,15.26,13.34,11.47, 9.54};
throtC4[30]={ 15, 15, 15, 15, 8, 6, 4, 3, 8, 9,
9, 9, 9, 9, 9, 9, 10, 10, 11, 11,
11, 11, 7, 0, 0, 0, 0, 13, 27, 28};

speedC5[30]={ 39.15, 37.3,35.91,34.79,34.04,33.54,33.28,34.22,35.74, 37.1,
37.96,39.18,38.63,35.49,33.13,30.66,28.57,26.64, 18.1,10.78,
4.84, 0.08, 0.06, 0.05, 0.04, 0.02, 0.01, 0.11, 2.29, 5.38};
distC5[30]={ 76.78,72.85,68.71,64.93,61.09,57.15,53.41,49.66,45.82,41.98,
38.14,34.01,29.93,25.94,22.42,19.09,16.03,13.24,10.89, 9.39,
8.58, 8.35, 8.34, 8.34, 8.33, 8.33, 8.33, 8.33, 8.21, 7.81};
throtC5[30]={ 2, 5, 5, 7, 7, 8, 11, 16, 17, 13,
16, 17, 1, 1, 1, 0, 0, -13, -46, -55,
-55, -55, 0, 0, 0, 0, 0, 5, 17, 20};

speedC6[30]={ 51.08,50.76,50.35,50.24,50.42,50.58, 50.5,50.11,49.77,49.86,
50.08,50.28,50.46,50.61, 50,45.38,39.88,31.21,21.39, 12.1,
3.93, 0.1, 0.09, 0.08, 0.07, 0.06, 0.05, 0.03, 0.02, 0.01};
distC6[30]={ 76.72,72.64,68.66,64.72,60.75,56.81,52.85,48.87,44.93,40.95,
37.07,33.08,29.11,25.12,21.19,17.42,14.03,11.13, 9.07, 7.78,
7.12, 7.03, 7.03, 7.02, 7.01, 7.01, 7.01, 7, 7, 7};
throtC6[30]={ 16, 16, 15, 18, 18, 18, 15, 15, 15, 18,
18, 18, 18, 18, 0, 0, -44, -100, -100, -100,
-100, -100, -99, -4, 0, 0, 0, 0, 0, 0};

speedC8[30]={ 21.05,21.43,23.04,24.55,25.38,25.67, 24.7,23.92,23.29,22.57,
22.03,23.41,25.45,26.41,26.81,25.55,24.46,23.87,23.85,23.32,
21.87,20.26,18.88,18.43,18.58,18.86,19.33,20.05, 20, 19.7};
distC8[30]={ 38.46,37.24,35.97,34.67,33.26,31.83,30.34,28.93,27.59,26.29,
25.12, 23.8, 22.5,21.05,19.51,18.11, 16.7,15.32,13.97,12.68,
11.38,10.25, 9.19, 8.11, 7.1, 6.09, 5.05, 3.89, 2.79, 1.7};
throtC8[30]={ 5, 16, 19, 17, 15, 9, 8, 8, 8, 7,
9, 23, 23, 16, 10, 7, 7, 11, 11, 7,
3, 2, 2, 7, 9, 9, 11, 11, 7, 7};

```

```

speedC3i[15]={ 12.29,17.17,21.23,18.04,15.47,13.55,12.06,10.94,12.05,14.51,
16.89, 19.36,22.45,23.97, 24.19};
distC3i[15]={ 6.05, 4.62, 2.71, 0.77,-0.82,-2.22,-3.45,-4.56,-5.65,-6.97,
-8.51,-10.28,-12.3,-14.6,-17.03};
throtC3i[15]={ 33, 29, 12, -1, -1, -1, -1, 0, 8, 13,
13, 17, 18, 13, 12};

speedC7i[15]={ 48.65,49.06, 49.4,49.67,49.16,48.04,42.84,37.36,37.13, 37.4,
34.07,32.16,31.13,28.33,28.04};
distC7i[15]={ 73.43, 66.6,59.76,52.96,45.95,39.14,32.57, 27.1,21.98,16.82,
11.9, 7.39, 3.03,-1.08,-4.97};
throtC7i[15]={ 16, 16, 16, 16, 13, 11, -2, 1, 13, 5,
-2, 5, 2, 1, 6};

speedC12[15]={ 51.58,49.71,48.97,49.04,49.33,49.61,50.06,49.48,49.03,48.67,
48.27,48.08,48.49,48.92,49.41};
throtC12[15]={ 13, 13, 16, 17, 17, 18, 18, 15, 15, 15,
15, 17, 17, 18, 18};

speedC34[10]={ 37.15,39.42,40.73, 43.5,44.59,44.95,45.39,45.95,46.38,46.66};
throtC34[10]={ 29, 28, 26, 26, 14, 17, 17, 18, 16, 18};

speedC56[10]={ 54.27,51.83,49.24, 47.8,46.66, 45.9,46.25,47.03, 47.9,48.49};
throtC56[10]={ 15, 10, 13, 13, 13, 14, 17, 17, 18, 18};

```

```

speedD1[30]={ 32.00,32.05,32.23,32.35,31.13,28.87,28.75,28.69,28.65,28.10,
26.51,25.37,24.6,23.91,22.73,21.25,20.05,20.43,20.74,20.22,
17.56,15.05,14.22,13.65,13.21,12.88,12.61,12.55,12.57, 12.6};
distD1[30]={ 77.18,73.39,69.63,65.87,62.14,58.69,55.43,52.23,48.94,45.63,
42.40,39.39,36.47,33.58,30.76,28.26,25.85,23.59,21.11,18.59,
16.47,14.64,12.99,11.31, 9.68, 8.19, 6.67, 5.25, 3.8, 2.35};
throtD1[30]={ 17, 17, 17, 17, 12, 13, 14, 14, 14, 11,
10, 10, 10, 9, 8, 4, 8, 9, 9, 6,
0, 1, 3, 3, 3, 3, 3, 3, 3, 3};

speedD2[30]={ 38.63,38.37, 38.1,37.85,37.64,37.46,37.29,37.13,36.98,36.84,
36.73,36.62,36.52,36.19,33.12,30.56,28.42,26.61,20.47,14.09,
9.4, 5.67, 3.64, 3.52, 3.41, 3.31, 3.22, 3.13, 3.04, 0.05};
distD2[30]={ 76.48,72.43, 69,64.91,60.87,56.89,52.96, 49,45.04,41.36,
37.55,33.71,29.91,26.07,22.45,19.12,16.04,13.19,10.76, 8.99,
7.78, 7.01, 6.57, 6.2, 5.85, 5.52, 5.19, 4.88, 4.57, 4.3};
throtD2[30]={ 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
10, 10, 10, 5, 0, 0, 0, 0, -19, -35,
-35, -30, 0, 0, 0, 0, 0, 0, 0, -74};

speedD3[30]={ 35.92,35.78, 35.7,35.66,35.63,35.62,35.61,35.46,29.27, 21.1,
19.01,22.56,24.54,22.06,17.27,17.95,18.97,19.16,15.58,11.01,
7.2, 2.2, 0.26, 0.24, 0.22, 0.2, 0.04, 0.02, 0, 1.93};
distD3[30]={ 75.66,70.92,66.26,61.59,56.93,52.29,47.68,43.05,38.48,34.96,
32.35,29.76,26.72,23.57,21.04,18.78,16.41, 13.9,11.69, 9.97,
8.8, 8.17, 8.09, 8.06, 8.03, 8, 7.85, 7.85, 7.85, 7.76};
throtD3[30]={ 20, 20, 20, 20, 20, 20, 20, 20, 0, 0,
18, 16, 14, 0, 0, 9, 9, 1, 0, -24,
-25, -43, -51, -66, -100, -100, -100, -100, -100, 17};

speedD4[30]={ 53.64,53.71, 52.3,49.31,45.61,41.88,38.86,36.71,35.52,34.86,
34.3,33.79,33.33,32.91,31.91, 30.1,28.29,23.56,19.84, 19.2,
21.14,22.96,24.47,25.75,26.41, 26.6,26.74,26.85, 27.5,30.08};
distD4[30]={ 76.72, 72.4,68.05,63.87,59.95,56.36,53.02,49.94,46.98,44.15,
41.43,38.68,36.01,33.32, 30.7,28.21,25.88,23.73,22.01,20.46,
18.87,17.11,15.26, 13.3,11.24, 9.19, 7.09, 5.04, 2.96, 0.72};
throtD4[30]={ 19, 19, 4, 7, 1, 1, 1, 4, 5, 7,
7, 7, 7, 7, 0, 0, -12, -33, 0, 15,
16, 16, 16, 16, 13, 13, 13, 13, 16, 33};

speedD5[30]={ 51.88,46.51,40.93,36.85,33.37, 32.4,33.51,35.01, 36.5,37.81,
38, 35.3,32.38,30.03,22.42,10.94, 4.4, 3.78, 3.79, 5.31,
6.92, 8.52,10.06,10.45, 9.5, 8.7, 6.33, 0.08, 0.06, 1.31};
distD5[30]={ 73.99,68.03,62.79,58.15,54.11,50.33,46.63,42.95,38.98,35.05,
30.89, 27.1,23.46,20.25, 17.4,15.64,14.91,14.51,14.13,13.67,
13.02,12.19,11.21,10.14, 9.12, 8.18, 7.35, 6.99, 6.96, 6.9};
throtD5[30]={ 7, 0, 0, 0, 0, 9, 16, 16, 16, 16,
7, 0, 0, 0, -87, -87, -33, -2, 6, 7,
7, 7, 7, -1, -1, -1, -40, -88, 2, 13};

speedD6[30]={ 55.06,55.04,55.01,54.99,54.96,54.94,54.81,53.76,52.75,51.77,
51.02,51.29,51.68,51.89,52.09,52.29,52.49,52.68,52.87,53.06,
53.24,53.65, 54.9, 56.3,57.68,59.03,59.95,60.14,60.09,60.03};
distD6[30]={ 77.96,75.05,72.11,69.21, 66.1,62.92,60.09,57.23, 54.4,51.64,
48.9,46.13,43.42,40.77,38.13,35.44,32.71,29.96,27.17,24.42,
21.68,18.92,16.08,12.93, 9.73, 6.48, 3.41, 0.01,-3.53,-7.07};
throtD6[30]={ 10, 10, 10, 10, 10, 10, 3, 0, 0, 0,
8, 15, 15, 12, 12, 12, 12, 12, 12, 12,
12, 28, 36, 36, 36, 32, 19, 11, 11, 11};

```

```

speedD7[30]={ 38.69,38.52,38.37,38.22,38.09,37.96,37.84,37.73,37.63,37.53,
37.57,37.94,38.35, 38.8,39.23,39.64,40.01,40.37,40.72,41.05,
41.36,41.65,41.92,42.18,42.43,42.51,42.73,42.94,43.03,42.86};
distD7[30]={ 73.86,71.71,69.53, 67.4,65.29,63.17,61.09, 59,56.92,54.85,
52.79,50.73,48.63,46.55,44.42,42.27,40.13, 38,35.86,33.58,
31.31,29.01, 26.7,24.37,21.89, 19.5,17.11,14.74,12.35,10.11};
throtD7[30]={ 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
13, 14, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 12, 12};

speedD8[30]={ 38.4,35.81,33.29,29.65,25.16,19.38,16.05,14.11,14.09, 16.5,
19.02,21.49,23.35,23.94,24.16,20.09,17.16,15.07,14.77, 16,
16.79,17.85,18.72, 19.5,18.98,16.47,14.48,14.13,15.69,17.13};
distD8[30]={ 47.17,44.21,41.36,38.59,36.19,34.19,32.63,31.35,30.15,28.96,
27.47,25.75,23.87,21.92,19.88, 18,16.43,15.08,13.87, 12.6,
11.23, 9.72, 8.25, 6.62, 4.95, 3.53, 2.23, 1.03,-0.22,-1.59};
throtD8[30]={ 0, 0, -10, -27, -27, -16, -4, -3, 12, 15,
15, 15, 13, 10, 6, -3, -2, -1, 9, 9,
9, 9, 9, 9, -1, -2, -2, 9, 10, 10};

speedD3i[15]={ 2.72, 4.48, 6.34, 8.33,10.43,12.58,13.81, 12.8,11.91,11.16,
11.43,12.35,13.59,14.76,15.88};
distD3i[15]={ 7.69, 7.46, 7.11, 6.64, 6.04, 5.3, 4.42, 3.57, 2.77, 2.03,
1.31, 0.55,-0.29,-1.19, -2.2};
throtD3i[15]={ 17, 18, 18, 18, 18, 18, 3, 0, 0, 0,
7, 10, 11, 11, 11};

speedD7i[15]={ 56.65,57.53, 56.8,55.71,54.59,53.45,51.76,49.85,48.05, 46.4,
44.89,43.49,42.17,37.81,32.29};
distD7i[15]={ 75.29,68.81,62.22, 55.9,49.91,43.86,38.02,32.44,26.97,21.73,
16.71,11.91, 7.21, 2.68,-1.15};
throtD7i[15]={ 18, 14, 5, 5, 5, 4, 0, 0, 0, 0,
0, 0, 0, -29, -1};

speedD12[15]={ 45.78,48.76,51.13,53.01,52.04,50.42,50.47, 50.5,50.53,50.55,
50.56,50.57,50.58,51.11,51.96};
throtD12[15]={ 22, 22, 22, 22, 15, 17, 17, 17, 17, 17,
17, 17, 17, 19, 20};

speedD34[15]={ 36.83,38.76, 40.3,42.29,44.44, 46.5,48.47,50.42,52.29,53.17,
53.56,53.87,54.16,54.33,54.01};
throtD34[15]={ 30, 30, 30, 30, 30, 30, 30, 30, 30, 26,
22, 22, 22, 20, 17};

speedD56[15]={ 70.81,68.99,67.09, 65.3,63.76, 62.3,60.92,59.59,58.33,57.19,
56.39,55.94,55.94,55.96,55.97};
throtD56[15]={ 0, 0, 0, 0, 0, 0, 0, 0, 0, 3,
3, 9, 11, 11, 11};

```

```

speedE1[30]={ 24.08,23.08,26.08,29.45,31.35,31.01,30.36,28.89,28.57,28.99,
27.48,24.69,23.73, 23.4,23.15,21.83,20.93,20.29,20.11,20.67,
18.41,13.44,10.94, 9.4,13.11,18.88,20.23,14.74, 6.12, 2.33};
distE1[30]={ 77.55,74.72,72.01,68.96,65.44,61.84,58.31,54.87,51.51,48.18,
44.75,41.61,38.74,35.93,33.16,30.44,27.91,25.59,23.18,20.71,
18.29,16.42,14.96,13.72,12.51,10.68, 8.34, 6.4, 5.17, 4.79};
throtE1[30]={ 2, 14, 19, 19, 16, 14, 13, 11, 14, 14,
6, 6, 9, 9, 9, 6, 6, 6, 8, 8,
-38, -35, -32, -10, 29, 25, 1, -48, -72, 4};

speedE2[30]={ 31.8,31.44,31.17,30.95,30.77,30.63,30.52,30.43,30.87,31.48,
31.97,32.37,32.68,32.94,33.14,33.29,33.41, 33.5,33.57,33.62,
33.66, 33.7,33.74,33.78,33.61, 33,32.87,33.37,34.19,35.24};
distE2[30]={ 73.52,71.34,69.15,67.01,64.82,62.73, 60.6, 58.5,56.38,54.23,
52.07,49.89,47.66,45.45,43.18,40.94, 38.8, 36.5,34.35,32.07,
29.8,27.49,25.19,22.85,20.53,18.27,16.01,13.76, 11.4, 9.06};
throtE2[30]={ 3, 3, 3, 3, 3, 3, 3, 3, 3, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 4, 4, 6, 8, 10, 12};

speedE3[30]={ 47.46,47.08,46.85,46.45,45.89,45.37,44.85,44.32,43.83,43.38,
42.96,42.57,42.56,42.37,41.93,41.48,41.01,40.56,40.12,39.27,
37.72,36.42,32.21,27.15,22.55,18.69,15.72,12.27, 8.25, 4.18};
distE3[30]={ 78.27,75.18, 72.1,69.01,65.99,62.98,59.98,57.06,54.18,51.35,
48.52,45.76,42.94,40.14,37.39,34.65,31.97,29.34,26.73,24.14,
21.59,19.19, 16.9,14.94,13.34,12.01,10.86, 9.94, 9.27, 8.88};
throtE3[30]={ 2, 3, 4, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 3, 1, 0, -1, -1, -1, -1, -19,
-19, -27, -84, -85, -82, -80, -66, -84, -84, -84};

speedE4[30]={ 45.43,44.93,44.17,43.02,41.43,39.85,37.75,35.12, 32.9, 31.6,
30.48,29.49,28.58,27.77,26.54,21.79,18.61,16.83,16.48,16.33,
16.21,16.42, 16.6,17.88, 20.9,23.98,26.78,28.61,30.11,31.29};
distE4[30]={ 77.05,73.43,69.76,66.23,62.76,59.49,56.25,53.29,50.46,47.87,
45.36,42.92,40.57,38.29, 36.1,34.05,32.37,30.91,29.57,28.26,
26.98,25.68,24.39,23.02,21.51,19.75,17.76, 15.6, 13.3,10.92};
throtE4[30]={ 15, 12, 9, 8, 6, 5, 0, 0, 0, 0, 3,
3, 3, 3, 3, 0, 0, 0, 3, 5, 5,
5, 8, 5, 17, 22, 22, 22, 20, 20, 20};

speedE5[30]={ 43.89,42.99,41.48,37.92,34.88,34.27,35.05,35.35, 35,34.26,
33.56,32.77, 31.3,29.46,27.37,21.14,14.96, 8.8, 5.67,13.25, 7.16,
1.21, 0.12, 1.6, 3.68, 6.1, 6.38, 9.18,12.25,15.47};
distE5[30]={ 75.57,70.53, 65.6,60.79,56.43,52.28,48.36,44.54,40.42,36.74,
33.23,29.66,26.23,23.04,20.07,17.46,15.62,14.42, 13.7, 8.14,
7.1, 6.65, 6.62, 6.09, 5.82, 5.32, 13.1,12.27,11.19, 9.77};
throtE5[30]={ -11, -8, -12, -26, -21, -3, 2, -6, -9, -10,
-10, -13, -15, -22, -45, -82, -81, -81, -26, -70,
-71, -82, -81, 19, 20, 17, 13, 16, 12, 9};

speedE6[30]={ 49.68,49.77,49.86,49.99,50.14,50.29,50.41,50.54,50.66,50.76,
50.85,50.94,51.02, 51.1,51.16,51.23,51.29,51.35, 51.4,51.45,
51.24,50.94, 50.6,50.26,49.93,49.56,49.18,48.83,48.53,48.24};
distE6[30]={77.97,75.39, 72.8,70.21,67.57, 65,62.16,59.53,56.68,53.82,
51.15, 48.5,45.89, 43.2,40.56,37.67, 35,32.31,29.41,26.69,
24.02, 21.4,18.79,16.17,13.34,10.77, 8.01, 5.27, 2.78, 0.1};
throtE6[30]={ 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
14, 14, 13, 13, 13, 13, 13, 13, 13, 13};

```

```

speedE7[30]={ 37.14,36.52,36.99,38.14,38.87,38.69,38.28,37.79,37.67,37.89,
38.05,38.19,38.3,37.64,36.98,36.45,36.06,35.73,35.47,35.27,
35.69,36.33,36.83,37.22,37.53,37.41,36.98,36.66,36.52,36.41};
distE7[30]={ 72.91,70.78,68.73,66.65,64.49, 62.3,60.16,58.03,55.95,53.92,
51.83,49.76,47.71,45.61,43.61,41.58,39.64,37.74,35.74,33.73,
31.76,29.76,27.59,25.53,23.47,21.37, 19.3,17.34,15.26,13.21};
throtE7[30]={ 17, 20, 24, 26, 22, 21, 19, 19, 22, 22,
22, 22, 22, 18, 18, 18, 18, 18, 18, 18,
21, 22, 22, 22, 22, 19, 19, 19, 19, 19};

speedE3i[15]={ 7.19, 10,13.16,16.71,20.07,21.91,23.01,23.86,24.55,25.58,
26.56, 27.38,27.43,26.65,26.74};
distE3i[15]={ 7.21, 6.65, 5.9, 4.94, 3.73, 2.35, 0.9,-0.64,-2.19,-3.82,
-5.5,-7.27,-9.08,-10.86,-12.62};
throtE3i[15]={ 29, 30, 37, 34, 25, 15, 14, 14, 14, 17,
17, 17, 10, 13, 14};

speedE7i[15]={ 46.25,46.94,46.78,46.42,45.73,42.49,40.22,39.29,37.65,35.06,
31.33,28.32,24.82,25.44,28.76};
distE7i[15]={ 74.23,67.76,61.24,54.76,48.28,42.03,36.41,30.79,25.49,20.39,
15.85,11.77, 8.08, 4.71, 0.92};
throtE7i[15]={ 16, 13, 13, 13, 10, 3, 7, 8, 5, -2,
-2, -2, 3, 20, 15};

speedE12[15]={ 31.07,43.98, 53.2,58.62,58.03, 57.2,58.09,57.35,60.46,56.49,
55.49, 55.2,55.27,55.54,55.71};
throtE12[15]={ 36, 29, 27, 24, 20, 21, 21, 19, 15, 18,
19, 19, 19, 19, 19};

speedE34[10]={ 26.74,27.59, 30.7,34.49,36.79,37.42,39.12,40.12,42.65,44.73};
throtE34[10]={ 14, 18, 28, 28, 24, 23, 27, 27, 24, 22};

speedE56[15]={ 58.12, 58.4,58.96,59.56,60.12,60.47,60.73,60.92,61.09,61.25,
61.4,61.46,61.45, 61.3,61.09};
throtE56[15]={ 14, 14, 18, 18, 18, 16, 15, 15, 15, 15,
14, 14, 13, 12, 12};

```

APPENDIX E: EVOLVED CODE

This appendix lists all evolved source code generated during this research. The code is difficult to read but it is syntactically correct. The two main reasons that make it unreadable is: 1) GP produces non-coding regions, 2) The tree structure of the individuals makes it most convenient to produce a single line source code. Hence, the code is not structured or well organized.

The code assumes that the following variables are used:

- speed - current speed of the agent
- distance - distance to either an intersection or a traffic light
- intersection – intersection present
- lightPresent – traffic light present
- Light- enumeration variable describing the light (GREEN, YELLOW or RED)

The results from all the action rules are a pedal pressure value (positive = throttle pressure, negative = brake pressure). The result from the sentinel rules are a Boolean result (either true or false) that indicates whether the context should request activation or not.

Two issues that need the attention of the function set used in this research are correct division and logarithmic functionality. Division with 0.0 results in an error and the natural logarithm does not allow the parameter to be 0.0. Hence, to prevent those situations from occurring the division and logarithmic functions need to be customized to enable all possible individuals to be

executed. The custom functions are described below:

```
double slog(double x) //Safe log from GP-generated code
{
    return (x==0.0?0.0:log(x));
}

double sdiv(double t, double n) //Safe division from GP-generated code
{
    return (n==0.0?1.0:t/n);
}
```

Intesection Turning, Sentinel Rules

```
//***** Agent A (2.3244)
(intersection?(47.776726<distance?0:(lightPresent?(distance<9.543138?(speed<d
istance?0:(lightPresent?92.892239:1)):0):1)):0);

//***** Agent B (3.0400)
(distance<52.140873?(distance>23.368023?(intersection?(intersection?1:0):(!li
ghtPresent?0:0)):lightPresent?(lightPresent?(!lightPresent?(speed<73.000274?
(!intersection?0:(lightPresent?1:(!intersection?(!intersection?(distance>93.1
76061?0:(intersection?(!lightPresent?1:0):0)):speed>3.933836?1:(!interseccio
n?1:(!lightPresent?0:0)))):1)):lightPresent?0:(distance<96.331675?0:(!light
Present?(lightPresent?(!intersection?(speed<8.340709?(!lightPresent?1:0):1):(
speed>43.702505?1:(!lightPresent?0:0)):0):1)):lightPresent?0:0):(!lightP
resent?(speed<83.849605?1:0):(lightPresent?1:(distance<3.219703?1:(!intersec
tion?1:(intersection?(distance>73.912778?1:0):(distance>91.772210?(!intersecti
on?(!intersection?(speed<1.001007?1:0):(distance<80.715354?0:0)):1):0))))):
(speed>12.320322?(!lightPresent?(speed<26.636555?1:(lightPresent?(speed<99.76
1955?0:(lightPresent?(!lightPresent?0:(lightPresent?(speed>48.048341?1:0):(!l
ightPresent?(speed<18.448439?0:1):0)):speed<87.112033?(!intersection?0:(dis
tance>45.661794?(lightPresent?1:1):0):1)):(!intersection?(distance>32.34962
0?1:(intersection?0:0)):intersection?(!intersection?1:(speed<87.343974?(ligh
tPresent?(!intersection?1:0):(!intersection?1:0)):lightPresent?(!lightPresen
t?0:0):0)):lightPresent?(!intersection?0:(!lightPresent?(intersection?1:1):
1)):1))))):(!intersection?1:(speed<29.490036?(intersection?0:0):(!lightPresen
t?(!intersection?1:(lightPresent?(distance<92.779320?(lightPresent?0:(speed>1
0.983612?0:0)):0):(!intersection?(distance>47.959837?1:0):0)):(!lightPresent
?1:0))))):0)):distance>77.828303?0:0));

//***** Agent C (3.3433)
(lightPresent?(distance>35.319071?(lightPresent?(speed>35.319071?(distance<98
.083437?(!intersection?0:(speed<distance?0:1)):(!intersection?(speed<98.08343
7?0:1):(speed>14.383373?(speed>57.991272?(!intersection?(!lightPresent?1:0):0
):(!lightPresent?0:1)):1)):lightPresent?0:(!intersection?0:1)):speed>50.5
32548?0:(distance>50.215155?0:(intersection?1:0))))):lightPresent?0:(!interse
ction?0:1)):speed>50.532548?0:(distance>50.215155?0:(intersection?1:0))));

//***** Agent D (2.9296)
(lightPresent?(intersection?(distance<31.592761?0:1):0):(intersection?(distan
ce>42.921232?(intersection?(!intersection?(!lightPresent?1:(!lightPresent?1:(
!lightPresent?1:1))):(!lightPresent?(speed>64.461196?(!intersection?1:1):0):0
)):lightPresent?(speed>98.919644?1:(!lightPresent?(speed<7.724235?1:0):(inte
rsection?1:1)):0)):(!intersection?(!lightPresent?1:(speed<0.192267?0:0)):di
stance>40.513321?0:1)):0));

//***** Agent E (4.3173)
(distance>30.869472?(distance<63.628040?(distance<61.311685?(distance<63.6280
40?(distance<61.311685?(intersection?56.413465:(distance>31.998657?0:(31.9986
57>32.996612?0:(lightPresent?(speed<23.789178?0:1):(speed<20.917386?1:(32.996
612>distance?0:0))))):0):0):0):lightPresent?(speed<30.442213?0:(intersec
tion?(distance>56.413465?0:0):1)):speed<20.917386?(speed<distance?0:1):0));
```

Intersection Turning, Action Rules

```
//***** Agent A (2.3316)
(((speed-(((sin((exp(sdiv(59.794915,(((sin((exp(sdiv(59.794915,sdiv(59.794915
, speed)))))-cos(77.013458))>cos(speed)?speed:(distance*sdiv(-12.991730,s
peed)))))))-cos(77.013458))>cos(speed)?speed:(distance*sdiv(85.735648,sp
eed))))*(speed<-37.846004?(exp(((exp(speed))*distance)):(distance>38.279366?
speed:distance))>distance?cos(speed):(((sin((77.013458>4.019287?distance:5
4.033021))>speed?(exp((sin((distance>4.019287?cos(speed)):54.033021))))):((
(sdiv((distance>speed?distance:(distance>(distance-(exp(sdiv(59.794915,-39.9
21262)))<speed?cos(sdiv((-61.760308-(-39.921262)),distance)):(cos(speed)))
?distance:54.033021),distance)>speed?(sin(speed)):(distance<sdiv(90.514847,1
4.004944)?(((sin((sin(4.019287))))+(speed>36.771752?((distance*4.019287)+(5.7
46635<distance?speed:-99.877926)):(distance<distance?(speed>speed?((-33.4757
53-(-95.312357))-cos(5.746635))):92.382579):sdiv(-75.292215,distance))-spee
d<speed?sdiv(distance,26.273384):speed)))-((speed>speed?(exp(distance)):(dis
tance<sdiv(-58.610798,14.004944)?(distance-sin(sin(4.019287))))):exp((cos(37.
571337)))))-(-11.301004)):exp((cos(((speed>89.135410?sdiv(63.267921,-90
.087588):(distance<37.571337?distance:90.234076))>speed?(sin(distance)):59.64
2323)))))-(-11.301004)<4.019287?(((sin((sin(4.019287))))+(14.004944>52.6
96310?sdiv(sdiv(-58.610798,14.004944),14.004944):(distance<distance?(speed>s
peed?34.360789:92.382579):sdiv(-75.292215,distance))-(speed<speed?sdiv(distan
ce,26.273384):speed))>speed?(exp((exp((sin(4.019287))))):(distance<sdiv(-5
8.610798,speed)?(distance-(speed>speed?speed:(distance<sdiv(-58.610798,14.00
4944)?(sin((sin(4.019287))))-sin(-70.537431))):exp((cos(37.571337)))))-(-
11.301004)):exp((cos(((speed>89.135410?sdiv(63.267921,-90.087588):(distance
<37.571337?distance:90.234076))>speed?(sin(distance)):59.642323)))))-(-11.3
01004))-((speed>speed?(exp(distance)):(distance<sdiv(-58.610798,14.004944)?(d
istance-sin(sin(4.019287))))):exp((cos(37.571337)))))-(-11.301004)):exp
((cos(((speed>89.135410?sdiv(63.267921,-90.087588):(distance<37.571337?distan
ce:90.234076))>speed?(sin(distance)):59.642323)))))-(-11.301004));

//***** Agent B (1.5804)
(exp((slog((12.063967>(exp((speed<30.100406?(cos(((sin(speed))<sin(94.158757
))?(speed<15.878780?(cos(speed)):cos((sin(80.281991))))):sin((exp(19.849239
))))))):exp((distance>49.851985?(exp((cos(distance))):cos((slog(21.610156)
)))))))?(speed>7.812738?(cos(((sin(speed))<(exp((sin((cos(91.366313)))))))?
(speed<28.315073?(sin((slog((slog((distance>distance?(exp((slog(speed)))):(s
log(91.051973))))>distance?speed:(distance>98.681600?27.106540:(sin((exp(dis
tance))))))):cos((sin((slog((cos((speed>distance?(speed>48.884548?(sin(15
.424055)):69.777520):distance))))))):sin((exp(19.849239))))))>73.989074?
(slog(speed)):exp((exp((distance<30.100406?(cos(((sin(speed))<(exp((sin((cos
(91.366313))))))?(speed<28.315073?(cos(speed)):cos((sin(distance))))):sin((
exp(19.849239))))))):exp((distance>49.851985?(exp((cos(distance))):cos((sl
og(21.610156)))))))))?16.855373):16.855373)))));

//***** Agent C (1.6308)
((sin(73.021637))<distance?pow((distance<speed?(pow(speed,5)>(33.744316>(dist
ance<speed?speed:71.636097)?pow(pow(distance,3),3):pow(59.166234,2))?(distan
ce>speed?cos(-52.122562)):sin(pow(distance,5))):13.791314):speed,2):(log(7
2.826319)));

//***** Agent D (1.1705)
```

```
(distance<5.307168?0.845362:(53.251747<8.786279?10.885952:(33.988464<(27.225562<distance?41.361735:speed)?10.885952:18.491165)));
```

```
//***** Agent E (3.3490)  
(49.546800>(distance<92.010254?(63.054292<speed?-11.154515:distance):(exp(distance))?(speed>19.833979?(-58.287301<11.978515?(-12.820826>-13.620411?((63.054292<speed?-92.980743:distance)<4.818872?(63.054292<(speed>71.391949?speed:(cos(exp(cos(12.717063))))))>-30.368969?(distance<28.922391?(speed<distance?18.198187:(exp(cos(speed))))):distance):speed)?speed:(speed>41.660817?73.699148:distance):70.055238)>13.162633?6.344798:14.810632):(sin(-39.451277))):-8.133183):39.481795):14.242988);
```

Red Light Driving, Action Rules

```
//***** Agent A (1.3194)  
(sdiv((slog(pow(((distance>speed?(speed<((distance>43.296609?(cos(-88.860744)):pow(speed,4))<89.434491?(speed<distance?(sin((slog(cos(-7.663198))))*(cos(speed))<0.222785?speed:pow(distance,5))))*(speed<0.222785?speed:pow(distance,5))):sdiv((exp(exp(-55.180517))) * speed), pow(speed,5))+(distance+sdiv(cos(pow(speed,3))), (distance<speed?(sin(distance)):speed))))):pow(speed,2))?(speed>(distance>36.423841?distance:86.901455)?speed:(slog((speed<speed?(pow(distance,4)*(pow(18.765831,4)*(cos(speed)))):(pow(distance,4)*(pow(18.765831,4)*(cos(speed))))):86.901455):(cos(14.255195))<speed?speed:(slog(pow((speed>(distance>36.423841?(sdiv(52.696310,-30.423902)+distance):86.901455)?speed:(slog((speed<speed?(slog(-38.035219)):(pow(distance,4)*pow(53.770561,5)))))<speed?speed:-9.750664),4))))),4)), (speed<distance?(slog(pow(speed,4))>(slog(pow(12.381359,3)))?(-54.173406+(distance<53.770561?distance:46.739097)):(slog((speed>(slog(pow(slog(pow(speed,4))),3)))? (44.889675+(cos((speed>3.848384?81.670583:-59.685049)))):(slog(speed))))):cos(sdiv((exp(cos(-76.476333))), ((distance*(cos(-76.476333)))-cos(sin((distance>(slog(pow(distance,4)-distance)))?(speed>distance?73.839533:16.922513):(speed<-31.113621?-6.833095:65.233314)))))))*sdiv(distance,(speed>(distance>speed?pow(distance,4):(cos(14.255195)))?-86.938078:-35.404523)));
```

```
//***** Agent B (1.2277)  
(speed<distance?(((pow(distance,3)*(cos(pow(distance,4))))>distance?-87.200537:8.883938)<distance?(speed<distance?(exp((distance<68.871120?-25.284585:pow(distance,3)))<distance?(speed<distance?(sdiv(speed,(34.440138*(cos(pow((speed<speed?(distance<94.787438?(slog(exp(speed))))):15.347758):speed)-pow(distance,4),3))))<distance?(speed<distance?(sin(sin((slog(pow((speed+-0.326548),4)*distance)))<speed?(sin((exp(sdiv(speed,distance)))<speed?(15.347758+(speed<distance?speed:-97.399823))<distance?(cos(-99.554430)):(slog(pow(speed,3))))):-44.730980)):-44.730980))):-44.730980):-44.730980):-44.730980):-44.730980):-44.730980):-44.730980);
```

```
//***** Agent C (6.0419)  
(((pow((((13.022248/(speed/((pow((distance/speed),3)+distance)/distance)+7.858516)))-66.252632)/distance)-66.252632)/distance),3)+distance)/distance)+7.858516);
```

```
//***** Agent D (5.3956)  
(slog(pow(sdiv(pow(sdiv(sin(sdiv(pow(distance,5),distance))),cos(sdiv(sdiv(-78.991058,(sin(distance))-distance),(cos(exp(distance))))-distance))))
```

```
-distance), 4), sdiv(speed, sdiv((slog((exp((cos(distance))))), pow(speed, 3))), 4));
```

```
//***** Agent E (6.0822)  
(57.292397<(distance<-30.259102?-38.975189:distance)?(slog(pow(((32.316049<speed?59.044160:76.281014)<distance?(slog((71.111178<distance?(pow(98.748741, 4)<pow(speed, 5)?(cos(pow((cos(-55.113376)), 5)):38.303781):distance))):-82.641072), 4)): (19.895016<distance?(slog((38.303781<(57.292397<(slog(pow(distance, 4)))? (slog(pow(71.111178, 2))): ((slog(38.303781)<distance?distance:pow((19.895016<(exp(pow(98.748741, 2)))?59.044160:(exp(71.111178)), 3)))? (slog(pow(distance, 2))): (19.895016<distance?(slog((38.303781<speed?speed:(19.895016<71.111178?(slog(38.303781)): (exp(pow(98.748741, 2))))<distance?(slog((19.895016<distance?(slog((speed<distance?(19.895016<speed?(-30.259102<speed?(exp(pow(-99.755852, 2))):speed):(sin(speed))):19.895016))):-82.641072))):-82.641072))):-82.641072))):-82.641072));
```

Green Light Driving, Action Rules

```
//***** Agent A (1.6736)  
(distance>(speed<distance?64.745017:43.082979)?(-7.467880<speed?(distance>distance?distance:79.436628):61.522263): ((distance>56.944486?61.522263:distance)>22.653889?-32.206183:72.203741)<22.653889?8.401745:(distance>-22.989593?((speed>56.944486?-85.308390:speed)>22.653889?-32.206183:72.203741)>22.653889?8.401745:(distance>-33.738212?30.600909:-90.289010)):56.944486));
```

```
//***** Agent B (1.2867)  
((distance<6.369213?distance:-46.604816)>-63.109226?((13.736381+sdiv((distance<24.753563?speed:(-90.997040+(speed<distance?pow(6.369213, 2):48.008667))), speed))+sdiv(((7.718131>72.801904?speed:48.319956)>speed?-12.552263:79.180272), (speed-40.964995)):speed);
```

```
//***** Agent C (2.0707)  
(9.872737>distance?(-0.686667<((speed>70.958586?distance:speed)*((speed>5.734428?57.475508:speed)-(distance>distance?16.000854:((-39.786981<speed?28.940702:-66.447951)+speed)>distance?-44.932402:-44.511246))>-34.104435?speed:distance)<speed?39.188818:((speed<speed?-80.309458:(-84.783471>1.431318?-11.447493:speed))-34.849086)?27.939695:27.341532):(distance<(-14.792322+(-0.686667+(-58.366649+97.515793)))?(35.850093-speed):15.274514));
```

```
//***** Agent D (3.1802)  
6.344798<(speed<((10.965300+distance)>36.271248?10.965300:(cos((distance*(4.141361+36.271248))))?((speed+10.965300)>36.271248?28.397473:26.615192):distance)?(((10.965300-(cos(((10.965300-sdiv(speed, (speed>36.271248?10.965300:((cos(speed)<speed?speed:distance)<distance?speed:speed)* (sin(speed))))))+distance)>36.271248?10.965300:(cos((distance*(4.141361+36.271248))))+speed))) -sdiv(speed, (speed>36.271248?(-5.258339<distance?(10.965300-(cos((-71.245460+(distance<30.851161?8.426160:-38.633381)))))-sdiv(speed, (speed>(10.916471+36.271248)?10.965300:-67.247536)):speed):-72.972808))-cos((10.916471+speed)))-sdiv(speed, (speed>36.271248?10.965300:-72.972808)):speed)-cos(((distance+distance)>speed?-71.233253:distance)<distance?distance:-90.484329)))-cos((10.965300+speed)))-sdiv(speed, (speed>36.271248?10.965300:-72.972808))));
```

```
//***** Agent E (2.5500)
```

```
( (pow(speed, 2)>sdiv(-51.506089, -93.292032)?(cos(((cos(distance))<distance?pow(distance, 4):(17.557298+-33.469650))>sdiv((pow(distance, 4)>speed?39.139988:pow(distance, 5)), (-5.093540<67.052217?sdiv(distance, speed):pow(distance, 3)))? -46.183661:pow(speed, 2))>( (exp(speed))>distance?(12.424085>(slog(pow(distance, 4)))? -58.568072: speed): speed)?(12.424085>(slog(pow((pow(-53.856014, 5)<pow(speed, 2)?(slog((speed+pow((exp(sdiv((exp(sdiv((sin((cos(sdiv(-51.225318, distance))))), pow(speed, 3))))), (sin((slog(distance))))), 2))))): (speed+-61.912900)), 4)))? -58.568072: speed): speed)): pow(distance, 3))*-46.519364);
```

Traffic Light Driving, Action Rules

```

//***** Agent A (4.3729)
(distance>-3.299051?(speed<43.784905?((distance>-22.513505?(speed>(Light!=YELLOW? (43.784905-(distance<speed?(redLight(distance, speed)+(Light!=GREEN?redLight(distance, speed):(distance<86.492507?(distance>-20.694602?(distance>-43.168432?greenLight(distance, speed):(distance<-88.488418?(speed>-85.216834?(distance>speed?redLight(distance, speed):greenLight(distance, speed)):(distance<-2.395703?greenLight(distance, speed):redLight(distance, speed))):greenLight(distance, speed)):(distance<-9.671316?(sdiv(redLight(distance, speed), (13.840144+32.016968))-11.465804):redLight(distance, speed)):(speed>speed?redLight(distance, speed):(Light==GREEN?(Light!=GREEN?sdiv(redLight(distance, speed), (speed+56.907864)):(Light!=GREEN?greenLight(distance, speed):(Light!=YELLOW?speed:redLight(distance, speed))):greenLight(distance, speed))):redLight(distance, speed))):redLight(distance, speed)?(Light!=GREEN?(Light!=YELLOW?(distance*(speed<18.631550?greenLight(distance, speed):43.784905)):redLight(distance, speed))-(Light!=GREEN?(Light==RED?(speed<-48.496964?redLight(distance, speed):redLight(distance, speed)):(speed<10.238960?-62.688070:greenLight(distance, speed)):(Light!=YELLOW?greenLight(distance, speed):greenLight(distance, speed))): (speed>distance?(Light==GREEN?(distance-(Light==GREEN?redLight(distance, speed):distance)):redLight(distance, speed)):(speed<speed?(Light==RED?-22.574542: speed)*distance):(Light==RED?greenLight(distance, speed):(Light==RED?greenLight(distance, speed):speed)):sdiv(((speed<distance?(distance+45.591601)+83.434552):(distance>-25.144200?greenLight(distance, speed):(distance<-54.960784?greenLight(distance, speed):redLight(distance, speed)))-distance), ((15.189062*(Light!=RED?redLight(distance, speed):(speed<-71.880245?redLight(distance, speed):redLight(distance, speed)))+(distance-(speed>63.145847?(-81.456954+-35.740227):greenLight(distance, speed))):((Light!=GREEN?(speed<76.799829?sdiv(76.317636, greenLight(distance, speed):(distance>-94.421216?redLight(distance, speed):greenLight(distance, speed)):greenLight(distance, speed))+sdiv(greenLight(distance, speed), -56.315806))<41.581469?(Light!=YELLOW?(Light==YELLOW?greenLight(distance, speed):( (speed>speed?(speed<71.654408?(Light==YELLOW?(speed<13.730277?(speed>speed?(Light!=YELLOW?greenLight(distance, speed):0.204474):(distance+-57.603687)):(Light!=YELLOW?(Light!=GREEN?speed:45.689260):(distance*speed))):redLight(distance, speed):(speed>92.742698?greenLight(distance, speed):(speed<distance?(distance>24.485000?redLight(distance, speed):redLight(distance, speed))+greenLight(distance, speed)):(speed<98.315378?(Light!=RED?redLight(distance, speed):-16.989654):redLight(distance, speed))):((Light!=YELLOW?(Light==GREEN?(distance<distance?(Light==YELLOW?-61.577197:-64.854885):(distance<distance?redLight(distance, speed):redLight(distance, speed)):(Light==RED?(Light==YELLOW?distance: speed):redLight(distance, speed)):(speed*-40.964995))- (Light==YELLOW?(speed>-91.735588?greenLight(distance, speed):sdiv(redLight(distance, speed), sdiv(91.351054, -39.982299))):((distance>-67.308573?greenLight(distance, speed):greenLight(distance, speed))-33.237708)-(speed>65.691091?(speed-speed):redLight

```

```

t (distance, speed) ) ) ) ) + (speed < distance ? (Light == GREEN ? redLight (distance, speed)
: redLight (distance, speed) ) : redLight (distance, speed) ) < distance ? (distance < dist
ance ? sdiv ( (speed < 53.691213 ? redLight (distance, speed) : redLight (distance, speed) )
, (Light != GREEN ? speed : greenLight (distance, speed) ) ) : ( (Light != YELLOW ? redLight (di
stance, speed) : redLight (distance, speed) ) * (Light != RED ? greenLight (distance, speed)
: speed) ) : greenLight (distance, speed) ) - speed) : ( (Light != GREEN ? sdiv (greenLight
(distance, speed), redLight (distance, speed) ) : greenLight (distance, speed) ) - speed)
) < 41.581469 ? (Light != GREEN ? (Light != YELLOW ? redLight (distance, speed) : ( (Light != GR
EEN ? sdiv (greenLight (distance, speed), redLight (distance, speed) ) : greenLight (dist
ance, speed) ) - speed) ) : (speed * distance) ) : (speed < 96.478163 ? (speed > -30.887784 ? dis
tance : greenLight (distance, speed) ) : greenLight (distance, speed) ) ) : (speed < 96.4781
63 ? (speed < -82.000183 ? greenLight (distance, speed) ) : (speed < distance ? (Light != GREEN
? (-30.887784 * redLight (distance, speed) ) : greenLight (distance, speed) ) : (speed < (Li
ght != GREEN ? sdiv (greenLight (distance, speed), redLight (distance, speed) ) : greenLig
ht (distance, speed) ) ? (redLight (distance, speed) < 41.581469 ? (greenLight (distance,
speed) < 41.581469 ? (Light != GREEN ? (Light != YELLOW ? redLight (distance, speed) : ( (Ligh
t != GREEN ? sdiv (greenLight (distance, speed), redLight (distance, speed) ) : greenLight
(distance, speed) ) - speed) ) : (speed * distance) ) : (speed < 96.478163 ? (speed > -30.88778
4 ? redLight (distance, speed) : greenLight (distance, speed) ) : greenLight (distance, sp
eed) ) ) : greenLight (distance, speed) ) : greenLight (distance, speed) ) ) : greenLight (d
istance, speed) ) ) : (speed < 96.478163 ? (speed < -82.000183 ? greenLight (distance, speed)
: (speed < distance ? (Light != GREEN ? (-30.887784 * redLight (distance, speed) ) : greenLi
ght (distance, speed) ) : (speed < (Light != GREEN ? sdiv (57.310708, redLight (distance, sp
eed) ) : greenLight (distance, speed) ) ? (redLight (distance, speed) < 41.581469 ? ( (Light
!= YELLOW ? redLight (distance, speed) : ( (Light != GREEN ? sdiv (greenLight (distance, spe
ed), redLight (distance, speed) ) : greenLight (distance, speed) ) - speed) ) < 41.581469 ? (
Light != GREEN ? (Light != YELLOW ? redLight (distance, speed) : ( (Light != GREEN ? sdiv (gree
nLight (distance, speed), redLight (distance, speed) ) : greenLight (distance, speed) ) -
speed) ) : (speed * distance) ) : (speed < 96.478163 ? (speed > -30.887784 ? redLight (distan
ce, speed) : greenLight (distance, speed) ) : greenLight (distance, speed) ) ) : greenLight (
distance, speed) ) : greenLight (distance, speed) ) ) : greenLight (distance, speed) ) ) : (
(25.717948 + (Light == YELLOW ? (Light != YELLOW ? (Light != YELLOW ? (Light == RED ? (Light == R
ED ? -29.233680 : 55.656605) : (-52.269051 + -91.589099) ) : redLight (distance, speed) ) : r
edLight (distance, speed) ) : (speed - greenLight (distance, speed) ) ) - greenLight (dist
ance, speed) ) ) ;

```

```

//***** Agent B (2.1456)
(speed > (distance < (speed > (Light == YELLOW ? greenLight (distance, speed) : 14.163640) ?
(Light == RED ? (distance < (speed > 29.978331 ? (Light != GREEN ? 36.149174 : speed) : 7.93176
0) ? 1.748710 : redLight (distance, speed) ) : (Light == RED ? (Light != GREEN ? 36.149174 : (Li
ght == RED ? greenLight (distance, speed) : (Light == RED ? speed : greenLight (distance, spe
ed) ) ) : (speed < 54.222235 ? (speed > (distance < (speed > 29.978331 ? (Light != GREEN ? 36.14
9174 : (Light == RED ? redLight (distance, speed) : (Light == RED ? greenLight (distance, spe
ed) : greenLight (distance, speed) ) ) : 7.931760) ? 1.748710 : redLight (distance, speed)
) ? (Light != GREEN ? (Light != YELLOW ? (speed < distance ? -4.806666 : redLight (distance, sp
eed) ) : (Light == RED ? greenLight (distance, speed) : (distance > distance ? speed : (speed <
distance ? 43.351542 : (speed < distance ? 4.049806 : redLight (distance, speed) ) ) ) : (Li
ght == RED ? greenLight (distance, speed) : (Light == RED ? speed : greenLight (distance, spe
ed) ) ) : 7.931760) : greenLight (distance, speed) ) ) : 7.931760) ? 1.748710 : redLight (di
stance, speed) ) ? (Light != GREEN ? (Light != YELLOW ? (speed < distance ? -4.806666 : redLigh
t (distance, speed) ) : (Light == RED ? greenLight (distance, speed) : (distance > distance ?
speed : (speed < distance ? 43.351542 : (speed < 54.222235 ? (speed > (distance < (speed > 29.9
78331 ? (Light != GREEN ? 36.149174 : (Light == RED ? greenLight (distance, speed) : (Light ==

```

```
RED?speed:greenLight (distance, speed) ) ) : 7.931760) ? 1.748710:redLight (distance, speed) ) ? (Light!=GREEN? (Light!=YELLOW? (speed<distance?-4.806666:redLight (distance, speed) ) : (Light==RED?greenLight (distance, speed) : (distance>distance?speed: (speed<distance?43.351542: (speed<greenLight (distance, speed) ? 4.049806:redLight (distance, speed) ) ) ) ) : (Light==RED?greenLight (distance, speed) : (Light==RED?speed:greenLight (distance, speed) ) ) : 7.931760):greenLight (distance, speed) ) ) ) : (Light==RED?greenLight (distance, speed) : (Light==RED?speed:greenLight (distance, speed) ) ) : 7.931760);
```

```
//***** Agent C (5.6140)  
(Light!=GREEN? ( (Light!=YELLOW? (Light!=GREEN?speed: (Light!=YELLOW?distance: (Light!=YELLOW? (Light==RED? (greenLight (distance, speed) ) / 23.734244) : redLight (distance, speed) ) : greenLight (distance, speed) ) ) : ( (speed/ (16.788232/ ((-14.169744+ (Light!=GREEN?28.708761:2.951140) ) / ((redLight (distance, speed) - (-30.124821) ) / redLight (distance, speed) ) * greenLight (distance, speed) ) ) ) - redLight (distance, speed) ) ) / (73.564867/redLight (distance, speed) ) ) : greenLight (distance, speed) ) ;
```

```
//***** Agent D (2.2379)  
(Light!=GREEN? (Light!=GREEN? ( (6.741538/ ((distance>speed? ( (Light!=GREEN? (Light!=GREEN?redLight (distance, speed) : redLight (distance, speed) ) : 14.322336) + (speed<3.738517? (speed/-78.850673) : ( (Light!=YELLOW? (redLight (distance, speed) < distance? (-20.273446* (-20.273446+redLight (distance, speed) ) ) : distance) : 6.741538) - redLight (distance, speed) ) ) : greenLight (distance, speed) ) < (Light==GREEN? (distance-greenLight (distance, speed) ) : redLight (distance, speed) ) ? ( (distance<(Light!=GREEN? (Light!=GREEN?speed:14.322336) : 14.322336) ? -52.812281: (Light==RED? (redLight (distance, speed) > speed? ( (Light!=GREEN? (-20.273446*14.322336) : 14.322336) + (speed<(-20.273446* (-20.273446+redLight (distance, speed) ) ) ? ( (-20.273446+redLight (distance, speed) ) - (-35.251930) ) / -78.850673) : ( (Light!=YELLOW? (speed<speed?speed:redLight (distance, speed) ) : 6.741538) - redLight (distance, speed) ) ) ) : greenLight (distance, speed) ) : 14.322336) ) / distance) : greenLight (distance, speed) ) * redLight (distance, speed) ) : 14.322336) : 14.322336);
```

```
//***** Agent E (6.0752)  
( ( (speed<44.248786?distance: (Light==RED? (distance<31.437116? (distance>speed? (21.610156>distance?22.928556:20.804468) : -84.453871) : speed) : -43.308817) ) > 11.551255? (distance>(distance>speed? (distance>14.139225? (distance>11.551255? (distance>(distance>speed?20.273445:-84.453871) ? (16.446424>-72.051149? (Light==YELLOW?24.051637:speed) : (Light!=GREEN?11.551255:16.446424) ) : (Light!=GREEN? (distance>(Light!=GREEN?14.139225:distance) ? (Light==GREEN?31.437116:11.551255) : (Light!=GREEN?-59.074679:16.446424) ) : 16.446424) ) : (Light!=GREEN?11.551255:16.446424) ) : (Light!=GREEN? (distance>(Light!=GREEN?14.139225:distance) ? (Light==GREEN? (Light!=RED? (Light!=RED?11.551255:11.551255) : 0.381481) : 11.551255) : (Light!=GREEN?-59.074679:16.446424) ) : 16.446424) ) : (Light!=GREEN?11.551255:16.446424) ) ? (distance>14.139225? (distance>11.551255? (Light!=RED? ( (speed<45.091097?distance:-82.146672) > 16.446424?12.112796: (Light!=GREEN?14.139225:16.367076) ) : 0.381481) : (Light!=GREEN?11.551255:16.446424) ) : (Light!=GREEN? (distance>(Light!=GREEN?14.139225:distance) ? (Light==GREEN? (Light!=RED? (Light!=RED?11.551255:11.551255) : 0.381481) : 11.551255) : (Light!=GREEN?-59.074679:16.446424) ) : 16.446424) ) : (Light!=GREEN?11.551255:16.446424) ) > -59.074679? (speed<44.248786?distance: (Light==RED? (distance<31.437116? (distance>speed? (21.610156>distance?22.928556:20.804468) : -84.453871) : speed) : -43.308817) ) > 11.551255? (distance>(distance>speed? (distance>14.139225? (distance>11.551255? (distance>(distance>speed?20.273445:-84.453871) ? (16.446424>-72.051149? (Light==YELLOW?24.0
```



```

51637:speed):(Light!=GREEN?11.551255:16.446424)):(Light!=GREEN?(distance>(Light!=GREEN?14.139225:distance)?(Light==GREEN?31.437116:11.551255):(Light!=GREEN?-59.074679:16.446424)):16.446424)):(Light!=GREEN?11.551255:16.446424)):(Light!=GREEN?(distance>(Light!=GREEN?14.139225:distance)?(Light==GREEN?(Light!=RED?(Light!=RED?11.551255:11.551255):0.381481):11.551255):(Light!=GREEN?-59.074679:16.446424)):16.446424)):(Light!=GREEN?11.551255:16.446424))?(distance>14.139225?(distance>11.551255?(Light!=RED?(speed<45.091097?distance:-82.146672)>16.446424?12.112796:(Light!=GREEN?14.139225:16.367076)):0.381481):(Light!=GREEN?11.551255:16.446424)):(Light!=GREEN?(distance>(Light!=GREEN?14.139225:distance)?(Light==GREEN?(Light!=RED?(Light!=RED?11.551255:11.551255):0.381481):11.551255):(Light!=GREEN?-59.074679:16.446424)):16.446424)):(Light!=GREEN?11.551255:16.446424)):(Light!=GREEN?11.551255:16.446424)):-84.453871);

```

Traffic Light Driving, Sentinel Rules

```

//***** Agent A (2.3244)

```

```

(lightPresent?(distance>52.342295?(!intersection?(lightPresent?0:(intersection?(distance>92.556535?(intersection?(speed>76.165044?1:1):(distance<50.828577?1:(!lightPresent?(speed>49.082919?(!intersection?1:1):0):(distance>12.195196?0:(!lightPresent?(speed<99.957274?1:(!lightPresent?(speed<18.628498?0:0):1):1))))):0):0):(distance>89.193396?0:(!lightPresent?(distance>20.261238?1:(intersection?(distance>78.893399?0:0):(speed<34.116642?(!lightPresent?(speed>1.928770?(!intersection?(lightPresent?(!intersection?(lightPresent?1:1):(speed>29.386273?1:1)):(speed<83.108005?(!lightPresent?0:0):0):(lightPresent?(speed<20.438246?0:1):(intersection?0:0)):0):0):0)):(distance>63.087863?0:(lightPresent?0:(speed>31.556139?(lightPresent?(speed<66.103091?1:(intersection?(distance<84.969634?(!intersection?(lightPresent?0:0):(distance>12.833033?0:1):(lightPresent?(distance<89.617603?0:0):(distance>79.726554?0:1)):(distance<47.410504?1:1)):(!intersection?(lightPresent?(lightPresent?1:(lightPresent?(speed<11.294900?1:1):(intersection?0:1)):(!intersection?(speed<38.923307?1:0):(intersection?(!lightPresent?0:1):(intersection?0:1))))):1):(!intersection?0:(intersection?0:(!lightPresent?(distance<64.592425?(lightPresent?(lightPresent?(!intersection?0:0):(intersection?(speed>17.899106?0:0):(speed<30.237739?0:1)):1):(lightPresent?0:(distance<21.897030?(lightPresent?(intersection?(lightPresent?1:(distance<59.251686?(!intersection?0:0):1):(lightPresent?(distance>44.499039?0:(!intersection?0:0):1):(speed<2.450636?(intersection?1:(distance>81.658376?0:(lightPresent?0:1)):0):(intersection?(lightPresent?1:(distance>51.997436?(intersection?1:(distance<56.056398?1:0):0):0)))):(distance<2.188177?0:1)))));

```

```

//***** Agent B (3.0400)

```

```

(speed>19.278542?(!intersection?(distance<70.607623?(lightPresent?(distance>17.438276?(intersection?25.034333:(distance<speed?(distance>39.167455?1:(intersection?1:(lightPresent?(distance>17.438276?0:(lightPresent?1:1)):(distance>49.220252?1:(!intersection?1:1))))):1):(lightPresent?1:1):(distance>49.220252?1:(!intersection?0:0)):0):(17.438276>distance?(!intersection?1:1):0):(lightPresent?1:0));

```

```

//***** Agent C (3.3433)

```

```

(lightPresent?(distance>42.072817?(distance<38.898892?1:(lightPresent?(speed>35.117649?(distance<63.811151?1:(speed<speed?0:(lightPresent?(lightPresent?(speed>38.898892?(distance<distance?1:(distance<90.594195?1:0)):1):0):0)):0):0):1):0);

```

```
//***** Agent D (2.9296)
(distance<76.485488?(!lightPresent?0:(distance>24.814600?(lightPresent?(distance>27.884762?(lightPresent?(intersection?0:(speed<24.250008?(distance>37.684255?(intersection?1:0):1):1)): (distance>39.240699?(lightPresent?1:1):(!intersection?1:1))):(!lightPresent?0:(!lightPresent?(lightPresent?1:0):0))): (distance>39.240699?(lightPresent?1:1):(!intersection?1:1)):1): (intersection?(!intersection?1:(distance<76.485488?(!lightPresent?0:1):(intersection?(!intersection?1:(!lightPresent?0:1)):(intersection?(distance<55.903805?0:(!lightPresent?0:0)):(!intersection?0:0))))): (intersection?(distance<55.903805?0:(!lightPresent?0:1)):(!intersection?0:0))));
```

```
//***** Agent E (4.3173)
(distance<9.518723?(distance<3.320414?(!lightPresent?0:0):(lightPresent?0:(distance<65.376751?(!intersection?(speed>17.322306?(intersection?95.974608:1):(lightPresent?(distance<distance?0:(!intersection?1:1)):(speed<64.369640?1:1))): (distance>56.953642?(!lightPresent?1:1):(intersection?1:(!lightPresent?(distance>78.405102?1:1):(lightPresent?1:0))))):1)): (intersection?(!lightPresent?(23.624378<speed?(distance>25.260170?(!intersection?(!intersection?0:1):(speed>46.659749?(intersection?(!intersection?(23.624378<23.624378?1:0):(!intersection?(speed<distance?1:1):(distance<72.026734?1:0))): (distance<5.633717?(!intersection?(!lightPresent?1:(intersection?1:0)):1):(!intersection?(intersection?(!intersection?1:1):0):(distance<23.624378?(distance<23.624378?1:0):1))))):0):(!intersection?(intersection?1:0):1):(!intersection?0:0)): (27.137059<96.331675?(distance>27.137059?(27.137059<distance?(distance>27.137059?(!lightPresent?0:(distance<87.636952?(distance>34.464553?(speed<distance?1:(!intersection?1:0)):1):(!intersection?0:0))):(!intersection?(intersection?1:0):1):(!intersection?0:0)):1):(!intersection?0:0)): (speed<96.331675?(distance>27.137059?(27.137059<distance?(distance>27.842036?(!lightPresent?0:(distance<96.331675?(distance>27.137059?(speed<distance?(distance>27.842036?1:(!intersection?0:1)):(!intersection?0:0)):1):(!intersection?0:0))):(!intersection?(intersection?1:0):1):(!intersection?0:0)):1):(!intersection?0:0))));
```

Urban Driving, Action Rules

```
//***** Agent A (2.0333)
(67.711416<(25.901059-((67.711416<(sin(pow(speed,2))?(sin(-55.162206)):(67.711416-(65.678884<(pow((slog((speed<-6.888028?(sin(speed)):pow(speed,5))))),3)-pow((slog((sin(pow(speed,5))))),3)?pow((slog((pow((slog(speed)),3)-(speed>67.711416?(sin((slog((65.678884-speed))))):speed))))),3):speed)))-speed)?(sin(-55.162206)):(67.711416-(speed<37.076937?pow((slog((65.678884-(speed<(65.678884-(65.678884-speed))?pow((slog((pow((slog((65.678884-speed))))),3)-(speed>65.678884?92.907498:speed))))),3):speed))))),3):speed))));
```

```
//***** Agent B (3.1051)
(speed<47.831659?29.789117:(speed>56.498916?10.977508:20.609149));
```

```
//***** Agent C (0.8235)
(((speed<(speed<84.337901?(21.091342<95.056001?speed:(22.504349*22.760704)):(22.504349*speed)?sdiv(speed,speed):(67.210913-speed))+sdiv(80.672628,(speed<65.279092?(speed<49.027985?sdiv(((speed>47.047334?21.091342:(speed+sdiv(speed,69.698172))<speed?(speed-(((speed<speed?sdiv((speed>44.212165?(speed>speed?11.078219:11.774041):speed),(speed<57.216101?90.466017:(speed>43.775750?sp
```

```
eed:68.105106))):((speed>speed?speed:(speed>73.790704?28.525651:speed))- (spee
d>speed?(73.772393*28.186895):speed))- (89.690847>50.846888?99.679555:21.0913
42))- (speed>19.464705?77.388836:speed))* (speed>19.464705?77.388836:speed))- (s
peed>19.464705?77.388836:speed)): (22.504349*speed)), ((speed<95.056001?(speed
-(speed>19.464705?77.388836:speed)): (22.504349*speed))+speed)):58.613849): (sp
eed>speed?(speed>speed?speed:(50.239570*speed)):35.871456))-94.946135)))+sdiv
(80.672628, ((speed<65.279092?(speed<49.027985?sdiv(speed, (22.190008*speed)):5
8.613849): (speed>speed?(speed>speed?speed:speed):35.871456))-94.946135))));
```

```
//***** Agent D (1.1777)
```

```
(speed<61.079745?(speed<(18.079165<61.079745?48.133793:sdiv(6.781213,37.53776
7)))?((34.272286-61.079745)+43.266091)+(60.765404-(speed<61.079745?( (speed>sp
eed?37.537767:99.310281)-52.400281):sdiv(6.781213,37.537767))))): (speed<61.079
745?( (96.752830<61.079745?( (18.079165<(18.079165-( (speed>speed?96.752830:99.3
10281)-52.400281))?( (16.306040-(speed>speed?96.752830:61.079745))+43.266091)
+(60.765404-speed)):sdiv(6.781213,37.537767))+ (60.765404-speed)):sdiv(6.78121
3,37.537767))<61.079745?( ( (speed<61.079745?( (speed<(96.752830+(60.765404-spee
d))?(speed<61.079745?( ( (18.079165-( (speed>speed?96.752830:99.310281)-52.40028
1))+43.266091)+(60.765404-speed)):sdiv(6.781213,37.537767)):sdiv(6.781213,37.
537767))+ (60.765404-speed)):sdiv(6.781213,37.537767))<61.079745?( ( (16.306040-
61.079745)+43.266091)+(60.765404-speed)):sdiv(6.781213,37.537767))+ (60.765404
-speed)):sdiv(6.781213,37.537767)):sdiv(6.781213,37.537767)):sdiv(6.781213,3
7.537767));
```

```
//***** Agent E (1.7617)
```

```
((33.005768>speed?speed:((98.425245+speed)*(speed>speed?speed:22.299875))- (s
peed*sdiv((speed-18.634602), (57.719657-( (speed<speed?(speed<speed?84.246345:s
peed): (sdiv(speed,45.713675)-(speed>speed?53.260903:sdiv(8.981597,60.509049)
))+speed))>58.482620?90.450758:(speed>speed?speed:speed))))))>27.640614?( (sd
iv((speed<59.453108?speed:(speed-97.427289)),31.305887)+(speed>speed?speed:(s
peed>speed?speed:( (92.434461-(sdiv(( (speed>53.019806?(speed>(speed<58.561968?
38.392285:speed)?(83.919797-speed):speed):speed)<(speed<(92.434461-speed)?97.
332682:speed)?97.024445:(speed>speed?(speed>(speed<59.453108?speed:(speed-97.
427289))?( (speed>52.235481?speed:85.634937)*speed):speed):speed)),speed)+(spe
ed>speed?8.536027:(speed>speed?speed:sdiv(31.073946,sdiv(speed,31.305887))))))
)-(speed>speed?19.876705:(speed>speed?speed:(speed+(speed>speed?8.536027:(spe
ed>speed?speed:sdiv(31.073946,sdiv(speed,31.305887)))))))+ (speed>speed?8.
536027:(speed>speed?speed:( (92.434461-speed)-(speed>speed?19.876705:(speed>sp
eed?speed:(sdiv(( (speed>33.005768?(speed>55.595569?(83.919797-speed):speed):s
peed)<(speed<(92.434461-speed)?97.332682:speed)?97.024445:(speed>speed?(speed
>(speed<59.453108?speed:(speed-97.427289))?( (speed>52.235481?speed:sdiv(4.516
739,speed))*speed):speed):speed)),speed)+(speed>speed?8.536027:(speed>speed?s
peed:sdiv(31.073946,sdiv(speed,31.305887))))))): (speed>speed?(speed>29.32
5236?sdiv((10.602130+( (speed<speed?(55.702383+speed):(74.349193-95.742668))*s
peed)),50.907926):9.161657):speed));
```

LIST OF REFERENCES

- [1] Abravanel, E., Ferguson, S. (1998) "Observational learning and the use of retrieval information during the second and third years" *Journal of Genetic Psychology*; Dec 1998, v.159, 4, 455(2)
- [2] Anderson, D., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D. (2002) "SETI@home: An Experiment in Public-Resource Computing", *Communications of the ACM*, Vol. 45 No. 11, November 2002, pp56-61
- [3] Anderson, J.R. (1993) "Rules of the Mind". Hillsdale, NJ: Lawrence Erlbaum.
- [4] Atkeson CG, Hale J, Pollick F, Riley M, Kotosaka S, Schaal S, Shibata T, Tevatia G, Vijayakumar S, Ude A, Kawato M (2000) "Using humanoid robots to study human behavior." *IEEE Intelligent Systems: Special Issue on Humanoid Robotics*,(2000).
- [5] Banks, S., Stytz M. (1999) "Considerations for the Next Generation of Air Force Computer Generated Actors ", *Proceedings from The Simulation Technology and Training (SimTecT99) Conference 29th March to 1st April 1999 Carlton Crest Hotel, Melbourne, Australia*
- [6] Banzhaf W., Nordin P., Keller R. E., Francone F. D. (1998) "Genetic Programming – An Introduction", pp239-276, 1998, Morgan Kaufmann Publishers, Inc., ISBN 1-55860-510-X
- [7] Bennett F. H., Koza J. R., Keane M. A., and Andre D. (1999) "Genetic programming: Biologically inspired computation that exhibits creativity in solving non-trivial problems." In *AISB'99 Symposium on AI and Scientific Creativity*, pp29-38, Edinburgh, Scotland, April 1999.
- [8] Bentivegna D., and Atkeson, C. (2001) "Learning From Observation Using Primitives" *Proceedings of the 2001 IEEE International Conference on Robotics & Automation*, Seoul, Korea, May 21-26 2001.
- [9] Bergström J., (2003) "Context-Based Reasoning: knowledge engineer vs. machine learning in the knowledge elicitation phase", Degree Project, Dalarna University, Sweden.

- [10] Biles J.A. (1994) "GenJam: A Genetic Algorithm for Generating Jazz Solos." In Proceedings of the 1994 International Computer Music Conference, ICMA, San Francisco.
- [11] Bojarczuk, C.C., Lopes, H.S., Freitas, A.A. (2000) "Genetic programming for knowledge discovery in chest-pain diagnosis", IEEE Engineering in Medicine and Biology Magazine, Volume: 19 Issue: 4 , July-Aug. 2000, pp38–44
- [12] Bonarini, A. (2001) "Evolutionary learning, reinforcement learning, and fuzzy rules for knowledge acquisition in agent-based systems". Proceedings of the IEEE , Volume: 89 Issue: 9 , Sept. 2001, pp1334 -1346
- [13] Brieman, L., Friedman J., Stone C.J., and Olshen R.A. (1984) "Classification and regression trees". Chapman & Hall, CRC Press, Boca Raton, FL.
- [14] Dahal, K.P., Burt, G.M., McDonald, J.R., Moyes, A., (2001) "A case study of scheduling storage tanks using a hybrid genetic algorithm", IEEE Transactions on Evolutionary Computation, Volume: 5 Issue: 3, June 2001, pp283 –294
- [15] Dasarathy, Belur V (1991) "Nearest Neighbor Norms: NN Pattern Classification Techniques". Los Alamitos: IEEE Computer Society Press, 1991.
- [16] Deutsch, S. (1993) "Notes Taken on the Quest for Modeling Skilled Human Behavior" Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation, Orlando, FL, March 17-19, pp.359-365
- [17] DoD 5000.59-P (1995) "DoD Modeling and Simulation (M&S) Master Plan" (online), Available: <http://www.dmsomil/briefs/msdocs/policy/msmp.pdf> (2001, October 22)
- [18] Eggermont J., van Hemert J. (2001) "Adaptive Genetic Programming Applied to New and Existing Simple Regression Problems", Proceedings from 4th European Conference, EuroGP 2001, Lake Como, Italy, April 2001.
- [19] Endsley, M. (1988) "Design and evaluation of situation awareness enhancement." Pp. 97-101 in Proceedings of the Human Factors Society 32nd Annual Meeting. Santa Monica, CA: Human Factors and Ergonomics Society.
- [20] Eklund, S. (2002) "A Massively Parallel GP Engine in VLSI", presented at the Congress on Evolutionary Computing, CEC-2002, Honolulu, 2002-05-17. Published in the proceedings "The 2002 IEEE World Congress on Computational Intelligence", pp 629-633.
- [21] EvoWeb (1999), "Evol-artists - a new breed entirely", EvoNews 11, Posted:29 Oct 99, Available 3rd of December, 2001at:

http://evonet.dcs.napier.ac.uk/evoweb/news_events/news_features/nf_article40.html

- [22] Feldt R., Nordin P. (2000) "Using Factorial Experiments to Evaluate the Effect of Genetic Programming Parameters", Genetic Programming, Proceedings of EuroGP'2000, volume 1802 of LNCS, pages 271-282, Edinburgh, 15-16 April 2000. Springer-Verlag.
- [23] Fernlund, H., Eklund, S., Gonzalez, A. J. (2004) "The CxBR Diffusion Engine – A Tool for Modeling Human Behavior on the Battle Field", Proceedings of the 2nd Swedish-American Workshop on Modeling and Simulation, Orlando, FL, February 2004.
- [24] Garcia, S., Levine, J. and Gonzalez, F. (2003). Multi Niche Parallel GP with a Junk-code Migration Model. Proceedings of the 6th European Conference on Genetic Programming (EuroGP 2003), Springer.
- [25] Goldberg, D.E. (1989) "Genetic Algorithms in Search, Optimization, and Machine Learning", Addison-Wesley, Boston MA, 1989
- [26] Gonzalez A. J. and Ahlers, R. H. (1994) "A Novel Paradigm for Representing Tactical Knowledge in Intelligent Simulated Opponents" Proceeding of the IAE/AIE Conference, May 31-June 3, 1994, Austin, Texas.
- [27] Gonzalez A. J. and Ahlers, R. H. (1998) "Context-Based Representation of Intelligent Behavior in Training Simulations" Transactions of the Society for Computer Simulation International, volume 15, no 4, December 1998
- [28] Gonzalez A. J. Georgiopoulos M. DeMara R. F. Henninger A. Gerber W. (1998) "Automating the CGF Model Development and Refinement Process by Observing Expert Behavior in a Simulation" Proceedings of the Computer Generated Forces Conference, Orlando, 1998
- [29] Gonzalez A. J. DeMara R. F. Georgiopoulos M. (1998) "Vehicle Model Generation and Optimization for Embedded Simulation" Proceedings of the Spring 1998 Simulation Interoperability Workshop, Orlando, FL March 1998.
- [30] Grefenstette J. J. (1986) "Optimization of control parameters for genetic algorithms", IEEE Transactions on Systems, Man, & Cybernetics 16(1) (1986) 122--128.
- [31] Gruau, F., Whitley, D. and Pyeatt, L. (1996). "A comparison between cellular encoding and direct encoding for genetic neural networks", Proceedings of the First Genetic Programming Conference, pp. 81-89.
- [32] Henninger A., Gonzalez A., Gerber W. Georgiopoulos M., DeMara R. (2000) "On the Fidelity of SAFs: Can Performance Data Help?" Proceedings of the Inter-service/Industry Simulation and Education Conference, Orlando, FL, 2000.

- [33] Henninger, A. (2001). "The Use Of Neural Network Based Movement Models To Improve The Predictive Utility Of Entity State Synchronization Methods For Distributed Simulations", Doctoral Dissertation, University of Central Florida, Orlando, 2001.
- [34] Holland, J.H. (1975) "Adaptation in Natural and Artificial Systems", University of Michigan Press, 1975. Reprinted by MIT Press, Cambridge MA, 1992
- [35] Hoi-Kau Yuen, Richards, T.J. (1993) "GTKAT: a grounded theory based knowledge acquisition tool for expert systems", Artificial Neural Networks and Expert Systems, 1993. Proceedings., First New Zealand International Two-Stream Conference on , 24-26 Nov. 1993, pp 152 - 155
- [36] Hsu, W.H., Gustafson, S.M. (2001) "Genetic Programming for Layered Learning of Multi-agent Taks", Proceedings of the Genetic Evolutionary Computation Conference, GECCO 2001, San Francisco, CA, July 9-11, 2001
- [37] Janeczko, C.; Lopes, H.S (2000) "A genetic approach to ARMA filter synthesis for EEG signal simulation", 2000. Proceedings of the 2000 Congress on Evolutionary Computation, Volume: 1, 2000, pp373 -378
- [38] Juidette, H.; Youlal, H. (2000) "Fuzzy dynamic path planning using genetic algorithms", Electronics Letters, Volume: 36 Issue: 4, 17 Feb. 2000, pp374 –376
- [39] Karr, C., Hollis, P., Trendennick, W., Kollenberg, J. (2000) "Fuzzy Modeling of Synergistic and Implicit Battlefield Effects in WARSIM" Proceedings of the 9th Conference on Computer Generated Forces and Behavioral Representation, May 16-18, 2000
- [40] Koza, J. R. (1992) "Genetic Programming", MIT Press, Cambridge MA, 1992, ISBN 0-262-11170-5
- [41] Koza J.R., Bennett III, F. H., Bennett F. H., Andre D., Keane M .A., (1999) "Genetic Programming III: Automatic Programming and Automatic Circuit Synthesis", Morgan Kaufmann Publishers, Los Altos CA, ISBN: 1558605436
- [42] Koza J. R. Keane M. A. Yu J. and Mydlowec W. (2000) "Automatic Synthesis of Electrical Circuits Containing a Free Variable Using Genetic Programming" Proceedings of the Genetic and Evolutionary Computation Conference, Las Vegas, Nevada, 2000, pp477-484.
- [43] Lee, S., Chen, J. (1994) "Skill learning from observations" Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on , 1994, pp3245 -3250 vol.4
- [44] van Lent, M., Laird, J. (1998) "Learning by Observation in a Complex Domain", Proceedings of the Eleventh Workshop on Knowledge Acquisition, Modeling and

Management, Alberta, Canada, April, 1998

- [45] Levenick, J. R. (1991) "Inserting introns improves genetic algorithm success rate: taking a cue from biology", Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991) ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 123-27
- [46] Luke, S. (1998) "Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup-97", Proceedings of the Third Annual Genetic Programming Conference, p.204-222, Morgan Kaufmann, Los Altos, CA, 1998
- [47] Luke, S. (2000) "Code Growth Is Not Caused By Introns". Late Breaking Papers of the Genetic and Evolutionary Computation Conference (GECCO 2000), Las Vegas NV, July. pp228-235.
- [48] McCulloch W.S. and Pitts, W., A logical calculus of the idea immanent in neural nets. Bulletin of Mathematical Biophysics, pp115-137, 1943
- [49] Mendes, R.F. Voznika, F.B. Nievola, J.C. Freitas, A.A. (1998) "Discovering Fuzzy Classification Rules with Genetic Programming and Co-Evolution", Proceedings of the Third Annual Genetic Programming Conference, pp 204-222, Morgan Kaufmann, Los Altos, CA, 1998
- [50] Miller, B.L. and Shaw, M.J. (1996). "Genetic Algorithms with dynamic niche sharing for multimodal function optimization", IEEE International Conference on Evolutionary Computation, pp 786-791. Piscataway, NJ: IEEE Press
- [51] Milzner, K., Leifhelm, B. (1992) "Learning by Observation and Active Experimentation in a Knowledge Based CAD-Environment" Computer Systems and Software Engineering Proceedings, 1992, pp 424 - 429
- [52] Moukas, A., Hayes, G., (1996). "Synthetic Robotic Language Acquisition by Observation", Proc. 4th Int. Conf. on Simulation of Adaptive Behavior, 568--579, MIT Press, 1996.
- [53] Newell, A. (1990) "Unified Theories of Cognition", Cambridge, MA: Harvard University Press.
- [54] Nordin, P., Banzhaf, W., and Francone, F. D. (1997). Introns in nature and in simulated structure evolution. In Proceedings Bio-Computing and Emergent Computation. Springer.
- [55] Norlander, L. (1998) "A Framework for Efficient Implementation of Context-Based Reasoning in Intelligent Simulation", Master Thesis, Electrical and Computer Engineering, University of Central Florida, Orlando, FL, 1998.
- [56] Pew, R. W.; Mavor, A. S., Editors (1997) "Panel on Modeling Human Behavior and

Command Decision Making”: Representations for Military Simulations, National Research Council, ISBN 0-309-08662-0

- [57] Pew, R. W.; Mavor, A. S. (1998) “Modeling Human and Organizational Behavior : Application to Military Simulations”, Washington, D.C. National Academies Press, 1998. ISBN 0309060966
- [58] Podgorelec, V.; Kokol, P.; Završnik, J. (1999) “Medical diagnosis prediction using genetic programming”, Proceedings. 12th IEEE Symposium on Computer-Based Medical Systems, 1999, pp202 –207
- [59] Pomerleau D. and Jochem T. (1996) “Rapidly Adapting Machine Vision for Automated Vehicle Steering” IEEE Expert: Special Issue on Intelligent System and their Applications, Vol. 11, No. 2, April, 1996, pp. 19-27.
- [60] Potter M. A. and De Jong K. A. (1994), “A Cooperative Coevolutionary Approach to Function Optimization”, The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature, pp.249-257, October 09-14, 1994
- [61] Potter M. A. and De Jong K. A. (1995) “Evolving Neural Networks with Collaborative Species.” In Proceedings of the 1995 Summer Computer Simulation Conference, July 24-26, Ottawa, Ontario, Canada, pages 340-345. The Society for Computer Simulation.
- [62] Quinlan, J.R., (1986) "Induction of decision trees", Machine Learning, Morgan Kaufmann, 1986, pp.81-106
- [63] Quinlan, J. R. (1987) “Generating Production Rules from Decision Trees”, Proc. Int'l Joint Conf. Artificial Intelligence (IJCAI-87), Morgan Kaufmann, San Mateo, Calif., 1987, pp304-307.
- [64] Rosenblatt, F. (1958), "The perceptron: a probabilistic model for information storage and organization in the brain", Psychological Review, 65:386-408
- [65] Saeki S., and Gonzalez A.J. (2000) "Soft-coding the Transitions between Contexts in CGF's: The Competing Context Approach," Proceedings of the Computer Generate Forces and Behavior Representation Conference, Orlando, FL, May 17, 2000.
- [66] Sammut, C., Hurst, S., Kedzier, D., Michie, D. (1992). “Learning to Fly”, Proceedings of the ninth International Conference on Machine Learning, Aberdeen, pp335-339.
- [67] Schaal, S. (1997). "Learning from demonstration." In: M.C. Mozer, M. Jordan, and T. Petsche (eds.), Advances in Neural Information Processing Systems 9, pp1040-1046, Cambridge, MA: MIT Press.
- [68] Schaal, S., (1999) “Is imitation learning the route to humanoid robots?” Trends in

Cognitive Sciences, 3(6), pp 233-242.

- [69] Schultz A.C. and Grefenstette J.J. and Adams W. (1996) "Robo-Shepherd: Learning Complex Robotic Behaviors," Proc. of the International Symposium on Robotics and Automation (ISRAM '96), ASME Press: New York, pp763-768, May, 1996.
- [70] Sidani, T., Gonzalez, A. (2000). "A Framework for Learning Expert Knowledge through Observation", Transactions of the Society for Computer Simulation International, vol 17, number 2, pp54-72.
- [71] Stuart R. and Norvig P. (1995), "Artificial Intelligence – A Modern Approach", Prentice Hall, 1995, ISBN 0-13-103805-2
- [72] Sutton, R., Barto, A. (1998). "Reinforcement Learning", MIT Press, Cambridge, MA, 1998, ISBN 0-262-19398-1
- [73] Takamatsu, J., Tominaga, H., Ogawara K., Kimura, H., Ikeuchi, K. (2000) "Extracting Manipulation Skills from Observation" International Conference on Intelligent Robot and Systems (IROS)'00 Vol.1 pp.584 - 589 2000 11 in Kagawa, Japan
- [74] Tambe, M., Johnson, W.L., Jones, R., Koss, F., Laird, J.E., Rosenbloom, P.S., and Schwamb, K. (1995). "Intelligent Agents for Interactive Simulation Environments", AI Magazine, 1995. 16(1).
- [75] Teredesai A., Park J., Govindaraju V. (2001) "Active Handwritten Character Recognition Using Genetic Programming", Proceedings from 4th European Conference, EuroGP 2001, Lake Como, Italy, April 2001.
- [76] Turner R. M., (1995) "Context-Sensitive, Adaptive Reasoning for Intelligent AUV Control: Orca Project Update", Proceedings of the 9th International Symposium on Unmanned Untethered Submersible Technology (AUV'95), Durham, New Hampshire.
- [77] Tzung-Pei Hong; Ke-Yuan Huang; Wen-Yang Lin, (2000) "A genetic search method for multi-player game playing", IEEE International Conference on Systems, Man, and Cybernetics, 2000, Volume: 5, 2000, pp3858 -3861 vol.5
- [78] Ventrella J. (1996) "Sexual Swimmers (Emergent Morphology and Locomotion without a Fitness Function)". From Animals to Animats. MIT Press 1996 pp484-493
- [79] Wang, J.W., Chen, C.H., Pan, J.S. (1998) "Genetic Feature-Selection for Texture Classification Using 2-D Nonseparable Wavelet Bases", IEICE(E81A), No. 8, August 1998, pp1635-1644.
- [80] Whitmore, J., Eddy, D., Schiflett, S., Tessier, P., and Dalrymple, M. (2000) "Adding Humannes to a Command and Control Agent" Proceedings of the 9th Conference on

Computer Generated Forces and Behavioral Representation, May 16-18, 2000

- [81] Wickens, C.D. (1992) "Engineering Psychology and Human Performance, second edition". New York, NY: HarperCollins
- [82] Wineberg, M., Oppacher, F. (1996) "The benefits of computing with introns", In Koza, J.R., Goldberg, D.E., Fogel, D.B., Riolo, R.L., eds.: Genetic Programming 1996: Proceedings of the First Annual Conference, Stanford University, CA, USA, MIT Press (1996) 410-415
- [83] Wolpert D.H. and Macready, W.G. (1995) "No free lunch theorems for search", Technical Report SFI-TR-95-02-010, 1995.
- [84] Wolpert D.H. and Macready, W.G. (1997) "No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1, pp. 67-82, 1997
- [85] Zadeh, L.A.(1988) "Fuzzy logic" IEEE Computer, Volume: 21 Issue: 4 , April 1988 pp83-93