Falconet: Force-feedback Approach For Learning From Coaching And Observation Using Natural And Experiential Training

Gary Stein
University of Central Florida

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
FALCONET: 
FORCE-FEEDBACK APPROACH FOR LEARNING FROM COACHING AND OBSERVATION USING NATURAL AND EXPERIENTIAL TRAINING

by

GARY STEIN
B.S. C.S. University of Central Florida, 2004
B.S. E.E. University of Central Florida, 2004
M.S. Cp.E. University of Central Florida, 2006

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

Summer Term
2009

Major Professor: Avelino Gonzalez
Building an intelligent agent model from scratch is a difficult task. Thus, it would be preferable to have an automated process perform this task. There have been many manual and automatic techniques, however, each of these has various issues with obtaining, organizing, or making use of the data. Additionally, it can be difficult to get perfect data or, once the data is obtained, impractical to get a human subject to explain why some action was performed. Because of these problems, machine learning from observation emerged to produce agent models based on observational data. Learning from observation uses unobtrusive and purely observable information to construct an agent that behaves similarly to the observed human. Typically, an observational system builds an agent only based on prerecorded observations. This type of system works well with respect to agent creation, but lacks the ability to be trained and updated on-line. To overcome these deficiencies, the proposed system works by adding an augmented force-feedback system of training that senses the agents intentions haptically. Furthermore, because not all possible situations can be observed or directly trained, a third stage of learning from practice is added for the agent to gain additional knowledge for a particular mission. These stages of learning mimic the natural way a human might learn a task by first watching the task being performed, then being coached to improve, and finally practicing to self improve. The hypothesis is that a system that is
initially trained using human recorded data (Observational), then tuned and adjusted using force-feedback (Instructional), and then allowed to perform the task in different situations (Experiential) will be better than any individual step or combination of steps.
To my parents, I won’t be a professional student forever.
ACKNOWLEDGMENTS

Computers aren’t smart, they are more of high-speed idiots, programmed by low-speed idiots.

–Unknown

Good judgment comes from experience, and experience comes from bad judgment.

–Frederick P. Brooks

A clever person solves a problem. A wise person avoids it.

–Albert Einstein

Any sufficiently advanced technology is indistinguishable from magic.

–Arthur C. Clarke

Man is the lowest-cost, 150-pound, nonlinear, all-purpose computer system which can be mass-produced by unskilled labor.

–NASA Spokesman

It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is the most adaptable to change.

–Charles Darwin

I bet the human brain is a kluge.

–Marvin Minsky
# TABLE OF CONTENTS

LIST OF FIGURES .................................................. xiii

LIST OF TABLES ................................................... xviii

LIST OF ACRONYMS/ABBREVIATIONS ............................... xxi

CHAPTER 1 MULTIMODAL LEARNING ................................. 1

1.1 Introduction ................................................. 1

1.2 Human Learning ............................................. 3

   1.2.1 Baseball Allegory ..................................... 3

   1.2.2 Computer Comparison ................................. 5

1.3 Learning Approaches ......................................... 8

   1.3.1 Observational Learning ............................... 9

   1.3.2 Instructional Learning ............................... 16

   1.3.3 Experiential Learning ............................... 19

   1.3.4 Learning Summary .................................... 21
# CHAPTER 2 HAPTICS

## 2.1 Haptics and Telepresence

- **2.1.1 Haptic Interaction**
- **2.1.2 Virtual Reality and Teleoperation**

## 2.2 Extrasensory Input

- **2.2.1 Virtual Forces**
- **2.2.2 Conceptual Understanding**

## 2.3 Training Mechanism

- **2.3.1 Human to Human Training**
- **2.3.2 Machine to Human Training**
- **2.3.3 Human to Machine Training**

# CHAPTER 3 PROBLEM DEFINITION

## 3.1 General Problem

## 3.2 Specific Problem

## 3.3 Hypothesis

## 3.4 Contributions

# CHAPTER 4 ALGORITHM ANALYSIS

## 4.1 Manual and Semi-Automatic Methods
4.1.1 Classical Methods ............................................. 42
4.1.2 Expert and Knowledge-Based Systems ...................... 48
4.2 Automatic Methods .............................................. 52
  4.2.1 Data-Based Algorithmic Approaches ......................... 53
  4.2.2 Artificial Neural Networks (ANN) .......................... 58
  4.2.3 Genetic Algorithms and Genetic Programming .............. 67
  4.2.4 Reinforcement Learning/ACO/PSO ......................... 76
  4.2.5 Hybrid Algorithms .......................................... 82

CHAPTER 5 SOLUTION IMPLEMENTATION ........................... 88

  5.1 Algorithmic Analysis Summary ................................ 88
  5.2 Algorithmic Implementation ................................... 92
    5.2.1 NEAT Implementation ................................... 93
    5.2.2 PSO Implementation .................................... 103
    5.2.3 PIGEON Implementation .................................. 106
    5.2.4 Code Implementation .................................... 110
  5.3 Learning Approaches Implemented ............................. 112
    5.3.1 Observational Learning Stage ............................ 112
    5.3.2 Instructional Learning Stage ............................. 116
5.3.3 Experiential Learning Stage .............................................. 118
5.3.4 Combination of Stages ..................................................... 120
5.4 Implementation Summary .................................................. 122

CHAPTER 6 ALGORITHM SELECTION ....................................... 123

6.1 Experimental Simulations ................................................... 123
  6.1.1 The Chaser Simulation ............................................... 124
  6.1.2 Sheep Simulation ....................................................... 126
  6.1.3 Crane Simulation ....................................................... 129
  6.1.4 Car Simulation ......................................................... 132
6.2 Comparison of Algorithmic Methods ..................................... 136
  6.2.1 Method of Comparison ............................................... 138
  6.2.2 Algorithm-Specific Parameters .................................... 140
6.3 Algorithm Selection per Stage ........................................... 155
  6.3.1 Observational Comparison ........................................... 156
  6.3.2 Instructional Comparison ............................................ 165
  6.3.3 Experiential Comparison ............................................. 173
  6.3.4 Additional Experimental Testing ................................... 179
6.4 Summary of Method Experiments ........................................ 188
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1 Comparison and Combination of Learning Approaches</td>
<td>190</td>
</tr>
<tr>
<td>7.2 External Results</td>
<td>194</td>
</tr>
<tr>
<td>7.2.1 Human Performance</td>
<td>194</td>
</tr>
<tr>
<td>7.2.2 Preprogrammed Agents</td>
<td>197</td>
</tr>
<tr>
<td>7.3 Single Phase Learning</td>
<td>199</td>
</tr>
<tr>
<td>7.3.1 Observation Only</td>
<td>199</td>
</tr>
<tr>
<td>7.3.2 Instructional Only</td>
<td>208</td>
</tr>
<tr>
<td>7.3.3 Experiential Only</td>
<td>212</td>
</tr>
<tr>
<td>7.3.4 Single Phase Summary</td>
<td>217</td>
</tr>
<tr>
<td>7.4 Two-Phase Learning</td>
<td>218</td>
</tr>
<tr>
<td>7.4.1 Observational then Instructional</td>
<td>219</td>
</tr>
<tr>
<td>7.4.2 Observational then Experiential</td>
<td>228</td>
</tr>
<tr>
<td>7.4.3 Instructional then Experiential</td>
<td>234</td>
</tr>
<tr>
<td>7.4.4 Two-Phase Learning Summary</td>
<td>240</td>
</tr>
<tr>
<td>7.5 Three-Phase Learning</td>
<td>241</td>
</tr>
<tr>
<td>7.5.1 Observational then Instructional then Experiential</td>
<td>242</td>
</tr>
<tr>
<td>7.6 Final Analysis Comparison</td>
<td>249</td>
</tr>
<tr>
<td>7.7 Approach Experimental Summary</td>
<td>255</td>
</tr>
</tbody>
</table>
7.8 Conclusions ......................................................... 258

CHAPTER 8 FUTURE WORK ........................................ 260

8.1 Reduce Effort ...................................................... 260
  8.1.1 Computer-Based Training ............................... 260
  8.1.2 Stopping Criteria ........................................ 261

8.2 Architecture Changes .......................................... 262
  8.2.1 Using Contexts ........................................... 262
  8.2.2 Modularity and Concurrency ......................... 264

8.3 Improve Learning ............................................... 266
  8.3.1 Alternative Crane ....................................... 266

8.4 Extensions ....................................................... 270
  8.4.1 Training Human Subjects ............................ 270

APPENDIX A LEARNED NETWORKS ............................... 271

APPENDIX B QUESTIONNAIRES ................................ 274

LIST OF REFERENCES .............................................. 275
LIST OF FIGURES

Figure 5.1 Linear Network for Chaser ........................................ 94
Figure 5.2 Feed-forward Network for Chaser with One Hidden Layer .... 95
Figure 5.3 Recurrent Network for Chaser with One Hidden Layer Fully Connected 95
Figure 5.4 Mutation Adding a Single Neuron ................................ 97
Figure 5.5 Mutation Adding Different Dendrites ............................... 97
Figure 5.6 Two Parent Genomes ............................................ 100
Figure 5.7 Aligning the Two Parents, Swapping Matching Genes Uniformly ... 101
Figure 5.8 Resulting Child Genome ....................................... 101
Figure 5.9 Aligning the Individual with Best Self and Global Best for PSO ... 108
Figure 6.1 Example of Chaser Interface .................................. 125
Figure 6.2 Example of Sheep Pie Wedges ................................. 127
Figure 6.3 Example of Sheep Interface .................................. 128
Figure 6.4 Example of Crane Interface .................................. 131
Figure 6.5 Example of Car Interface .................................... 133
Figure 6.6  Chaser Simulation Experiential, PSO Constants  . . . . . . . . . . . . . . . 142
Figure 6.7  Chaser Simulation Observational Violet, PSO Constants  . . . . . . . . 142
Figure 6.8  Chaser Simulation Observational Orange, PSO Constants  . . . . . . . 143
Figure 6.9  Chaser Simulation Experiential PIGEON, PSO Constants  . . . . . . . 144
Figure 6.10 Chaser Simulation Experiential, PSO Structure  . . . . . . . . . . . . . . 145
Figure 6.11 Chaser Simulation Observational Violet, PSO Structure  . . . . . . . . 146
Figure 6.12 Chaser Simulation Observational Orange, PSO Structure  . . . . . . . 146
Figure 6.13 Chaser Simulation Experiential, Chain Cutoff  . . . . . . . . . . . . . . 149
Figure 6.14 Chaser Simulation Observational Violet, Chain Cutoff  . . . . . . . . 149
Figure 6.15 Chaser Simulation Observational Orange, Chain Cutoff  . . . . . . . . 150
Figure 6.16 Chaser Simulation Experiential, Alternate Handoff  . . . . . . . . . . . 152
Figure 6.17 Chaser Simulation Observational Violet, Alternate Handoff  . . . . . 152
Figure 6.18 Chaser Simulation Observational Orange, Alternate Handoff  . . . . 153
Figure 6.19 Chaser Simulation Observational Violet, Algorithm Comparison  . . 157
Figure 6.20 Chaser Simulation Observational Orange, Algorithm Comparison  . 157
Figure 6.21 Sheep Simulation Observational Violet, Algorithm Comparison  . . 159
Figure 6.22 Sheep Simulation Observational Orange, Algorithm Comparison  . . 159
Figure 6.23 Crane Simulation Observational Violet, Algorithm Comparison  . . 161
Figure 6.24 Crane Simulation Observational Orange, Algorithm Comparison  . . 161
Figure 6.25 Car Simulation Observational Violet, Algorithm Comparison . . . . . . 163
Figure 6.26 Car Simulation Observational Orange, Algorithm Comparison . . . . . . 163
Figure 6.27 Chaser Simulation Instructional Green, Two Methods . . . . . . . 167
Figure 6.28 Sheep Simulation Instructional Green, Two Methods . . . . . . . 168
Figure 6.29 Crane Simulation Instructional Green, Two Methods . . . . . . . 170
Figure 6.30 Car Simulation Instructional Green, Two Methods . . . . . . . 171
Figure 6.31 Chaser Simulation Experiential, Algorithm Comparison . . . . . . . 174
Figure 6.32 Sheep Simulation Experiential, Algorithm Comparison . . . . . . . 175
Figure 6.33 Crane Simulation Experiential, Algorithm Comparison . . . . . . . 176
Figure 6.34 Car Simulation Experiential, Algorithm Comparison . . . . . . . 178
Figure 6.35 Chaser Simulation Observational Violet, Alternate Long Run . . . . . . 183
Figure 6.36 Chaser Simulation Observational then Experiential Violet . . . . . . 186
Figure 6.37 Chaser Simulation Observational then Experiential Orange . . . . . . 187
Figure 7.1 Chaser Simulation - Observational versus Experiential . . . . . . . 200
Figure 7.2 Chaser Simulation - Joystick Movement Comparison . . . . . . . 206
Figure 7.3 Chaser Simulation - Instructional Population, Green . . . . . . . 210
Figure 7.4 Chaser Simulation - Joystick Movement Comparison . . . . . . . 213
Figure 7.5 Chaser Simulation - Position Comparison . . . . . . . . . . . . . . . . 213
Figure 7.6  Chaser Simulation - Observational then Instructional ............................................. 221
Figure 7.7  Sheep Simulation - Observational then Instructional ............................................. 222
Figure 7.8  Crane Simulation - Observational then Instructional ............................................. 223
Figure 7.9  Car Simulation - Observational then Instructional ................................................. 224
Figure 7.10 Chaser Simulation - Observational then Experiential .......................................... 229
Figure 7.11 Sheep Simulation - Observational then Experiential ............................................ 230
Figure 7.12 Crane Simulation - Observational then Experiential ............................................ 231
Figure 7.13 Car Simulation - Observational then Experiential ................................................. 231
Figure 7.14 Chaser Simulation - Instructional then Experiential ............................................. 235
Figure 7.15 Sheep Simulation - Instructional then Experiential ............................................. 236
Figure 7.16 Crane Simulation - Instructional then Experiential ............................................. 237
Figure 7.17 Car Simulation - Instructional then Experiential .................................................. 238
Figure 7.18 Chaser Simulation - Joystick Movement Comparison ........................................... 243
Figure 7.19 Chaser Simulation - Observational then Instructional then Experiential 245
Figure 7.20 Sheep Simulation - Observational then Instructional then Experiential ................. 246
Figure 7.21 Crane Simulation - Observational then Instructional then Experiential ................. 246
Figure 7.22 Car Simulation - Observational then Instructional then Experiential ................. 247
Figure 8.1  Crane Simulation - FALCONET Context Split ..................................................... 268
Figure A.1 Best FALCONET Network for Chaser .......................... 271
Figure A.2 Best FALCONET Network for Sheep .......................... 272
Figure A.3 Best FALCONET Network for Car .......................... 273
LIST OF TABLES

Table 6.1  Comparison of Different PSO Constants .......................... 141
Table 6.2  Comparison of Different PSO Structure .......................... 147
Table 6.3  Comparison of Different Chain Cutoff Percentages .............. 148
Table 6.4  Comparison of Different Alternate Handoff Values ............... 153
Table 6.5  Observational Learning - Comparison of Algorithms on Chaser ........ 156
Table 6.6  Observational Learning - Comparison of Algorithms on Sheep .......... 160
Table 6.7  Observational Learning - Comparison of Algorithms on Crane .......... 160
Table 6.8  Observational Learning - Comparison of Algorithms on Car ........... 162
Table 6.9  Instructional Learning - Comparison of Two Methods on Chaser .......... 167
Table 6.10 Instructional Learning - Comparison of Two Methods on Sheep .......... 168
Table 6.11 Instructional Learning - Comparison of Two Methods on Crane .......... 169
Table 6.12 Instructional Learning - Comparison of Two Methods on Car .......... 171
Table 6.13 Experiential Learning - Comparison of Algorithms on Chaser ........... 173
Table 6.14 Experiential Learning - Comparison of Algorithms on Sheep .......... 175
### LIST OF ACRONYMS/ABBREVIATIONS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACO</td>
<td>Ant Colony Optimization</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Networks</td>
</tr>
<tr>
<td>AR</td>
<td>Augmented Reality</td>
</tr>
<tr>
<td>AAR</td>
<td>After-Action Review</td>
</tr>
<tr>
<td>ART</td>
<td>Adaptive Resonance Theory</td>
</tr>
<tr>
<td>CPPN</td>
<td>Compositional Pattern Producing Networks</td>
</tr>
<tr>
<td>CxBR</td>
<td>Context-Based Reasoning</td>
</tr>
<tr>
<td>FIR</td>
<td>Finite Impulse Response</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithms</td>
</tr>
<tr>
<td>GP</td>
<td>Genetic Programming</td>
</tr>
<tr>
<td>HMM</td>
<td>Hidden Markov Model</td>
</tr>
<tr>
<td>IIR</td>
<td>Infinite Impulse Response</td>
</tr>
<tr>
<td>LTI</td>
<td>Linear and Time Invariant</td>
</tr>
<tr>
<td>MLP</td>
<td>Multi-Layered Perceptron</td>
</tr>
<tr>
<td>NEAT</td>
<td>NeuroEvolution of Augmenting Topologies</td>
</tr>
<tr>
<td>PNN</td>
<td>Probabilistic Neural Network</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>PSO</td>
<td>Particle Swarm Optimization</td>
</tr>
<tr>
<td>PbD</td>
<td>Programming by Demonstration</td>
</tr>
<tr>
<td>RL</td>
<td>Reinforcement Learning</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machines</td>
</tr>
<tr>
<td>VR</td>
<td>Virtual Reality</td>
</tr>
</tbody>
</table>
At this point in time in the continuum of computers, it can be agreed upon that building an intelligent agent model from scratch is a difficult, expensive and time-consuming task. It is the general hope that there should be an easier means of creating an agent for a specific purpose that works well in that domain. There are many domains that use some form of computer-based entity to perform a task, both purely on a computer or in the real word. Each of these entities needs some form of intelligent agent to perform the task in a manner similar to a human. For example in a war game simulator, each of the different units may be controlled individually by a different agent. Because these agents are meant to be an analogue of a human in the same scenario, they are expected to react in a competent and intelligent manner that would be externally indistinguishable from a human controlled entity. Therefore, the agent should not act in a way that produces jerky motions or distinct changes in actions because those would seem less human-like, even if they were actually the optimal motion. Additionally, as a side task, it may be determined that each agent should act in an individual style that produces a set of actions that could be considered wrong in a given
situation, such as modeling a group of trainees. The individualism of each of the entities can be key to producing a convincing simulation. To program each of the agents explicitly would be a long and arduous process, for both the time to analyze the behavior to make a model and the physical process of encoding them into a program.

Building software agents that behave in a human-like fashion in simulations has been a difficult process. Traditionally, these have been built manually, through an arduous process of interviewing knowledgeable individuals (experts) and thereafter coding their knowledge in some machine understandable paradigm. This process, although successful, has represented a barrier to the development of such agents as a result of their high cost and long duration. This obstacle was reported in the literature back in the 1980’s by Feigenbaum [Feigenbaum 1984] when he coined the term “knowledge engineering bottleneck.” In spite of much research and some improvements in the situation, it remains true to this day.

The work described in this dissertation seeks to replace this manual method with one that is less expensive and of shorter duration. The inspiration comes from the way humans learn to perform tasks. The entirety of the investigation described in this dissertation is to take a multi-phased learning process that is traditionally emphasized in human learning, and formalize the procedure to an autonomous agent.
1.2 Human Learning

The four main phases of human learning envisioned by this research are historical, observational, instructional, and experiential. This process in a human-inspired sense can be related to a child learning how to play baseball, which will be used to illustrate a theme.

1.2.1 Baseball Allegory

Historical learning deals with past information, whether it is exemplar or doctrine. This process is fully supervised because the answers are prescribed. This phase of learning is associated with what may be referred to as “book learning.” All information learned is based on what is already known to be the “correct” answer. A child can read a book on baseball, learn all the rules, and study the statistics of all the greatest players. From the rules, he can understand the game and be aware of what should happen while playing. With the statistics, the child could figure out what level of play would be considered average performance and what is the best.

Historical learning may serve to “learn” how to play the game, but not the intricacies of physically playing it. This can be gained from observation when the child can watch the form of his favorite player on how that player batted, caught, and threw. He can also observe all the situations of when to bunt or steal a base were called for. All of this can be done with preexisting information and studied without ever touching a bat, ball, or glove. By simply
watching games on television, the child can observe how professionals act during a game and pick up habits to imitate. Just by mimicking the behavior and doing what others are doing will allow the child to play a passable form of baseball.

At a later date, the child, that has studied all the observable aspects of baseball, finally gets to play a real game on a little-league team. When playing on a team, the child is being observed by his coach. In playing the game, the child lines up to the plate just like Joe DiMaggio in perfect form to hit the ball. After some unsuccessful swings, the coach comes in to correct the child slightly to a form more suited to his stature and abilities. Later while on first base, the child recalls a time just like this one where Rickey Henderson stole second base and attempts the steal. In the attempt the coach calls off the runner because he did not believe the child could outrun the throw. While there are many more related stories, the point is that, while it is good to have all of information, mistakes can be made when purely basing information on historical or observed data when acting on a given situation. A proper coach will be able to analyses his players and give helpful advise to improve.

Finally, the child, knowing the rules and given instruction by a coach on how to play better based on his capabilities, can then practice on his own or during games. While batting, the stance he observed from Joe DiMaggio and then modified by the coach is working well for the child, giving a good batting average. But when playing more games, the child finds that if he bends his knees a little more he reduces strikeouts and gets on base more. This evolution of the stance has given him greater ability to play that was learned on his own
through practice. After all the knowledge gained from others, the child must gain experience from playing the game in order to gain greater improvements.

For a human to learn a game like baseball, it is a natural process to find out information about the game such as how to play and what is considered good; then to observe others playing and mimic what they are doing; then get instruction by a coach that gives hands-on training; and finally just play the game and improve more through practice. This same procedure is followed for other games as well.

1.2.2 Computer Comparison

There are many similarities in the way in which a computer agent can be trained to perform a task and the human learning process. In agent creation, the historical information would be directly programmed as rote actions. We liken historical learning to manual agent creation, and not necessarily to true machine learning per se. However, it could also be related on how to frame the agent so later learning can be done. The inputs and outputs need to be analyzed on how to interact in a simulation and coded to fit within the rules of that domain. In addition, a metric must be created to judge how well the agent performs.

For the Observational phase, a database of information can be fed to the algorithm. An agent can be created based on the numerical similarities found through observation of intelligent individuals. Statistics of what actions were performed during example situations can
be used to determine actions for other situations the agent is presented with. Many methods stop at the observational stage, (Dejong and Mooney 1986) (Lee and Shimoji 1991) (Sammut et al. 1992) (Henninger et al. 2001) (Fernlund et al. 2006). Some claim that learning is done and the agent can achieve sufficient performance based on both the training and validation data with observation alone. This information can be used to create an agent based on human knowledge. However, the problem arises when decisions are made based purely on the closest observed situation without adjusting for differences in the external environment or internal abilities. When an agent is introduced to new situations or transplanted out of its current domain into one that is similar, actions that would seem unintelligent may be done because the observed data is no longer equatable.

These differences can be fixed in the Instructional phase with an expert coach. This phase is semi-supervised because the “right” answer is generally known, but it may not be explicit. Since this phase is done on live data or in real-time, the actual correct answer may not be known until after the fact. The “right” answer during the run would be an answer that is just “not wrong” in that particular situation. It can be said that while it may be difficult to define what is right, it is usually easier to define what is wrong. Reinforcement learning algorithms work well in this field by using negative reinforcement to actively discourage certain policies from being done by an agent. A proper coach could spot problems to fix the actions of the current agent when it is acting incorrectly. This information can be incorporated later without the need to add to the pile of data and train again from the beginning. An outside instructor can be used to simply give positive or negative points to the agent and this
guidance can be used to influence future decisions. With computers, this can be done from scratch (i.e. no a priori knowledge at all) with an initially random agent; however, the theory is that an initially trained agent that knows the general rules and situations should fare better. Instruction can be used to modify some habits or inconsistencies that the agent learned during its original observational training phase. The problem with full instruction is that it requires a live person to actively coach the actions of an agent in real-time. Real-time, in the simulation sense, is considered very slow, where one second equals one second. This means that thirty hours of training time will commit a human trainer for many days. Humans may be good trainers, but they do not have infinite patience to go through every possible encounter that might arise. This problem calls for another phase to fill in the gaps.

The Experiential phase can be done in which the system will practice on its own in many different situations that are slightly varied in order to improve itself. This phase is not supervised by a human with only the simple feedback of whether it was successful at the end of the run on some metrics of performance. Evolutionary computation techniques such as Genetic Algorithms (GA) and Genetic Programming (GP) operate in this domain by trying to evolve new things in many different agents and those that do better are kept and those that do worse are discarded. In this way, slight variations can be made in order to improve efficiency or find a way to account for data that may not have been seen before. In the computer world, this can be done in hyper-time with hundreds of simulations in seconds that can cause adjustments in actions to improve the capabilities of an agent in a significantly shorter time. This process can also be done on its own with an initially random
set of agents. However, being first bootstrapped with human observed information then coached to be better, an agent should have an initial starting point as being pretty good and increase its capabilities to account for more situations that have not been explicitly learned or that are not fully enumerated.

1.3 Learning Approaches

As any human knows, there are many ways to learn. As defined here, learning approaches represent different ways to for a human to perform learning by arranging the data or experiments in a specific way. These approaches are the learning processes and not the method of specifically causing the learning in an individual. Likewise, it is the same for computers. In the most generic sense:

“A learning process is activated by the input information, obtained from a teacher or from a learner’s environment (external or internal). Such a process involves the learner’s prior knowledge (background knowledge), and is motivated by the learner’s desire to achieve some goal (to solve a problem, to understand given facts or observations, to perform a task, etc.).” (Michalski 1993)

It is an approach to learning that we define in this dissertation, regardless of what specific machine learning method is followed. The approach is more related to how the data are perceived and given to the learning system. Depending on the exact intention, the dataset
can be framed in such a way that agents can be created using any one of the many different learning methods. However, some approach paradigms work better with one specific learning method than do others. The three approaches to learning examined here are: Observational learning, Instructional learning, and Experiential learning. These approaches, respectively, fall into the mode of supervised, semi-supervised, and unsupervised learning.

In supervised learning, the exact input and output are known in advance. The purpose would then be to have a system that generalizes or performs induction to create a mapping from the inputs to the outputs. In semi-supervised learning, not all mappings are known, but positive or negative suggestions can be made to guide the system at different states during operation. In unsupervised, no training influence is made during operation, but a score is given at the end judging the performance. These can also be decomposed into the learning stages of a person that would first silently observe someone else perform a task, then be shown how to do it by a coach or teacher, and finally practice performing the task until the skill improves. We now discuss these approaches individually in depth.

1.3.1 Observational Learning

One of the main obstacles to be overcome in the creation of intelligent agents and making them commonplace is to reduce the effort involved in creating them. Observational learning seeks to minimize the role of the knowledge engineer and, if the system is robust, possibly even that of the programmer. This, of course, has the effect of making the process more
automated, less expensive, faster and possibly with fewer errors because there is no human go-between. Humans learn much of what we do via observation. It is the old “here, let me show you” approach to teaching and learning. In the Observational learning process:

“The human needs no knowledge of the internal workings of the system, and the human expert is never asked to articulate the information or methods he uses - a step which has proven to be a bottleneck in current knowledge-based system developments.” (Dejong and Mooney 1986)

Observational learning, in its most basic form, is the process of recording externally-available environmental variables and the actions of a human performing a task without interaction with the human performer, or explicitly inquiring about his/her internal mental state. All data recorded are an objective record of the training event. Thus, the approach is to collect data from an already intelligent agent such as a human performing a task, collect both the inputs and outputs of the human, and attempt to discover the mapping between the actions chosen for the situation. Because the human is considered intelligent, an artificial agent performing equally proficient in a given situation would be said to be performing intelligently.

Other reasons for learning from observation relate to the expert themselves and how they often find it difficult to articulate their expertise.
“human operators cannot exactly and quantitatively describe their control schemes and strategies. Namely, as they become experienced, they lose awareness of the detailed process in interpreting sensory information.” ([Yang and Asada 1992])

This is especially true in the case of motor skills. For example, in learning how to suddenly stop a moving car, an expert asked about how much pressure she places on the brake might not be able to describe it properly in words. Observational learning, in effect, tells the expert “don’t tell me, show me.” The small details are no longer considered; they just happen when performing some task. However, it is important to have this information because “the unconscious skills of the experts can be subtle and important, particularly in the reflex response.” ([Lee and Shimoji 1991]). When the behavior/performance is recorded, all the little actions can be actively observed and thereafter modeled using a machine learning method that can then learn those skills.

Observational learning has been applied to learn the skills for many tasks. In ([Lee and Chen 1994]), the authors attempt to create a non-linear controller from the observation of a human controlling the system. In ([Kaiser and Dillmann 1996]), the authors attempt to learn how to insert a peg using a robotic arm after a human demonstrates the same action with a joystick. In ([Atkeson et al. 2000]), the authors attempt to take this work to another level by creating humanoid robots that are able move and function using the same physical characteristics as person. In ([Morales and Sammut 2004]), the authors attempt to create an autopilot from the observation of a human in a simulator. It is already known that these
tasks can be performed by a human, and therefore it is left to the learning system to figure out how they did it so that it can be repeated in a general fashion.

Learning from observation can take an entire data stream. Therefore, some effort must be made by the programmer to divide the data in order to facilitate learning. In (Bentivegna and Atkeson 2001), the authors identified a series of primitive actions that are performed in a task. The data are divided into subsets that only relate to that action, and trained separately to learn and refine those actions. In (Gonzalez et al. 1998), the authors used Context-Based Reasoning (CxBR) as a mental means to divide the data based on the context. A context is defined by a certain state of the environment that dominates procedure. Because in a context “only a limited number of things can be expected to happen” (Gonzalez et al. 1998), the training for any individual context is reduced. However, now transitions between contexts must be discovered and the data manually partitioned, although further research is being done to reduce this burden (Trinh 2009).

Because the term observational data was defined as a series of variables from the environment and the actions of a human, it would seem very natural to use data-based algorithmic methods. In (Sammut et al. 1992) and (Isaac and Sammut 2003), the authors used decision trees (DT) to create a series of rules to control an autopilot based on the observation of a trainer. The method is a direct implementation of a DT to learn motor skills by using recorded information to create a purely reactive flier. The system lacked memory and an entirely new tree would need to be created if new information was observed. In (Harries et al. 1998), the authors improved upon the method by making a forest of trees based on contexts
and slice the data when new observations significantly differed from agent expectations. In either case, the systems only had a discrete number of decisions and lacked the adaptability desired.

Artificial Neural Networks (ANNs) also appear to be a good method for learning with observational data because an exemplar of input and output pairs exists to perform learning. In (Fix 1990), the author modeled the behavior of drivers in traffic changing lanes in on a circular track. The author found the system could learn the speed and lane in which to be in order to avoid collision. However, preconditioning of data was required to sort the other cars, the inputs specifically had to be set to the number of cars on the road, and a time-delay input feature was required to construct some memory. In (Lee and Shimoji 1991), the authors used an ANN for skill transfer from an expert; however, they note that proper adaptation “depends on the complexity and structure of the target system.” In (Sidani and Gonzalez 1994), the authors used a recurrent ANN to give a time perspective and memory to the problem. Additionally, the system used Fuzzy Artmap (FAM) to group data for situational awareness and calls the right ANN for training. Each of these systems used ANNs trained on observed data and were shown to create an agent. However, the problem was that structure of the network was left to the programmer and it was unlikely to be optimal. Nevertheless, this problem has been overcome to some degree by systems such as NeuroEvolution of Augmented Topologies (NEAT), which has been successfully applied to car simulations (Stanley et al. 2005a). In this way, the system can then “automatically determine what information is useful in making the decisions.”
Learning from observational data has also been done with GPs. In [Fernlund et al. 2006], the authors created a system, called GenCL, to model a human driving a car in simulation. This type of system has the ability to create complexity by using inputs in any arrangement in the Koza tree [Koza 1992]. This system took recorded information of several drivers in a simulator and provided a micro-simulator to compare the results of the human with the AI agent. CxBR was used on a manually-divided dataset to reduce the amount of information needed to be learned by the GP. Additionally, the system also used the GP to determine the transitions between contexts, further reducing the human interaction. The advantages of the system were the ability to create a complex agent without defining the structure and the ability to produce human readable code from the system that could be analyzed. The disadvantage of the system was the lack of internal memory (with the exception of the past outputs that were passed as inputs for an external recurrent link) and having to separate the data into contexts. In [Trinh 2009], the author extended this work by using various state-of-the-art partitioning and clustering techniques called COPAC in order to automate the contextualization of observed data. The partitioned and the number of contexts are then processed by the existing GenCL system to produce the final agent.

However, there are some problems relating to learning from observation. In [Friedrich et al. 1996], the authors find a problem in their model called Programming by Demonstration (PbD) in which the teacher plays a critical role in the way they teach. The problem “occurs as soon as the operator uses sensors that are not available to the robot or if the operator employs mental models that allow for partially replacing actual robot perceptions.” A simple
car example would be driving in a city with a series of traffic lights. The human sees all of
the lights in the distance and make intelligent decisions about slowing down. However, if the
system is only passed the state of the closest traffic light at any point in time, it may not
be able to adjust or may attribute actions to the wrong information. In 1991, the author found two main problems with the observation approach. The first deals with
drivers who are too good, resulting in a system that, in the words of Pomerlaeu, “will never
be presented with situations where it must recover” and the second is “overlearning from
repetitive inputs.”

These pose problems with the data collected and the way learning from observation is
performed. The proposed fix to the first problem is to create abnormal training scenarios from
which the human experts must extricate themselves. However, this step requires a degree of
interaction because it involves an additional intrusive step and not purely observing. The
second problem is also difficult because trouble spots are usually only a small percentage
of the overall input data. It would require human interaction to find the problem areas
and focus the system in that area. This is because most learning systems will attempt to
minimize areas for the most data possible, to the detriment of the minority cases. Some
research has been done to reduce this limitation of correlating the AI’s responses with the

Observational learning is the pure form of capturing human intelligence. It is typically an
off-line stage where all the information from the human expert is fixed since once captured
the data can no longer be modified. In this way, the data can be large and historical, with
many recorded situations where the artificial agent can learn to be similar to a human. This human-like agent may not have learned optimal performance, but should be competent for the given situations. However, the problem is that “it is not possible to anticipate all possible situations while collecting data” (Stanley et al. 2005a). The main recourse in fixing a system that errs during the performance stage because of lack of breadth in its learning would be to bring in the expert again and train for that specific situation, and start the learning process over. This can limit the scalability of such a system. Finally, because the data is fixed, there is no new information for the system through which to adapt. Even though Observational learning still suffers from unresolved issues, it represents a potentially revolutionary way to build models automatically. We next discuss one way to overcome some of these unresolved cases.

1.3.2 Instructional Learning

Instructional learning is another approach used (Kosuge et al. 1991) (Gillespie et al. 1998) (Coelho et al. 2001) (Schaal 2003) (Goodrich and Quigley 2004) to create intelligence agents without any direct programming. Instructional learning, also called coaching, as defined here is a method for learning on the fly from a coach. It is similar to Observational learning except that the learning happens with real-time feedback from the trainer. The relationship between the teacher and student is dynamic, where the trainer adjusts the action to specifically account for what the student is doing.
There are several ways to provide the student-teacher relationship for the motor skill learning task, as discussed in (Gillespie et al. 1998). Three methods are described by Gillespie using the example of someone learning to play tennis. In the first method (Indirect Contact), the student and teacher each grasp the tennis racket at different points and the teacher swings. In this method the student will learn the position and motion but cannot perceive the amount of force required to implement the reaction. In the second method (Double Contact), the student grasps the racket and the teacher grasps the student’s hand. Using this method the student gets the motion, but also the amount of force the teacher uses to swing the racket. The student also has multiple points of contact feeling the dynamics of the racket and the teacher’s hand. In the third method (Single Contact), the teacher holds the racket and the student grabs the teacher’s hand. This is the most similar to observations in that the student watches the racket swing but also feels the motion of the swing without touching the racket. These different instructional paradigms show different approaches for how to coach. Using these examples, it can be interpreted that the demonstration of the motion is important, but so is the force while doing the motion. Additionally, because the teacher’s hand is there, it can provide live corrections of the student’s swing and the student can feel the feedback. In Gillespie’s system, the motion and feel of these systems are implemented by using haptic-based force-feedback devices. Overall, this training approach is the most underdeveloped for training intelligent agents, and could be key in making them operate more human-like. These ideas are discussed in greater depth in Chapter 2: Haptics.
In (Schaal 2003), the authors implement an instructional system on a robot platform. In their system, the authors attempt to overcome the old system of observation which “normally consisted of manually pushing the robot through a movement sequence.” They attempt to learn a control policy that is “bootstrapped by watching a demonstration” (Schaal 2003) but then follows the teachers trajectories by imitation while adjusting for kinematics and dynamics. In (Kosuge et al. 1991), the authors use a “hybrid of force and position control” by having the human teacher move the end actuator around in a robotic arm and then “monitor signals between master and slave manipulators.” Using the force applied, the system creates a set of control laws that monitor the input force, and not just the position.

The system does not have to exist as wholly independent, as already demonstrated in (Schaal 2003). It would likely be difficult to train any system by coaching alone because it could initially be erratic. Furthermore, long-term training is time consuming because Instructional training must be done in real-time. If the system could already perform the task and just needed to be refined, the coaching process becomes much easier by only training when the behaviors are different. The system is penalized for acting differently, and rewarded when it acts similar to the human during a run. As an on-line training strategy, a reinforcement method would match well with this type of learning because those methods are explicitly able to operate on the type of data being presented. The learning method would have to be able to adapt to a reward and punishment scheme. This type of learning process seems best suited to the class of Reinforcement Learning (RL), Ant Colony Optimization (ACO), or Particle Swarm Optimization (PSO) systems by the nature of the corrective data.
The main flaw of Instructional learning is the amount of time it takes to train. Because one second equals one second in real-time, the expert would need to spend many hours just tuning the system. However, because the trainer is there, problems that arise during operation can be immediately resolved by adding more reinforcement. The process is the definition of adaptability by allowing the coach to change things on-the-fly. In summary, coaching in the Instructional stage can best provide the refinement to an existing model in a way that overcomes some of the issues of Observational learning, such as the inability to adjust for errors seen in the performance phase without retraining the system from scratch with newly introduced observations.

1.3.3 Experiential Learning

Experiential learning is dissimilar to the other two approaches because there is no right answer at any specific time instance, but incremental rewards or a score given at the end of operation, like an after-action review. An after-action review is defined as:

“...a way for a team to reflect on and learn while they are performing...The objective is to learn as you perform, understand why the interim objectives were or were not accomplished, what happened, what lessons can be learned, and what can be quickly driven back into the performance process.” (Baird et al, 2000)
The approach is similar to human practice when performing a task, where the system performs a set of actions over and over again in different scenarios and is graded on performance. This is the typical operating method of a GA/GP, where the fitness function determines the training or of RL, where the points gained during operation affect later actions. There is no direct teacher in this approach, only a series of scores that eliminate human involvement beyond setting up the different scenarios. The system can then be exercised numerous times in an attempt to see all possible problems.

However, this system does not exist in a vacuum either. The initial training of the agents in this fashion would likely be unsatisfactory because they are typically based on random numbers for the original weights or coefficients, although it could eventually learn an optimal solution. In (Schaal 1997), the authors combine learning from demonstration and experiential learning to create a policy for the pole balancing problem. The main thought is that the demonstration (observation) is done to prime the system (bootstrap) to produce a policy. Then the Experiential learning takes over to repair the policy, although it is being done on-line through the reward of a single step. It is possible that the demonstrated states would not encompass all the reachable states. Thus, a working agent may only work in a stable region without any knowledge of bad regions. Using a reinforcement learning system, the reward for the given action would eventually be tracked back to all other states thus giving a global policy.

Experiential learning as a whole does not directly reflect modeling a human, however, it does model the process in which a human learns. By itself, Experiential learning is
a computerized form for creating agents that in the end could be intelligent, but which learned the actions on its own. The actions may be optimal but could be very unhuman-like by exploiting random features in the simulation. However, it did require the least amount of effort by removing the need to have an expert at all. The system can be said to scale to infinity because it is only constrained by computation and can adapt as a result of the nature of the training features of the environment that are continually changing.

In summary, the Experiential learning approach can address a major problem with Observational and Instructional learning concerning unseen situations if the goals is to produce higher performing agents. It would be nearly impossible to observe every situation or coach for all problems. Experiential training allows for many more situations to be encountered by the learning system in order to create a better agent. In a reciprocal process, the addition of Observational and Instructional learning to Experiential learning can quicken the process by introducing a bootstrapped and competent agent. This bootstrapped agent can increase the chances that the final agent will act similar to a human since it was originally based on a human.

### 1.3.4 Learning Summary

In this chapter we discussed how it is difficult and expensive to build human-like agents by hand. Human learning gives us the inspiration to seek ways to teach an agent how to act without doing it manually. We put together the two conventional learning approaches (Ob-
servational and Experiential) and introduce a third one (Instructional) that we hypothesize will facilitate the other two. We continue and expand on the discussion of Instructional learning by discussing haptics as the means to interface with the agent in the Instructional phase of learning.
CHAPTER 2
HAPTICS

We intend to use a force-feedback system for training by means of Instructional learning to overcome the deficiencies of observational learning by adding augmented sensing of agents. To gain an understanding of how this stage can work, a background in haptics is essential. In this chapter we discuss the state-of-the-art in haptics.

2.1 Haptics and Telepresence

Haptics is the study of the sense of touch. In computer terms, haptics refers to interaction with a computer system by means of touch. In [Srinivasan and Basdogan 1997], Srinivasan defines a haptic interface as one that “displays tactual sensory information to the user by appropriately stimulating his or her tactile and kinesthetic sensory systems.” The human tactile system is defined later in that paper as “the sense arising from the skin in contact with [an] object” and the kinesthetic system is defined as “the sense of position and motion of limbs along with the associated forces.” Typically, the tactile is associated with interactions involving using one’s hands for sensory input by feeling the surface or vibration of physical objects. Kinesthetics can be associated with knowing where one’s hands are in three
dimensional (3D) space without looking. On the most basic level, mice and joysticks can be considered haptic devices because they provide input to the computer and passive haptics as one feels the interaction with the device. Additionally, one would use his/her sense of position to press a button or move one's hand to accomplish a desired input. However, computer haptics usually refers to an active force-feedback system within a device. Some examples of force-feedback are joysticks that vibrate when a player is hit in a video game or a car steering wheel in a simulation that provides artificial resistance when turning. Haptics are used in simulation and Virtual Reality (VR) to give a better sense of “being there.” These devices provide additional sensory cues that are presented to a human in a natural way. Because humans have evolved over millions of years to combine together all available sensory cues in a multimodal input in order to survive (Mitchell 2002), this extra haptic information can be combined with computer visuals to help in the decision making process.

2.1.1 Haptic Interaction

Haptic interaction with a computer has been a field of research for over 50 years (Fisher et al. 2004) when computer scientist were looking for new and innovative ways to get and receive information from the computer. The 1960’s introduced teleoperation in which remote devices were controlled by a joystick. The 1970’s brought Braille-based tactile displays. The 1980’s saw the first force-feedback teleoperation, where the environmental effects were transferred back to the controller. By the 1990’s, the first major haptic device, the Phantom by SensAble,
went for sale, and the first haptic joysticks and gloves were sold by Immersion Inc. Haptic devices utilizing force-feedback exist now, but are still not commonplace, being mostly used in game/simulation controllers, medical training, teleoperation, and very recently in automotive applications.

In the past decade, many new devices were created to attempt to take advantage of the sense of touch. Many haptic interfaces used currently are meant to mimic the controls of the real-life feedback of vehicle controls in simulation, namely joysticks and steering wheels. These devices can convey a force through embedded motors to stimulate the sense of touch. However, joysticks and steering wheels are not the only haptic interfaces. In an ambitious project call the Rutgers Master II (Bouzit et al. 2002), a pneumatic cyberglove was made that can both sense finger tip position and provide a feedback force. Using this system, a user can grip items in a simulation and be able to tell its size, shape, and hardness. In (Chen et al. 2006), the authors created a similar arrangement for the legs using an exoskeleton for controlling a robotic pair of legs. With the haptic feedback, the human acts as the control loop using the knowledge gained through years of bipedal motion. Not all haptic interfaces need be complicated. In (Deligiannidis and Jacob 2006), the authors constructed a virtual scooter for a virtual reality simulator. The scooter had fans attached to the front which blow air on the user’s face which is proportional to the current speed. This simple interface gave extra inputs that were very natural and almost instantly understandable.

The sense of touch has also been shown to be fairly quick with kinesthetic sensing operating at 20-30 Hz, sensorimotor control at 5-10 Hz, and tactile sensing up to 10,000 Hz.
It is also stated that “Haptic feedback is best for interfaces that are complex, sensorially overloaded, and require continuous control” (Fisher et al. 2004). Haptics are said to be a “high-bandwidth modality” and “reduce mental workload” (Haanpaa and Boston 1997). Additionally, haptics can be used as cues for information transfer, as it has been shown that human participants can learn different vibrations almost subconsciously in under 25 minutes (Enriquez et al. 2006). The main focus for this dissertation, however, is in the process of motor skill transfer. In (Avizzano et al. 2002), the authors taught individuals to draw shapes using a pen with haptic feedback. A shape would appear on the screen and the user had to draw the shape without thinking about it in non-cognitive fashion. It was found that without error information, people had the tendency to repeat same incorrect action. Then in a second stage, a spring-based attractor was applied to push the pen to stay on the path. The participants were then able to improve the skill beyond just the visual information and kept the skill after multiple trials. In (Feygin et al. 2002), participants were asked to move a Phantom 3D joystick on a given path in three dimensional space. The authors found that complex motor skills that are difficult to explain and describe verbally or even visually, can be conveyed easily by haptic means. The results supported the hypothesis that visual cues helped with position and shape, but haptic guidance was better with respect to timing. Additionally, it was shown that the combination of vision and haptics together were better than vision alone. Two perceptual modalities can be combined to produce increased accuracy.

In (Bingham et al. 2006), the authors examined the reactions of a human when not being able to see their own hands during a task. The participants were shown an object, which
was then covered by a divider, then asked to grab the object. The participants were always short of the goal, but when they were allowed to touch the object first without seeing it they were able to perform much better. The authors theorized that the sense of touch gave a better sense of expectation and changed how the humans interacted. In these experiments, the evidence points to the ability of haptics to impart improvements in reactions and skill over visual stimuli alone and potentially reduce the mental workload.

2.1.2 Virtual Reality and Teleoperation

Virtual reality (VR) and teleoperation are fields that have incorporated haptics. In VR, the main purpose it to put the user in an environment, which is typically done through a three dimensional visual display. The use of haptics is used to enhance presence. Presence is “construed as the private sensation of the user of being there in the virtual environment” (Casati and Pasquinelli 2005). In VR, it is the ability to feel objects or sense motions that give a greater understanding of the environment. This idea is further incorporated by teleoperation to improve operation. In teleoperation, a robotic vehicle is controlled at a distance through some form of remote control. A goal is to have telepresence, “the transfer of human senses to remote locations by feeding back sensory information from the remote environment” (Elhajj et al. 2001). If the user of the teleoperation can sense information from the remote vehicle such as collisions with walls or wind resistance, they can make better decisions about operations over just visual feedback. In (Ansar et al. 2001), the authors use
simulated forces for medical training by overlaying the accompanying forces that would be expected based on the visuals. This helps the fidelity of the simulation by providing forces that would be expected in the real world. In a different medical arrangement (d’Aulignac et al. 1999), the authors provided a method to train individuals on a virtual 3D leg using a haptic interface. The visual leg can be felt through the device based on actual leg pressure measurements.

2.2 Extrasensory Input

Another interesting aspect of haptics is that it can be used to convey information that normally would not be felt by human in the environment normally. These sensed feeling can help the operator make decisions or perform control.

2.2.1 Virtual Forces

One concept is the idea of virtual forces. These are forces that are felt through the haptic interface that are not felt by the remote system directly. Examples of this would be collision avoidance, monitoring, and advisory systems. In (Casiez and Chaillou 2005), the authors use a virtual spring force that is directed away from walls. This is a force the does not exist as part of the actual remote environment but instead attempts to make the user move
away from walls by making it harder to steer into them. This is a force that is felt directly through haptics and that is directly interpretable. In (Bretz 2001), the author provides an overview on the assistance of drive-by-wire in automobiles. Systems are being implemented now that provide haptic functionality to signal events or assist by making it harder to press on the gas when approaching a car from behind or making it harder to steer out of a lane. These are not overriding factors, but subtle differences in activation that can be felt by the driver, if only subconsciously. In this instance the system is acting as advisory system that is meant to alert but not control the vehicle. In (Clover 1999), the system is different in that it uses admittance theory where the system is semi-autonomous and robotic arm performs the given action based on the command. The system is in control and the only haptic feedback is the differences between what the user wanted and what actually happened. This was to promote safety by simulating artificial walls that do not allow the user to perform certain actions. These virtual forces do not actually exist but are meant to help the overall control to increase human performance.

2.2.2 Conceptual Understanding

Virtual forces can also be used for conceptual understanding. In (Ahmad et al. 2006), the author attempted to help the user detect ripple torque in DC motors which is a consequence of the electromagnetic (EM) field. An EM field is not directly visible. However, the “illusion of reality is created by providing the appropriate stimuli to the human sensing systems.”
The feedback is said to be unambiguous in nature. In (Bussell 2004), the author uses force-feedback to teach students the concepts of physics. Some students do not react well to visual cues but instead use kinesthetic learning. They grasp concepts from physical interaction. Force-feedback provided a way to directly feel the effects of changes in parameters for mass and gravity which would make the force increase or decrease. This concept can also be applied to springs, current, and other physical phenomenon. Through a future extension, it can be postulated that the feedback of even non-physical systems such as a learning algorithm could be felt from the parameter change that would too subtle for visual analysis. This type of system is seen in (Bruemmer et al. 2005), where a collaborative robot-human team is analyzed. The author found that “in order to realize the potential benefits of collaborative control, the human operator must be able to understand robot intentionality and predict robot behavior.” Typically, the entire cognitive burden is on vision. However, lower workload and more control was achieved through haptic interfaces by removing the vision and giving the user what the robot feels through vibration.

### 2.3 Training Mechanism

Haptics have been used in training in different ways and in different directions. The overall point is to produce a training system that feels natural. Haptics have been used for training in human-to-human, machine-to-human, and human-to-machine. Although our purpose is
to create intelligent agents, which would involve human-to-machine transfer, insight can be found in mechanisms that have been applied to teaching in other ways.

2.3.1 Human to Human Training

In human-to-human instructional systems, a person teaches another person to perform a task. This is a normal method of training, but in this instance haptics devices are used as the medium. The majority of these systems seem to be applied to medical training for surgery and human rehabilitation. In (Cai et al. 2003), the authors created a simulation for the insertion of a catheter in blood vessels. Normally, this would be done by practice, inserting the catheter into a dummy. However, using haptics, a virtual dummy can be constructed in simulation for tactile sensation but also the artificial forces of a teacher can be integrated into the process. A teacher doing the same task can be tracked, and the student can watch how an expert performs the surgery. In (Gunn et al. 2005), the authors created a VR system in which doctors from around the world could collaborate on surgical training. Each of the students had Phantom devices for force-feedback. In VR, the students could watch the teacher perform an operation, and in turn, the instructor could guide the hand of the student. This type of system mimics one where the senior doctor could guide the scalpel of a student if they were in the same room. In (Dinse et al. 2005), the authors believe that perceptual skills improve with practice for impaired individuals and that “Human haptic performance is not fixed, but subject to major alterations through learning processes.” Systematic stimulation
of a pattern through haptics can improve the timing and spacing, even if the exact motion is ignored. Just the feeling in the area makes it easier to learn and remember motions.

2.3.2 Machine to Human Training

Machine-to-human training is only a slight modification of the human-to-human training. The hand is still directed, except now no longer by a human teacher through the haptic device but by a virtual teacher. In (Gillespie et al. 1998), the authors attempted to train participants by having them feel the already known optimal strategy and then learn that strategy themselves. The system would guide the hand of the student in the cart balancing problem. They found that “if an operator is shown the analytically obtained optimal control early on they can bypass some of the usual practice time.” Therefore, if the students feel and see an already known existing answer, they do not need to discover one from scratch. This is the backward parallel of the bootstrapping method desired in creating an intelligent agent. A known solution exists, the system can then skip looking for alternatives. In (Bayart et al. 2005), the authors created a system that would teach an individual to draw Japanese ideograms. There is a specific way to draw characters in Japanese, and the authors created three different setups for teaching through haptics: a control where no haptic feedback was given; a full guidance where the users hand is dragged in the right motion; and partial feedback where teacher and the student are connected by a virtual spring. They found that the full guidance outperformed no guidance with respect to position and improved timing.
accuracy. However, they also found that partial guidance outperformed full guidance. They attribute this to the idea that “people learn from their mistakes” and that if the teacher provides full control, no mistakes are made and the student is unprepared for when actual issues come up because they grow a dependency on the teacher. In (Crison et al. 2005), the students are learning how to use a milling machine. There is a certain known procedure to prevent gouging metal using a certain speed and depth depending on the material being milled. The force-feedback response to the user was proportional to the penalty, which is the same as a spring-based method. They found that students that used the system outperformed students that were only shown how to operate the tool properly. As an interesting aside, experts did worse than novices with the system, possibly because they were expecting too much realism, or else the experts had a working system that differed from the “book version.” In the latter, the experts may have felt that they were better than the system.

2.3.3 Human to Machine Training

The actual method to be implemented in this dissertation is the human-to-machine training to produce intelligent agents using haptics. This method is less researched than the other methods and models the human forces versus the agent’s reaction. It borrows concepts from human-to-human relationships and this training system matches the instructional mode that is missing from the computer agent training models. In (Coelho et al. 2001), the authors use human development models to gain insight into robot creation. They believe that infants
have an incremental model for learning tasks in a temporal sequence. They break the task up into primitive states that are optimized through a set of rewards. This method, however, does not use haptics to learn intent but as feedback to the user while creating the original system, thus making it closer to an observational system. In (Aleotti et al. 2005), the authors used haptics as part of a Programming by Demonstration (PbD) model. The system used virtual fixtures for haptics for training the system. A basic system of vibration was implemented to demonstrate what the physical system could not do to the teacher. The human operators were said to learn tricks to complete and improve the system. This, in effect, used haptics to create a better trainer, but did not use the agent responses through haptics for training the agent. In (Eguchi et al. 2004), the author’s system was similar in that the haptics of the differences of the virtual system and the actual robot to be trained were fed to the teacher. Again, this improved the teaching to create better trajectories that were then learned by backpropagation. In (Goodrich and Quigley 2004), the authors learned to steer a car in order to stay in a lane. The system first started with a Proportional-Derivative (PD) controller to keep the system basically in the lane. Then the human trainer operated the vehicle while the PD was running and the haptics of the system was based on the admittance control policy. Then a Q-Learning reinforcement system was used to learn the corrective force the human used against the PD controller. In essence the system learned when not to trust the control law using the amount of force the teacher was willing to exert to overcome the basic control.

Still none of the aforementioned systems applies the action of the agents as a perceived force then uses the amount of force the teacher is willing to exert as a cost function. A
system that operates in this way has been currently undiscovered in literature. This would be a direct application of haptics through the conceptual understanding of the intent of the agent using the sensitive tactile and kinesthetic senses. This virtual force and the amount of teacher-exerted force could be used to train the agent. This system could be used to augment traditional learning from observation by only penalizing the places in which the agent and human response differ. The amount of penalty would be directly proportional to force exerted. It is hypothesized that all the usefulness of haptics, including the high bandwidth interface beyond visual, the ability to track time based motions better, the sensitivity of the sensorimotor system, and the natural feeling of input and output for a human could be used to an advantage in training agents.
CHAPTER 3
PROBLEM DEFINITION

This chapter clearly and concisely defines the problems addressed in this dissertation. These definitions build on the background and state-of-the-art discussed in the previous two chapters.

3.1 General Problem

In this technologically progressing world, the need for intelligent, human-like agents is quite apparent. Such agents could be used in military applications from simulated training assistants to autonomous members of the future war-fighter. Additionally, these agents could find their way into commercial products in personal vehicle monitors or in the entertainment market. A need for an intelligent program that can assist or perform the task of a human will likely be in great demand. The problem, however, is that the creation of these programs can be a long and difficult process. Under the current system, each agent typically has to be created manually by a programmer for the specific task, given knowledge of the operating environment and a list of design goals. This knowledge must be collected from someone who is (hopefully) an expert in that field and organized in a domain-appropriate manner. Then the knowledge must be disseminated to those individuals who create the agent but
who may not know the subject matter well. The developers then produce an agent based on that knowledge and proceed to produce a working system within the guidelines that they are given. Likely hundreds of man-hours later, a specific agent for that particular task is created, demonstrated, and either further refined or placed into use. The next time an agent needs to be created for a different task, the programmer either tries to isolate what similarities may exist from the old system or re-starts the process from scratch. Many different systems have been proposed to automate this process by means of automated capture, automatic clustering, or self-programmed systems. By replacing the manual steps of collecting and programming with automated processes, intelligent agents can become pervasive in a variety of fields because they will be easier to produce and take less time to create.

3.2 Specific Problem

One such solution is the Learning from Observation paradigm that collects inputs and outputs from a human performing a specific task. The main idea behind this operation is that if a human is given a situation, he/she will take an appropriate action, and these actions can be observed and captured by a learner. The inputs for the human are internal and external stimuli that cause the human to pursue an intelligent action. Those actions are observed and can be considered the outputs of the human system. Using the human as an ideal model, there must exist some calculable mapping from the inputs to the outputs of this model. Since the human subject can be considered intelligent, a computer model with a similar mapping
should therefore be considered to behave intelligently. The advantages of this system are that it takes the data collection from a subjective system of human experts to an objective system of computed reactions. There need not be an interpretation of results or a translation of meaning from the test subject of what they were trying to do, but only noting the explicit actions performed. The first step of data collection can, therefore, be considered complete.

The next step is the physical creation of the agent. This would normally be an analysis of the data collected from a human test subject and a development of a system that produces similar results. This step has also been automated through different means in the past using Artificial Neural Networks, Genetic Programming, and Reinforcement Learning. The major problem with this approach is that each of the methods can only be optimized based on the observed situations. It is assumed that the system has generalized for unseen situations and will react appropriately during the performance stage. If during testing the system does not perform as expected, the only option is to introduce additional observed data for that case and re-start the learning process, depending on the plasticity of the model. There needs to be a process to correct the system on-line. In this fashion, inappropriate behavior can be fixed quickly and effectively.

The proposed solution is to use haptics for feeling the agent intent and correct for problems directly. A Instructional phase of learning is added to allow a human trainer to coach the already bootstrapped agent. The haptic part augments this phase of the on-line training because it is used to increase the bandwidth, increase the sensitivity, and reduce reaction time of the human computer interaction for the information in which the trainer can impart
to the agent. Haptics have also been shown to be a natural method for human training that can be incorporated into agent training.

Furthermore, for the agent to be intelligent, it needs to be able to improve upon its original teachings and broaden its experience to operate in other similar, but not identical environments. An Experiential learning phase explicitly allow for this ability. Having the agents practice, and improve its ability on a larger set of situations should increase the chances of producing a more intelligent agent.

### 3.3 Hypothesis

The hypothesis is that the synergy of Observational, Instructional, and Experiential learning being presented to a single agent architecture, in that order, should produce a better agent than any single step or combination of any two steps. The implementation of this architecture is using NEAT, an ANN that uses GAs to create structure, coupled with PSO to optimize the weights can be used with a haptic interface to originally record data for the Observational phase, to provide force-feedback for reward/punishment during the Instructional phase, and to express the actions while practicing on its own in simulation during the Experiential phase. Therefore, the single joystick interface can be used to create high performing agents once the simulation domain is created.
3.4 Contributions

• A novel system for the automatic creation of intelligent agents that incorporates learning from observation, instruction and experience in one package.

• Promote the use of alternative input devices in human computer interaction by showing an application of haptic devices.

• Creation of a tool that could be used by non-programmers to produce intelligent agents in a known environment.

• Evaluation of different algorithms for comparison of performance

• Introduction of a novel algorithm that is a hybrid of several others from literature (NEAT and PSO)

• Contribute data to the research community on the three types of learning and how they can be combined together.
CHAPTER 4
ALGORITHM ANALYSIS

The goal of this chapter is to review the literature related to manual, semi-automatic and automatic methods that have been invented to produce intelligent agents. It will be a rather long survey of algorithms which will discuss the merits and defects of each. Finally, a discussion on the combinations and hybridization of techniques will be discussed for producing intelligent agents. For those less interested in algorithmic history, it is recommended to skip to Section 5.1 ALGORITHMIC ANALYSIS SUMMARY.

In the technical paper “Where’s the AI?” [Schank 1991], Schank provides a general criticism and overview of the state of Artificial Intelligence (AI) told by the experiences of the author in dealing with large scale AI systems for academia and industry. The main issues identified pertaining to AI are: the capturing of human intelligence, the scale up factor, and adapting to changes in an environment. Capturing human intelligence is not just recording data from experts but actively displaying that intelligence back to the user in some form. This would be necessary for any form of AI system meant to mimic human intelligence. The “scale up factor” pertains to AI systems that should be able to function in more than toy domains. A system that can appropriately work with only a small number of inputs and limited solution space should be shown to also be feasible for a larger solution space using the
same method. Finally, a system should adapt to changes or differences in the environment to be considered a full AI. A system that cannot adjust to noise or small differences from the training data will not be useful in practice. These are the basic tenets of AI put forth by Schank that will be used as motivation and judge of AI algorithms and techniques.

4.1 Manual and Semi-Automatic Methods

The category for manual and semi-automatic methods has been separated out for individual discussion. These methods require a large degree of human interaction in the coding of an agent versus one being produced automatically from a dataset. Typically, these tend to be older methods; however, because of this fact, they are better understood and used more often in practice. The main problems discussed are the difficulty of producing an intelligent agent without an expert in the given field and, once an agent is produced, verifying that the information that it encodes is correct and consistent.

4.1.1 Classical Methods

Classical methods represent creation technique used in the past to build intelligent agents that cannot learn. These systems can still be classified as Artificial Intelligence depending on the laxness of the definition. In many respects, these methods are some of the first implemented when building an intelligent agent.
At the most basic, although by no means a simple program, is the naive programming technique for producing an intelligent agent. It is the general, straight-forward method of programming an intelligent agent for a particular problem. Since the agent is generally produced for a particular project, it can lack the ability to be reused in another domain. The behaviors of these types of systems are sometimes referred to as “scripted” entities that perform a given list of actions. These systems can be made to imitate intelligent behavior with enough tweaking by the programmer; however, this conflicts with the goal of producing an artificially intelligent agent without all the required time and effort of a programmer. Naive programming itself is not a specific technique and lacks any specific formalism. However, it is a good starting point for a comparison to the desired metrics. It can encode human behavior, but at the direct cost of manually programming behaviors. The system will not scale up without a rewrite, and, without a formalism, the exact method for recoding can be difficult. Furthermore, because the behaviors are scripted, the system will not be able to adapt to any environmental changes. Therefore, naive programming, while popular and easy to implement in small scale problems, does not meet the AI goals put forth.

There are methods that add levels of formalism and complexity to the method of creating agents. Since they are not the focus of this work, they will be also categorized under classic methods. These methods include: state machines, induction systems, fuzzy logic, heuristic search trees, and many others. These systems can also be declared as AI, in that they make a decision based on inputs. However, they typically lack in many of the other desired areas
including difficulty in creation/maintenance and inability to adapt. Several examples using these types of systems will be discussed later.

Much work has been done in the field of state machines in both the analysis of what they can physically compute and formalizing them to be understandable. Each state can encode bits of human reasoning for a given problem in order to perform some task. In a deterministic state machine, the system will start in a predefined state and transition to other defined states based on the inputs through predetermined transitions. A basic state machine forces a set of computations to a specific area of code that is operating mutually exclusive to all other states. Because of this, code specific to that state can remain relatively simple because it does not need to compute information that is not relevant to that state. This property can allow for a scaling of a system, given that each state can remain small and new states and transitions can be added for more complicated tasks. However, this can also be a failing since the transitions increase with the square of the number of states and the ways in which to transition increase exponentially with the number of inputs. If the formality is maintained, there is less of a technical limitation if the computing power exists, but, since each transition is hand-coded, creating and guaranteeing correctness of those links becomes a limitation. The programmer would have to know which state to be based on the inputs, and what are the appropriate states than can be reached from any other state. In [Brooks 1986], Brooks attempts to remedy some of these problems by using layers of control in a hierarchical fashion in a mobile robotic platform. Using different LISP modules, different levels of control with different goals can be applied at the same time. When levels of logic
fail, the system could revert back to a more primitive state. In some fashion, it could be
declared to be able to adapt, but it is only falling back to other hand-made code that also
could fail. The formalism of a state machine gives improvements over naive programming
with respect to some scaling since tracing and adding code can be improved. However, the
same difficulties of manually creating code for each state, ensuring the correctness of being
in a state and when to transition to others is still difficult.

Another formal system that has been classically used in AI is induction systems. Using
a system of logical predicates and inputs, the agent can “reason” certain relationships and
actions when performing a task. The AI system is created by hand by encoding a series
of logic statements. These statements can then be strung together into a tree that can
chain together reasons that are not otherwise directly stated. Starting with an hypothesis,
the inputs can chain through the logical statements attempting to reach a conclusion of
whether the hypothesis is true. A common way to implement this system is with the Prolog
programming language since it is specifically tailored to these types of problems. In (Mitchell
et al. 1986), the authors created a system that “generalizes from examples” in order to encode
human intelligence. A series of logical statements are put forth about a set of items such as
fragility, density, and volume. Then rules are made based on the given examples of items
that are safe to stack on one another. Then a question is asked if some other object can
be stacked. One of the advantages of this system is that it can “explain” why it thought
something could be stacked by tracing the logical tree. This is important in the overall
goal of capturing and displaying human intelligence. However, these rules are not directly
gleaned from examples automatically in any way. The programmer must create and confirm the correctness of each logical statement along with adding new concepts. All reasoning is done with manually programmed rules for each example and are therefore closed world systems than cannot adapt to new information.

A different spin on formal logic is the appropriately titled Fuzzy logic. Fuzzy logic is meant to describe a world that is not black and white. In \cite{Zadeh1975}, the author expresses that “It is a truism that much of human reasoning is approximate rather than precise in nature.” Fuzzy logic contains values on the continuum between 0.0 and 1.0. The membership functions can express things other than true and false such as unlikely, likely, very likely, etc. Using these fuzzy functions, variables can be arranged into fuzzy sets and fuzzy operators can be applied to produce useful logical statements. This ability would pertain to the human expressiveness that is being portrayed by the AI. Fuzzy logic allows for real and continuous data to be presented and processed by a logic system. It has been applied to both control and decision based applications such as \cite{Lee1990}. In that paper, the author is able to capture human intelligence by using linguistics of words used to create a fuzzy membership function. Those functions can then be used in real-world problems where exact values are not known. As a method, it can encode human intelligence in an understandable way. In addition, the resulting answer, after applying fuzzy operators, can also be converted into a human understandable form. The main drawbacks are the creation of the rules and adaptation. Directly there is no defined method to create the rules in a way that would reduce the programming work load, although more recent methods will be
discussed later. The rules themselves are not adaptive to change. However, because of the nature of the membership function, small fluctuations may be absorbed, allowing the system to continue to work.

Finally, a classic method of AI used is heuristic search trees. In (Pearl 1984), the author states “Heuristics stand for strategies using readily accessible information to control problem-solving processes in man and machine.” These strategies, which are selected by the programmer, can define the particular merit of a given input situation or action. In this form of AI, the actions and solution space can be thought of as a tree. This method can be used in forms of game theory and path planning. This method involves producing a set of states that can be reached from a given start state. The goal is to produce a chain of transitions that reach the goals state from the start. The heuristics involve scoring the position and the order of evaluation. This method is generally considered AI and is actively used in many board games and robotic systems. The advantage is that the system could be created with a state generator for applying actions and a heuristic function to judge the value of that state. Its major faults are the lack of adaption and scalability. This type of system typically needs to have deterministic outcomes to model future states in order to make a decision. Each position needs to be enumerated and any stochastic information would not allow for direct links between states. Additionally, the combinatorial basis for the search space increases exponentially eventually limiting the possible analysis based on the number of possible outcomes.
In general, the classical methods are used in current applications that are normally called AI. The main issue presented is that while each method can produce results that, from an outside perspective, seem quite intelligent, they lack some of the requirements to make a truly intelligent agent. While each system can deal with a variety of inputs and situations, they are still created by hand by a human programmer, they cannot scale up without the addition of code, and they do not adapt to changes in the environment. These systems, in a way, can be considered fragile since they are created by hand for a specific application to produce a certain result. A failure in these systems would make them seem like a “computer” by getting stuck in a loop or completing the same action for a given situation. For these reasons, the classical systems would not be a fit for the future of intelligent systems.

4.1.2 Expert and Knowledge-Based Systems

Expert or knowledge based systems are a form of AI that have been actively used for several decades for intelligent systems. Its advantage over some of the previous systems is that there is a formal process of storing and producing results. Additionally, many tools exist that allow the system to not be written from scratch, but instead use an existing shell to enter data. This reduces the work on a programmer, but adds a new job of a knowledge engineer. The main goal of an expert system is to take the knowledge from an expert in the field of the problem domain and encode their responses to a given situation. Therefore, a system following the responses of the expert would itself act as an expert in the field. One of
the main advantages of this type of system is that it skips the computational aspects of the many classical forms of AI and produces “correct” results for a given problem in the same way an expert would. A more apt definition of a knowledge based system:

“A computerized system that uses knowledge about some domain to arrive at a solution to a problem from that domain. This solution is essentially the same as that concluded by a person knowledgeable about the domain of the problem when confronted with the same problem.” (Gonzalez and Dankel 1993)

An expert system meets at least one of the requirements put forth by Schank, namely the capturing of human intelligence. In fact, that is the main tenet of the expert system encoding process. Unfortunately, the process of capturing this information is still human driven and requires the work of not only a programmer, but of an expert as well. In Gaines and Shaw (1993), the authors put forth the basic steps of a Knowledge Based System.

- the knowledge engineer interviews the expert to elicit his or her knowledge
- the knowledge engineer encodes the elicited knowledge for the knowledge base
- the shell uses the knowledge base to make inferences about particular cases specified by clients
- the clients use the shell’s inferences to obtain advice about particular cases

Using this method, an intelligent agent can be produced for a given problem domain. There exist many different systems and shells which have differences in which aspect they concen-
trate, but each requires “a substantial role for the knowledge engineer” (Boose and Gaines 1989). For expert systems, the gaining of knowledge usually involves interviews with the expert. According to Forsythe and Buchanan 1989, “interviewing is a difficult task that requires planning, stage-management, technique, and a lot of self-control.” A tremendous amount of work is involved in eliciting the information from the experts, including dealing with uncooperative experts and having the communication skills to ask the right questions. Additionally, the problem of combining expert knowledge from multiple experts can also pose a struggle since expert may use their own “rules of thumb” in order to examine a situation. It has be stated that “knowledge acquisition is a bottleneck in the construction of expert system” (Hayes-Roth et al. 1983).

Expert systems can be used for a variety of applications including classification and diagnosis. To gain the knowledge required to do these tasks, there has been much research in the field of Automated Knowledge Acquisition. In this field, systems have been produced that ask the domain experts the right questions for building the knowledge base. One such system called AQUINAS was developed for Boeing (Broose and Bradshaw 1999). This system, which had actual use in the company, was used as a method of expertise transfer. It used a shell to enter in a form of questions and answers such that AQUINAS would generate the rules for the underlying expert system. These included implication, solution, absolute, and specialization/generalization rules. These rules can then be used in the prototyping and selection of items by the end user when creating a product.
In another system called ENIGMA defined in (Giordana et al. 1993), the rules are set up in a similar manner; however, it uses the additional approach of model-based reasoning. The first order rules test for the failing of a motor using a set of observed outputs and compares it to an internal model. This is more of a diagnostic approach that uses knowledge of the expert contained in the motor model and compares it to the produced results.

In (Perkins and Austin 1990), the authors attempt to incorporate temporal reasoning into an expert system. While variables based on the situation are important, it can be even more important to see how those values changed over time. In yet another system described in (Gonzalez et al. 2005), the shell was created to generate rules concerning tactical military knowledge in an automated fashion by posing the questions in increasing detail. By using a domain-explicit shell and dividing the system using Context Based Reasoning (CxBR), which will be discussed more in depth later, the system could reduce the work of knowledge acquisition by asking only related questions one context at a time within the domain.

In all of these systems, however, the knowledge acquisition is still manual and requires the explicit organizing of relevant information, without benefit of machine learning techniques. However, some individuals such as (Whitehall et al. 1990) express that this is a good point of expert systems, since explanation-based learning systems can fail on sets of problems without complete, information and empirical learning systems are “easily misguided” by raw data sets. Whitehall states that their system is said to produce more intelligent results; however, they needed to manually encode partial sub-trees and basically give the system the answer
to get the answer. This is contrary to the goal of reducing the work of a human individual when producing an agent.

The major failing in expert systems is the knowledge acquisition to encode the expert into rules. The reasoning method is sound in that it can produce a competent agent that has the characteristics of the expert, but the hand coding of the rules is time consuming. Moreover, in the process of encoding the rules, the artificial intelligence is taken away and replaced with a system that “knows” the answer within a small domain without generalizing to a larger domain. The scale up requirement begins to fail as the number of man hours to encode those rules and gain the knowledge from an expert increases. The system can become infeasible in large scale. Additionally, the system does not learn or adapt to trends.

### 4.2 Automatic Methods

There have been many different ways conceived to reduce the human workload in order to create intelligent agents. Each of these systems can take a set of inputs and outputs presented to it and, in some fashion, automatically create a mapping from one to the other. While the basic goal is the same, the actual method of implementation and the paradigms used vary drastically. Some methods are purely based in data and apply matching or clustering functions directly to the current input. Others develop models or functions to approximate the data and decisions based on known correct data. Still others do not require decisions or answers at all and purely make an agent based on a rating metric. Each of these various
approaches has advantages and flaws particular to an application or dataset. Pure manual creation requires explicit knowledge about the environment and typically known correct answers to be encoded. These automatic methods, however, can reduce the manual effort used for intelligent agent creation by allowing an algorithm to do much of the work.

4.2.1 Data-Based Algorithmic Approaches

There are many algorithmic approaches to creating an intelligent agent from a data set. In fact, there are too many to even list and cite accurately. This section is composed of a few methods that do not fit directly into this dissertation but are technically interesting and are adapted further in later sections. However, each algorithm has a common theme in that they require a stream of input and output pairs in order to produce a result.

When a programmer is presented with an input of known data and a set of known output data, there are several direct approaches that are can be applied which would be used in tools to produce agents. If the data is purely numerical, a Least Squared Method (LSM) may be applied to produce coefficients to model the inputs versus outputs. If, on the other hand, the data is to be just modeled, then statistical methods could be applied in order to find a mean and standard deviation of the data for a normal fit (Neter et al. 1996). Alternatively, if the data is categorical, a k-Nearest Neighbor (kNN) might be applied to find past input matches in the data set (Dasarathy 1990). These are approaches that can produce a model from a dataset of known inputs and outputs. That model then can be put into an agent to
produce a system that generates outputs from a new set of inputs. These can be considered AI; however, many will discount these methods as just math and statistics. In the goals put forth for AI, these types of algorithms can be shown to model a human and can basically scale up with additional data. However, these are typically a preprocessing step and is not usually said to learn or adapt. However, when presented additional data, they could be retrained and produce a new system.

One of the more basic models previously discussed was the LSM. In basic terms, the input-to-output relationship is thought to be a Linear and Time Invariant (LTI) system. Therefore, there should be a set of coefficients that multiply with the input that can produce the output. There is a direct means of pseudo-inverse that can be taken to minimize the error to produce those coefficients. This only works by pretending the system is linear or by modifying the inputs to produce a larger pseudo-linear space. This is done in ([Wickens and Responses 2004]), to produce a complex linear model of a set of inputs. The author argues that linear systems are understood better through many years of research and can be easier to adjust. The system creates all combination of possible linear systems in a hierarchy and uses a set of probabilities as it goes down the tree. The system uses real data in an attempt to create the best linear system. A reason this could be considered AI is that it might select a model that is unexpected even to a trained programmer or engineer with knowledge of the data. This independent selection would be the form of Artificial Intelligence in some sense. However, the system fails some aspects of scalability. The number of combinatorial input combinations expand at a factorial rate and, without a method for searching the space
better, would prove practically infeasible. Additionally, if no adaptation is done, the system would have to be re-run with new data with any environmental change.

When a system needs to be created for a time variant or not fully-known environment, it is possible that one would want to model the probability of a certain event. This aspect is presented in the seminal work “A Mathematical Theory of Evidence” (Shafer 1976). The work is rooted in the basic belief that the probability of an event is scaled between 0 to 1 using the Shafer, later Dempster-Shafer, method. This is a statistical means of predicting a given state given a set of evidence. This could be used to perform an action based on the known state. Another statistical method typically used is Naive Bayes. Naive Bayes has been used in computer terms for over 50 years for the purposes of information retrieval and classification (Lewis 1998) although its actual creation was in 1763 by Thomas Bayes (Churchman 1946). The main theory is that probability for some output for a given input can be calculated knowing the probability of the output, the probability of the input, and the probability for the input given that same output. For the purposes of agent creation, the goal is typically to determine the correct action given some input data from the environment. It would be required to know the overall frequency that that action occurs, the frequency of that input over all inputs, and the probability that that input would occur assuming that the given action is correct. All this information can be collected from data recorded in the input and output pairs. The naive part of Naive Bayes is the assumption that each of the input variables is independent from each other. Although it is stated: “the independence assumptions on which Naive Bayes classifiers are based almost never hold for natural data sets” (Lewis 1998).
These types of classifiers still work reasonably well and are used in numerous applications. A Bayes classifier has multiple good points, including a basis in probability that can be directly understood and all the information can be directly gained from recorded human information. It can also be the problem because the system is purely based in the given information. Assumptions of the dataset representing the actual probabilities of the testing environment can be faulty. Adjustments to new environmental changes can be made through human interaction to the known probabilities or additional corrected data must be presented to the system. Additionally, the system typically would need discrete actions or output classifications in order to work properly, although that topic has been researched (Frank et al. 2000).

Another method of statistical analysis done on a data set uses rank statistics and is popularized in the decision trees (DT). A decision tree is used to represent decisions at many levels. It begins at the root where a decision is made on what branch to follow based on some input information. Once that branch is selected, the next decision is made until eventually a leaf node is reached. The leaf node is typically some form of classification. While they existed previously, an automatic way of creating these trees was developed by Quinlan in (Quinlan 1986) aptly titled “Induction of decision trees” which introduced ID3, later followed by C4.5 and M5. Quinlan states that these systems were explicitly created to reduce the amount of human time producing rules from “protracted interaction between a domain specialist and a knowledge engineer.” The system builds a tree on every level by trying to maximize the “information gain” at every level. This is done using Shannon’s
theory of information and entropy. This method partitions the data at each branch until a leaf either contains no differences in output classification or differences less than a certain probability. The decision tree is a good method of encoding human intelligence because it can be easily read and parsed. Furthermore, it is also possible to transfer that knowledge into production rules (Quinlan 1987) which may be even more human expressible. Decision trees are very good for creating static classification, but are known to be bad on noise and changing environments. Additionally, when new information is added, the whole of data must be re-processed at once to map out the information gain.

These algorithms have been known to work using data sets to produce intelligent agents. They reduce the workload of a programmer by automatically combining together information in some way from a given list of known input and output pairs. Each has individual flaws that are specific to the algorithm, but the main flaw of purely data-driven algorithms is the assumed correctness of the data and the requirement to have labeled data for presented inputs. Any mistakes made during the agent’s execution would be difficult to correct directly without taking the inputs for the incorrect action and relabeling them for the correct action. Even then, the system would have to be re-trained from scratch in the hope it would be fixed for that particular problem. The requirement to have labeled data at each time instance can create additional work for a human. Additionally, only data that has been prerecorded and labeled can be used. This would eliminate almost any form of adaptation online through using these direct methods.
Another much-lauded automatic method for performing the automated agent generation system is the Artificial Neural Network (ANN). Its popularity as a method of biologically inspired computation began in some of early work of McCulloch and Pitts (McCulloch and Pitts 1943) when they took the idea of “the nervous system is a net of neurons” and transferred it into a mathematical function. In their paper, the authors take the biologically-tested idea of stimulus and impulses traveling to each of the connected neurons for a method of connected calculus to explain human reactions and feelings. In (Hebb 1949), Hebb later attempts to “bridge the gap between neurophysiology and psychology” by theorizing how the conceptual nervous system would induce positive and negative stimulation through an “arousal function.” These ideas were then later augmented into a true computational model by (Rosenblatt 1958) for the creation of the perceptron. Rosenblatt takes the ideas of the connection of the optic nerves and the retina to further the feedforward network of connections as the structure of an ANN. However, more importantly, a method of automatically trying to “associate specific responses to specific stimuli” (Rosenblatt 1958) created a way to train this network. This was done by biasing a network of binary outputs to affect the probability of choosing the best response. This showed a how to train a network but the system used a binary model of excitation, and a single layer of connected components was unable to compute non-linearly separable information (Minsky and Papert 1969). This work put a major damper on the research in the field; however, models were adjusted such as
Hopfield (1982) who replaced the binary model with a linear one along with a weight factor with the summing term and that was changed with a delta function using the Hebbian learning curve. The Hopfield net consisted of simple asynchronous computational units with “some capacity for generalization.” Later in (Sanger 1989), the author showed that Hebbian learning methods were also applicable to unsupervised learning, where the network could be used to encode the presented inputs without explicit need of “correct” information.

However, it was in (Rumelhart et al. 1986) that the authors found a method for structuring and training that proved that ANN was still a powerful method. The method involved a multi-layered perceptron (MLP) for computation, where a minimum of another layer was inserted between the input and output layers. This hidden layer allowed the system to recognize the XOR problem that plagued the single layer perceptron and was shown to be able to “form arbitrarily complex decision regions” (Lippmann 1987). Additionally, each neuron was given a sigmoid squashing function for the summation of the inputs. This improved upon the linear model and was more similar to the original binary model of excitation without “the hard limiting.” The method developed was the Backpropagation algorithm that used a gradient descent search techniques that attempts to minimize the least mean squared error (LMS). This is done by modifying the weights backwards from the outputs using the derivative of the activation function and the LMS error. This method is then applied backwards along all of the layers to the lowest level. The sigmoid function is useful in this regard because the derivative of the function is simple to compute. The gradient descent method is applied to the weights of the network as the input data are presented again and again.
for each iteration until a minimum in the error space is found. Backpropagation was found to be a powerful tool in both classification models and non-linear function approximation. Backpropagation has become a standard among ANNs; however, much of the research for the next few decades has been to analyze its properties and address some of the shortcomings.

In an early work (Shavlik et al. 1991), the authors attempted to compare the differences in the symbolic and neural learning methods on supervised learning tasks by using ID3 and Backpropagation on a series of test datasets. The authors found that while the methods are dissimilar, the “two basic approaches to machine learning ... frequently address the same general problem.” The general consensus of the authors was that backpropagation outperformed ID3 on some trials while on others “the differences were not statistically significant.” Additionally, they found that backpropagation took significantly more time to train. Other findings included that backpropagation was less susceptible to noise; however, both systems degraded similarly in the presence of missing features. Therefore, on least these sampling of datasets, backpropagation was shown to be at least as good as an ID3 decision tree.

Further analysis of the system was done in order to judge the properties of backpropagation. In (Lee et al. 1991), the effect of the initial weights of the MLP was looked at for effects on the convergence. It was shown that small initial weights can avoid premature saturation of the sigmoid function. This would cause slow convergence and longer training times, since the derivative would be flat in those extreme regions. Additionally, backpropagation does generally guarantee a convergence based on the gradient descent method to an error minimum. However, it is not guaranteed to be the global minimum and therefore the “right”
answer. In [Krogh and Hertz 1992], the authors improved generalization by introducing some factors of weight decay. This allowed for better convergence in linear and non-linear systems when learning models. The reasoning is that “weight decay can suppress some of the effects of static noise on the targets.”

Different methods exist for affecting the speed of convergence for backpropagation by changing the weights in different ways in order to improve performance. However, there is another way to drastically affect how a MLP works. The structure of the network and the number of hidden nodes change how the network operates. In a standard backpropagation ANN, the number of hidden layers and nodes per layer is fixed. However, it is unknown what the best number is because it is very problem specific and is sometimes discovered through simple trial and error. This increases the amount of programmer work and decreases the autonomy of the system. In [Khaw et al. 1995], the authors developed a method to compute this number of nodes using some signal theory. They attempt to maximize the signal-to-noise ratio based on the number of inputs, outputs, and training size to create an upper and lower bound. Then adding some information on accuracy through experimentation, an optimal feedforward MLP can be created. Others, such as in [Fahlman and Lebiere 1990], take a more systematic approach by incrementally adding hidden units one at a time after a network is minimized. A series of candidate new networks are produced “when no appreciable error reduction occurs.” Each of the new candidates is evaluated independently and the one that improves the most is selected to repeat the process. This greedy method that they call Cascade neural networks is able to outperform a fixed architecture.
Finally, backpropagation is not the only method for training MLPs. It has been seen, that backpropagation can be slow and converge to local minima. Other methods such as Quickprop (Fahlman 1988) attempt to improve convergence by increasing the change in weights as much as possible and still be stable by using the second derivative along with some other heuristic functions. Based on experiments, the method was generally as good and took significantly less iterations to converge. A more exotic method proposed in (Puskorius et al. 1991) called Node-Decoupled Extended Kalman Filter (NDEKF) method treats the weights of the feedforward MLP “as a parameter identification problem for a nonlinear dynamic system.” The system takes the error difference as noise for an extended Kalman filter and attempts to adapt the weights in a fixed network by treating them as states in the filter. The system is able to outperform gradient descent in terms of number of iterations and improve in accuracy. The concept of borrowing from other fields of computation and coupling them to the structure of an ANN is very interesting and is shown to be very common in the later section on Hybrid systems (Section 4.2.5).

Another point of analysis not discussed yet is the effect of time with respect to the ANN. A feedforward MLP by itself has no direct linkage to the aspect of time. A set of values is presented, the inputs are multiplied by weights, and those values are sent to the connected neurons until the outputs are calculated. The system by itself does not have memory between reactions during the execution stage, although it could be argued that the order of presentation could matter for the training stage. This makes the system reactive in nature that each individual input is treated separately. This is fine for independent
input/output presentations, but what if the system is meant to model a system over time such as with a chain of human reactions. As stated in (Elman 1990), “There are many human behaviors which unfold over time. It would be folly to try to understand those behaviors without taking into account their temporal nature.” Different methods for modeling this type of problem have been researched. In (Elsner 1992), the author presents a challenge to model a time series non-linear function. The most direct approach is used which is to create more inputs from the existing inputs, each of which is a discrete time step back. In this way a short history of inputs can be used to compute the outputs called a time-delay ANN. This can be thought of like a non-linear Finite Impulse Response (FIR) system whose inputs only affect the outputs at the maximum of the delay time.

A similar method is used in (Werbos 1990) where the author uses the old inputs and the old outputs when running the system. The results of this system introduce several new concepts. The system added recurrent links in which past outputs now affect future outputs. This can be considered a non-linear Infinite Impulse Response (IIR) system that even a single impulse can create an infinite oscillation. This also means that the system can become less predictable because a single change in the past affects all future values. However with this complexity, a highly advantageous effect happens in that the network now has a memory. Recurrent links literally have a link to the past information that can be used for creating more complex actions. Actions include periodic functions, numerical integration, and possible planning for future events. At this point, all recurrent links happen on the macro-level from outputs of the time-step before to the inputs of the current time-
step. Micro-level recurrent links can also happen between nodes and even self-looping nodes. Weights on these type of connections are much more difficult if not impossible to fix using standard backpropagation; however, there has been work in the field ([Pineda 1987]) to allow this to happen in certain cases. Other training methods are not directly limited by recurrent links, which will be discussed more in future hybrid sections (Section 4.2.5). Another aspect with respect to time is time varying systems. In ([Heskes and Kappen 1991]), the author did a study on ANNs in a dynamic environment. The author found that backpropagation ANNs can fail to generalize if the data is not fixed. The author finds that “in a changing environment there is a trade-off between adaptability and accuracy.” The networks can become “over-fit” after a number of iterations, and if the environment changes, the network does not always transition properly.

MLPs are not the only form of ANN. There are many other competing systems invented over the years that are considered ANNs through either structure or historical purposes for the operational method. In ([Carpenter et al. 1992]), the authors introduce a method call FuzzyARTMAP which is a type of adaptive resonance theory (ART) neural network with match tracking that uses the fuzzy-min operator. Unlike the Fuzzy Min-Max Classifier (FMMC) in ([Simpson 1991]), which encodes the weights of the ANN as the fuzzy-min or fuzzy-max of the values presented to it for each classification, ARTMAP can have an arbitrary number of output nodes that are created based on a vigilance parameter for each classification. Additionally, the ability to create or attach to the correct node for an input during the match tracking stage has proven to “learn any consistent training set to 100%
accuracy” (Carpenter et al. 1992). This system has shown tremendous interest for its ability to do on-line learning. A current popular system introduced in (Cortes and Vapnik 1995) presented a method now commonly called Support Vector Machines (SVM). It is a learning system that can do “two-group classification problems” by drawing a hyperplane decision boundary to separate the two groups. While at first the system would seem to fail the same classic problem of linear separability that broke the perceptron, the trick is that the input data is passed through a kernel function that converts it into a much higher dimensional space. One item liked by researchers is that there is only one optimal hyperplane separator in the high-order space. This means that an answer can be derived eventually, regardless of data presentation. Several obvious modifications have been made to the system to support one-vs.-many and many-vs.-many voting schemes to account for standard binary nature of the outputs. Another network type introduced in (Specht 1988) is the Probabilistic Neural Network (PNN). Its main contribution is replacing the sigmoid function with a Gaussian one. While the structure is similar to the MLP, PNNs can create “decision surfaces which approach the Bayes optimal under certain conditions.” This could produce the best classifier and is heavily based in standard probability theory. The selectable parameters for smoothing, computing probability based on data, and ability to separate out the system to be processed in multiple distinct sets which are combined together later, has made this system popular. Finally, a new system called Compositional Pattern Producing Networks (CPPN) introduced in (Stanley 2007) is a form of ANN that separates itself by using an arbitrary function in each of the different nodes rather than homogeneously a sigmoid or Gaussian
function. The structure is similar to an MLP with recurrent links. However, because of
the nature of both the ability to have recurrent links and the various internal functions, it
cannot use backpropagation in order to calculate the weights between nodes, but instead
uses another system that is mentioned below in the hybrid section.

After all of this discussion on various aspects of ANN, the topic must be tied back to
the main point of creating intelligent agents. Neural Networks in general are considered an
automated form of machine learning algorithm that can take in data during the training
phase to produce a model that can be used later. This model, in the case of an MLP, is
known to be able to fit any non-linear function or classify an arbitrary space. Relating to
the number of links, it can scale with respect to the problem domain. With recurrent links,
it is known to have memory and produce time evolving functions. The structure itself is
well defined and based on the biology of a human. Harking back to the precepts required of
AI, the system can capture human intelligence and display it back. The only caveat is that
the underlying structure and weights produced are not directly understandable in human
readable form. The system can scale up to arbitrarily long problems because it produces the
agents automatically from input data. However, the computation time for training could
be long for large inputs sets and without an automated way of determining structure for
large problems more human interaction would be required. ANNs have also been shown to
adapt to changes, but they require error per input. The last item is the only real failing of
using the ANN method by itself. With a few exceptions, the standard ANN is a supervised
learning method that requires a sample set in order to train. This would mean that not all
learning paradigms discussed later would be able to train an ANN directly. However, using a hybrid method, the computational structure that is an ANN can be adapted to emulate a human intelligence automatically.

4.2.3 Genetic Algorithms and Genetic Programming

Another common method in the field of AI is evolutionary algorithms, in the form of genetic algorithms and genetic programming. They are another biologically-inspired process modeled on the evolutionary characteristics of living things that was originally introduced in [Holland 1975]. This method is meant to be an improvement on search techniques in a problem space that should be better than random search, hill-climbing, and simulated annealing, although it has been shown that this cannot always be true ([Wolpert and Macready 1995]).

As a basic generalization, a GA usually contains a population of individuals, each with a certain encoding of their make-up similar to DNA. Each of the individuals is then tested in a process and graded with a fitness function that will give a relative measure of how “fit” each individual is. Those individuals are then selected and typically paired up with some factor probabilistically weighted toward fit individuals combining together for mating. When two individuals are mated, the genes from each individual are combined together through crossover to create a new individual with some additional probability of having a random mutation in the genes. The new individual is then inserted back into the population for later evaluation. This process is repeated and can either happen in a generational model where all
the parents are killed off and replaced by all the children, or in a steady-state model where one individual are replaced one at a time in an incremental way. Modeled after evolution and the survival of the “most fit”, the theory is that fit individuals combining with other fit individuals will produce even more fit individuals down the line by the combination of the best genes from each parent. However, there is a delicate balance between exploitation and exploration. If only fit individuals are combined together exploiting a feature, the system might prematurely converge and, without additional variance in the gene pool, will begin to stagnate. If random mating occurs and the genes change drastically, thus exploring the solution space, a convergence on a specific solution may not happen and key features from parents may not meet. These effects are most related to the selection process which uses different common methods such as fitness selection, rank selection, and tournament selection. Additionally, crossover can take different methods for combination of genes from the parents such as single/multi-point crossover, uniform crossover, or some method specific to the representation. The main semantic difference between a GA and a GP is what is created by the representation. A GA might encode just the parameters for a simulator, while a GP will typically be able to generate code that can be compiled or a script that can be interpreted directly.
4.2.3.1 Genetic Algorithms (GA)

GAs are popular when the number of combinatorial possibilities of a problem space is large and an intelligent means of searching them is needed. The structure of a GA can vary and can usually be decoupled from the problem domain being analyzed. When the GA was first introduced in \cite{Holland1975}, it used a binary string representation which is now typically called the Simple GA (SGA). In a SGA, each gene is represented by a bit as part of the entire genome and it was up to the application specific information to decide how to use that information and develop the fitness function. For certain test problems such as the OneMax, the bits are used directly and the fitness function is based on the number of ones in the genome. In a very old example in \cite{Grefenstette1986}, the author chose to have a GA discover the parameters for numerical optimization problems. The binary genome encodes numbers that are used for function approximation. In \cite{DahalEtAl2001}, the authors use the GA binary string to encode an integer directly and used a complex simulator of results to discover the optimal solution. In \cite{JaneczkoLopes2000}, the authors used a GA to encode the coefficients for a discrete time filter by having the binary string encode a fixed point number between -1 and 1. In \cite{HongEtAl2000}, the genome translated into a series of branches in a game theory tree. This was done to bypass the evaluation of the exponentially growing size to evaluate each and every leaf of a move tree. In each example, the structure of the GA was turned into a format specific for the problem formulated by the author. GAs are very flexible in their encodings and it is up to the programmer to decide
what information needs to be optimized and how the genome can be used to represent that information. After that point, all that is needed is a fitness function to determine how well the genome performed and the process begins. However, it requires a degree of work for the programmer, because the GA does not inherently have a structure of its own for processing problems.

GAs are also affected by the fitness function and the surface of the solution space created by it. Because GAs are typically attempting to move toward a solution, a fitness function that has many distinguishable values and where the best solution is a combination of two good solutions is desirable. However, GAs can solve problems where this is specifically not the case, such as the deceptive problem (Goldberg 1987). Another good feature of GAs is the ability to work on moving fitness functions. In (Rand et al. 2006), the authors showed that crossover had a significant effect on moving fitness. It was found that when some external factor “changes the landscape” GAs still work because “the new elementary schemata that need to be recombined are already present in the population but are located in different individuals.” The population had to retain genetic diversity in order to be robust and adaptable. The individuals that did well in one environment did not necessarily do well in the changed environment, but as long as the entire population has not converged, genes from individuals that excelled in other areas could be brought in to create better children.

The GA itself also contains parameters to be adjusted to tune the system for a specific problem. In (Feldt and Nordin 2000), the authors did a sensitivity analysis of the 16 different parameters that could be adjusted. Over the many runs for the different combination of
changes, they found that the main components of importance when adjusting were population size, number of generations, and mutation rate. Larger population size and number of generations increased performance of the GA; however, both of these factors also increase the amount of computation time. Luckily, since in normal operation each individual in a GA is evaluated separately, the problem becomes parallelized quite easily. Thus separating out the population over a cluster of network computers can increase the computation at an almost linear rate. Using a cluster, larger populations can be evaluated for additional generalizations in the same amount of time.

There are other interesting concepts that have been added to GAs to increase their similarity to the biological process after which they are modeled and increase their performance. In [Levenick 1991], the author took a cue from biology and introduced the idea of using introns in the gene sequence. For purposes of computer science, an intron is a non-encoding sequence that exists within DNA that seemingly serves no purpose, but is also inherited from parents. However, the author found that the “insertion of introns was demonstrated to produce as much as a ten-fold increase in successful evolution.” This is attributed to allowing crossover to split genomes in places that do not split the usable genes. In [Goldberg et al. 1989], the authors used a variable length structure they titled a “messy GA.” In their form, the GA is encoded in a ring which contains an index and a value. In this way, the specific order of the genes no longer matters. The string can shrink and grow in size to accommodate the problem. This was done because “nature has formed its genotypes by progressing from simple to more complex life forms.” This also reduces workload because it is no longer re-
quired by the programmer to set up the string arrangement in a specific form. In (Miller and Shaw 1996), the authors take the idea of dynamic niches to try to reduce the local optima problem. The idea behind a niche is to avoiding crowding by keeping other individuals out of one’s “space.” If too many individuals fall into the same niche, the fitness of all of those decrease. This method keeps the individuals from premature convergence. This is also done in (Deb and Goldberg 1989), the authors associate this model with the formation of species as in biology. Like individuals are associated with a species and if too many of that species exist, then the species as a whole suffers. Each of these biological inspired processes seems to augment the standard GA.

In summation for GAs, they are a powerful tool that can be used in almost any problem that requires a search and has some means to formulate a fitness function. One of the main issues discussed is that they do not specifically have a structure of their own, but instead must be formulated in a way specific to the problem. However, once that structure is found, the problem can be evaluated. Many of the ideas and concepts behind a GA are biologically inspired and, as a whole, the more similar it is modeled to the natural world the better the system can perform. GAs can be used to create an intelligent agent automatically. All that is required is for the programmer to identify a structure specific to the problem and translate the GA’s genome into that space. The system can model human intelligence by creating an individual that is most fit when the actions of the GA system match those of the human. One issue would be that the gene itself can be completely non-comprehensible, but the translation to the domain specific structure may be. The GA can scale up to any number
of parameters and is known to work in very high dimensional space. GAs are adaptable to changes in the environment as long as the population has not converged too much and features such as niching and speciation help with this process. The main issues are the creation of the structure and the time it takes to evaluate it, since it can require a large population and many generations to find an answer. The latter, however, is not additional work of a programmer but rather, merely computation time.

4.2.3.2 Genetic Programming (GP)

Genetic programs are a specific form of GA that have a specific composition that the structure ends in some form of computer program. In \cite{Koza1989}, Koza introduced a specific method for automatically creating LISP programs using trees. LISP is convenient because it allowed for self-modifying code and already existed as a tree because it is a functional language, but the concepts developed can be used in any programming language. Koza used the standard methodology of the GA but, since the representation was a tree, crossover became the swapping of sub-trees between individuals. Because of the storage mechanism, every sub-tree also evaluated to a value, therefore, every place in the tree can be swapped. Using this method, the author was able to have the system write programs for symbolic function approximation and Boolean logic. The problem was formulated by having the correct responses to several problems and having the system create functions until it matched the outputs perfectly. A main advantage of GP is that it gives the GA a structure and
creates code that, to some degree, is human readable. However, one of the problems is that the collection of elements can be specific to the problem. The programmer must define a terminal set, which would include a list of variables that can be used as input and the constants available to the system such as floating point numbers or Booleans. Additionally, the programmer must define a function set that is composed of mathematical functions such as trigonometric, logarithmic, and arithmetic. The elements available affect what can be encoded and the number of combinations possible both of which may be distractions to the system.

Genetic programming is not limited to generating source code. In (Koza et al. 2000), Koza “programs” electrical circuits by defining a set of functions that relate to connecting a circuit and discovering the values of the discrete components. In (Bojarczuk et al. 2000), the functions are defined as if-statements to create a simple rule set for classification problems. However, function approximation and symbolic expression evaluation such as (Eggermont and van Hemert 2001) seem to be the main thrust of the algorithm. However, they have been used in agent applications such as RoboCup in (Hsu and Gustafson 2001). The tree structure gives a dynamic size to the complexity of the system. But one problem with that is that the tree can grow to extreme sizes and create branches of code that actually do nothing. The paper (Garcia et al. 2003) shows that the later part may not be a problem for the GP because the “Junk-Code” equates to the introns of a GA. However, since the introns can slow evaluation time or cause the system to run out of memory, the known introns are biased for selection and can artificially “grow and shrink in the learning process” (Garcia et al. 2003).
GPs have almost all of the same abilities and advantages as GAs for the purpose of intelligent agent creation. They do gain the additional advantage of having an existing structure in order to operate. They do however have a disadvantage in potential code bloat and having to define a function set that may not be able to compute all known functions.

With both GPs and GAs, the methods can produce agents from some fitness function. This fitness function can be a comparison to known data, or simply a fitness of how the created system works in a simulation. A possible problem introduced is that neither method explicitly has a way to adapt online. In both methods it can be seen that an agent is created from the genome and then evaluated. The forms of change come at the crossover and mutation level for a new individual in the next generation, but no adaptation is done while the system is functioning. This could be because the system is so decoupled from the way in which the system is evaluated. Another possible issue is that the way in which the fitness function is evaluated at the end of a simulation gives the advantage of one fitness over another, but does not have any specifics about individual instances encountered. Depending on the fitness function, an individual that is excellent for a short period and bad during the rest of the time could gain the same fitness as an individual that is simply average during the entire time. Finally, these systems lack any form of recurrent links or memory, unless it is specifically introduced as part of the simulation model. This has to be introduced externally, such as the time-delay inputs or passing of the past outputs as inputs outside of the GA similar to an ANN.
4.2.4 Reinforcement Learning/ACO/PSO

Other techniques in AI used for automatically creating artificial intelligent systems are Reinforcement Learning (RL), Ant Colony Optimization (ACO), and Particle Swarm Optimization (PSO). Each of these methods offers some way to take information from a simulation or data series to produce an agent. Each of these methods is to some degree also biologically inspired based on observed traits of animals. Reinforcement learning is more direct animal-like application in that when rewarded for an action, that action will be repeated more often, and when punished, that action will be repeated less. In ACO, it is the physical observation of how ants work together in a way that each ant is simple but as a colony the combined behavior is very complex. In PSO, the particles are treated like social groups with attractors, and the combination of individual agents can produce complex emergent behavior.

4.2.4.1 Reinforcement Learning

Reinforcement learning as an algorithmic method is different than learning by reinforcement. It is stated in [Kaelbling et al. 1996] that reinforcement learning “has a strong family resemblance to eponymous work in psychology, but differs considerably in the details and in the use of the word reinforcement.” Instead, reinforcement learning “is a way of programming agents by reward and punishment without needing to specify how the task is to be achieved.” There are several forms of RL each has its own slight twist on how it learns, but the main
point is to build a form of optimal policy for a given current state \( a \) that will maximize the reward. A policy consists of a given action for a every given state. Each method can also start with basically any initial policy and should converge to the optimal under certain conditions. Additionally, each method is thought of as a form of dynamic programming in which the cost of old states are stored in a table rather than fully computing for each time a state is reached. In TD Learning (Sutton 1988) the value of reinforcement is updated based on the old state, the current state, and the instantaneous reward. In this system the value function is learned explicitly through “temporally successive predictions.” There are two separate systems: a critic that discovers the value for a state, and a reinforcement learner that takes an action based on the value of a future state with an exponential discounting rate of Lambda. This method works in fixed supervised domains, but has also been shown to converge in stochastic domains (Dayan and Sejnowski 1994).

Another popular method is Q-learning (Watkins and Dayan 1992). In this method, a Q-table is stored for a state/action pair whose action is always the max reward for a given state. It uses a recursive form of update which requires “trying all actions in all states repeatedly” in order to guarantee convergence. Finally, both methods require a discrete set of states and actions in order to operate. The dynamic programming table can become large purely based on the number of states and actions; however, there have been methods used for function approximation to compress this information, including ANNs (Tesauro 1995).

In (Moriarity et al. 1999), the authors compare reinforcement learning to genetic algorithms and contrast the differences including: policy representation, credit assignment, and
memory. In policy representation the GA might store the action to take given a state, while the RL would store a potential reward for a state/action pair and select the best action. For credit assignment, reinforcement leaning scores “reflect the quality of a sequence of decision rather than each individual decision.” GAs also generally do not account for any “bad decisions” in memory because it is implicitly assumed that those individuals would die out, while RL explicitly accumulates both good and bad scores. Reinforcement leaning is based on receiving scores while interacting in an environment. However, since it is shown that an optimal policy can be gained from an arbitrary one, it is also possible to start the system using prior information of known experienced costs. In this way, the system would not have to discover some values in the table but know them ahead of time.

RLs are an interesting technique for producing agents. It can mimic a set of human actions based on the current state through a reward function. This can be trained based on recorded actions, or even better, while running the simulation. The system can therefore adapt in real-time, including stochastic environments. Since the system is discrete, several divisions need to be made from continuous data with a set limit on accuracy. This number is then multiplied by the divisions for the other inputs and then further multiplied for the number of discrete actions available to calculate the total number of states/action pairs. This number can get out of the bounds for usable storage. This could be a scaling flaw except there are alternative means of estimating the lookup table. RL has the main advantage of training based on when good or bad events happen instead of a final fitness score or need
for a correct answer in an input/output pair. But has some drawbacks on enumeration of states and discrete actions.

### 4.2.4.2 Ant Colony Optimization (ACO)

Any Colony Optimization will be covered briefly, as it is another form of AI with useful applications. In ACO (Dorigo et al. 1996), the system is modeled after the method ants use when looking for food. A number of ant scouts basically wander around in an environment, crawling around obstacles, and into different places. When some ant finds food, it attempts to make its way back to the ant pile while laying down a pheromone trail from where it found food. Other ant happening on this trail will tend to follow it with some increased probability. It will follow the path or possibly leave it, and it is possible it will find a shorter path and leave its own trail. If a third ant comes by, it will follow the trail with a greater degree where two paths are down versus one and so forth. The trails also can vanish over time. In a combination of many of these ants following very basic rules, eventually it is shown that the ants will eventually discover the optimal path between the ant pile and the food. In the ACO method, the ants are replaced with many simple agents that have some list of numbers as a path and will attempt to get rewards of some fitness value. Other ants will compare their numbers to a similar ant and decide whether or not to take part of their path or with some probability choose another one. This interaction between different agents attempting to get the max reward creates a complex overall entity that can solve very complex problems.
The system was tried on the Traveling Salesman Problem and the Job Scheduling problem in which the system was able to get positive results. The authors also state that the system can be applied to “any combinatorial optimization problem.” With an increased number of “ant agents,” however, there is a point when the communication complexity between ants reaches a peak. In (Meuleau and Dorigo 2002), the authors found that the system is really a self-organizing method and has many similarities to reinforcement learning where pheromones relate to policy values but also it combines a method of Stochastic Gradient Descent (SGD) which makes the system be similar in some ways to ANNs. ACOs can be used to make intelligent agents, although it would need to have goals, fitness, and a way to map the problem into a combinatorial one. It is another method that requires discrete states of being.

4.2.4.3 Particle Swarm Optimization (PSO)

Another similar method is Particle Swarm Optimization (PSO) introduced in (Kennedy and Eberhart 1995). PSOs have very similar approach to the other methods in this section in that they use multiple simple agents that each have very simple goals, but they are dissimilar in that only the current fitness is needed and not a table of reinforcement values or a pheromone trail. Additionally, the PSOs are stored as a list of floating point numbers and specialize in real valued continuous problems. These values are stored as a state and they record a fitness function similar to a GA. At the very basically level, each of the particles has a small
amount of memory which they store the fitness and state of the best that they have ever been. The state of floating point numbers is treated like a location in N-dimensional space. Each individual is originally randomly placed in space and a fitness value is calculated based on the state. In normal operations, the rule is that they want to go toward the best they have ever been but they also want to go toward the best in the group. The particle produces a vector to each of these locations and those vectors are multiplied by a weight set of how much to be like the best or like they were. Those vectors are then added together in all dimensions, and finally create a resultant vector in a direction. Then each element of this vector is multiplied by a random number between 0 and 1. The final vector then represents a velocity vector. This is where the system gets its name, because a particle simulation is done from the current position using that velocity term to update its next position in space. The fitness of each individual is recalculated and the process begins again. This is a non-linear stochastic optimization process that can operate in real-time; avoid local minima by the use of momentum terms and multiple individuals; and avoids stagnation by having randomized vectors. Additional features such as forgetting factors exist to discount old states in the presence of changing environments.

Like GAs, it is an optimization problem that itself does not have a structure. It is always real valued, but those values can represent parameters for a simulation or coefficients for a classifier. It is up to the programmer to map the solution method into the domain of the problem. It is considered a “social optimizer” that uses slightly greedy steps to increase each individual, but as a group the combination works quite well. In [Angeline 1998], the author
contrasts the PSO with a GA as differences in philosophy. A GA does not have explicit memory of past performance but implicit information relative to individuals in the current population. The GA typically creates a new individual from two different individuals while a PSO just modifies itself based on two individuals: its past self and the best individual. A GA will usually wipe the population while a PSO persists through time.

For creating agents, a PSO has no direct structure but with formatting for the domain a PSO can work well with real world and real valued data. The math is very simple and does not take much time; however, the number of individuals and the effective time step of the simulation could affect performance. Otherwise, the system scales well with inputs and does not even require a sorted list but simply the best individual. Finally, the system works excellent with adaptation because it is constantly changing and moving with relative fitness that can adjust for environmental changes. The main problem, like GAs, is the lack of a formal structure to use the numbers generated.

4.2.5 Hybrid Algorithms

To overcome the limitations of a singular method, hybrid techniques are defined to combine advantages of multiple learning algorithms. This approach has become popular as researchers have found the drawbacks of each particular paradigm in reference to their work.
In (Schultz et al. 1996), the authors attempted to overcome some of the structural issues of a GA and the manual problems of a rule-based system. They wanted to take advantage of the human understandability of a rule-based system and the automatic creation by a GA. The system, SAMUEL, was applied to both problems in simulation and the real world by having the GA write “stimulus-response rule[s] of conditions that match against the current sensors.” First, the rules were made in simulation for many generations, and then the system was placed in a real robot for evaluation. With this system, no learning was done during a run as adaptation, just the standard GA process that changes values between generations through mating and mutation. Additionally, no learning was run in the real system when applied to real-world robot problems. In (Mendes et al. 2001), the authors use GPs to create a system of fuzzy logic-based rules. The rule sets are reportedly more like how humans decide because fuzzy is “more understandable” than hard values, therefore “simple” to understand. The system takes away the programming from the programmer and into the hands of the GP system. In (Towell et al. 1990), the authors used a rule-based expert system to train an ANN. The feedforward network connections and shapes of the ANN are based on the PROLOG-like rules of the expert system. The system was hand tuned to close to the “answer” already, and does not change “shape” afterwards. Then the typical process of backpropagation was done. This still requires the work of an expert system, but may allow for the later automatic expandability of an ANN. In (Tan 1997), the author combines the symbolic knowledge represented in rules and has a FuzzyARTMAP system refine those rules. The importance of this system is the automatic training using an ANN and overcoming
the sometimes lack of understandability of an ANN by using rules. In (Bonarini 2001), the author uses reinforcement learning to create fuzzy rules for a robot. The experience that was gained by the robot was used to modify a system of fuzzy sets. In this method, the training algorithm was able to separate the training from the representation. Although fuzzy logic can be used in a rule-based fashion, it can also be used in real-time in a control system. In (Juidette and Youlal 2000), the authors use a GA to create a fuzzy set function in order to perform path planning. The fuzzy set then weighted the decisions for moving in a space. The system, however, did not adapt to individual experiences but only learned from the overall success in a simulation. In (Kenue 1995), the author used an ANN to tune a fuzzy set for controlling an inverted pendulum cart. In this way, the system was able to use experience to learn online in an ANN but use the representation of a fuzzy set. The fuzzy sets and rules made this system more understandable and combined in the abilities of an ANN in a method that can create agents in an automatic way.

A mixture of training techniques has also been done in order to improve upon an existing structure. In (Wieland 1991), the author wanted to “control a series of unstable systems” such as the inverted pendulum cart problem using a fully recurrent ANN. However, the ANN itself should be able to encode a system to control, but there did not exist a good way to train without input/output pairs. In this system, a GA was used to evolve the weights of the ANN using a fitness function. In this way, the system learned the weights indirectly where a normal ANN would be unable to do so. In (Angeline et al. 1994), the authors use GAs to construct the links of an ANN and use a normal backpropagation for training the weights.
An ANN is good at training weights for certain problems but “the relationship between network structure and task performance is not well understood.” This idea was also done in \cite{Potter1995}, but they used only a feedforward ANN. However, the interesting twist is that it evolves a subset of the network, not the entire network at once. This was done to prevent stagnation of the ANN training by adding additional links. In \cite{Gruau1996}, the authors use a GA in several different ways to see how well it can train an ANN. In “direct encoding,” the system simply learns the weights of a fixed structure ANN while in “cellular encoding” the system learns the structure as well. In their system, a GP is used to program the entire ANN, using the ANN as the computational structure and the GA as the training method.

In \cite{Stanley2002}, the authors evolve both the weights and structure of an ANN using a GA. An interesting improvement of this system is that it uses additional properties of GAs such as speciation, which can be done because of historical markers. These markers are placed on new nodes and links created in the ANN to signify new structures being created. This improvement allows for closely related individuals to be identified and crossover to be done in the mating of two ANNs. The system called NeuroEvolution of Augmenting Topologies (NEAT), “strengthens the analogy between GAs and natural evolution by both optimizing and complexifying solutions simultaneously.” The system allows for the evolutionary training of genes represented in a genotype, which then create the phenotype that is the ANN “brain.” In these systems, the ANN implements the powerful structure that the GA was lacking. The ANN should be able to compute anything, but cannot be trained
without input/output sets, and it is difficult to train with arbitrary structure and internal recurrent links. Now only a fitness function is needed to train an ANN; however, the system does not adapt in real time. Efforts such as (Stanley et al. 2005b) have been done to add on-line learning in a steady-state GA way; however, each individual in the population does not adapt, just replaced in real-time during the simulation.

Other methods have been combined with the ANN structure to replace the training stage. In (Zhang et al. 2000), the authors use PSOs to evolve the weights of a feedforward ANN and the architecture in two stages. The first stage solves for just the weights by running the PSO on the weight values as a non-linear optimization problem for a fixed number of time steps. In the second stage, the architecture is modified by using a density function of how many hidden nodes should exist. By doing it in two stages, the authors attempt to avoid the “moving target problem resulting from the simultaneous evolution of both architectures and weights” (Yao 1993). In (Gudise and Venayagamoorthy 2003), the authors found that using a PSO instead of BP on an ANN made the latter able to converge to the global solution much faster and is able to converge better when the number of training points is small. In (Chen et al. 2004), the authors take multiple steps when creating an ANN. The system uses ACO to make the structure and PSO to calculate the weights. Since ACO is experience based, the nodes are added almost at random “in the walk” and information is shared among individuals to create a feedforward ANN. The best architecture is then chosen and then weights themselves are then optimized using a PSO. This is also done for the “moving target problem” and could be partially attributed to how to line up a structure
of different trees of networks. Different twists have also been done, such as using a GA to optimize the parameters of a PSO \cite{Angeline et al. 1998}. Furthermore, a full combination of steps has been done using a GA to discover the layout and determine what fuzzy set function to incorporate inside each ANN node, but then uses PSO to train the weights. This super hybrid method deemed HGAPSO has shown the ability to control “dynamic plant problems” better than a GA or PSO alone. The combination of PSO with ANN has shown the ability to train on data and structures which an ANN could not otherwise process using backpropagation. Additionally, the system is able to adapt quickly to changes by using PSO.
CHAPTER 5
SOLUTION IMPLEMENTATION

The proposed approach introduced in this dissertation accounts for the three major methods of training: fully supervised off-line, supervised on-line, and reinforcement on-line. The initial observation stage will produce an agent that bootstraps to a level of intelligence that can produce similar outputs to a human in already observed situations with some, but limited generalization. That same agent is then augmented on-line through haptic-based human input to correct for situations in which the original agent did not perform as well as required. Finally, that agent is then put into many different unseen situations to further generalize or in a specific situation to customize it to a specific task. In each additional step proposed, the original agent should improve its proficiency because the gaps in the agent’s knowledge are being filled. Using this system, a person should be able to create an intelligent agent without any programming of the agent itself. However to make this learning approach possible, an algorithm must be selected that can handle each aspect of training.

5.1 Algorithmic Analysis Summary

Summarizing Chapter 4, an algorithm was to be selected that can capture and display human intelligence, scale up from smaller problems to larger ones, and be able to adapt itself online.
to changes in the environment and compensate for noise. In addition, the algorithm needed to be able to properly work in all three learning approaches (Observational, Instructional, and Experiential) even though each has different requirements for training. It was identified that a ANN would be a good computational platform and structure that could display human intelligence. Additionally, ANN matches well with the Observational learning approach of mapping known inputs to outputs for complex problems. Furthermore, a GA was shown to be able to dynamically scale up to problems of different sizes based on need. GAs can also train in a sparse feedback domain based on only a final fitness values without preexisting known correct data and would be able to work with Experiential learning. Finally, Particle Swarm Optimization was found to be a quick training algorithm that can also function based on fitness using the group dynamic which would allow it to adapt. These features complement Instructional learning since it is in real-time and the fitness based on reinforcement is only given for negative events. From these requirements, a Neuroevolution system was selected that mimics the biological process just as we are attempting to mimic the human learning process.

Neuroevolution is a biologically inspired theory that borrows from the field of Evolutionary Neuroscience. This field is primarily concerned with the study of the human brain and explanations of how such a complex organ could have developed over thousands of years. Neuroevolution is a relatively new field of study that takes contributions from neural networks and evolutionary techniques in addition to evolutionary neuroscience.
Evolutionary Neuroscience is the study of the human brain and its relation to primates and biological ancestors. Evolution of the brain is asserted to be similar to Darwinian evolution of the species. Brains evolved complex connections over time, as a result of experiences and reuse. Creating a complex system such as the human brain began with a much less complex system that was progressively enhanced as more environmental challenges were encountered by early humans (Panksepp 1998).

In its most basic terms, neuroevolution uses GAs (Holland 1975) to evolve ever more complex ANNs (McCulloch and Pitts 1943) capable of solving increasingly more difficult problems. GAs and neural networks are common concepts these days. We will forgo a detailed discussion of them here because it was covered in Section 4.2. However, for the sake of completion, we will briefly say that GAs create a population of individuals that represent solutions and proceed to compare these individuals to a fitness standard to determine the “goodness” of each individual. “Good” individuals are allowed to “mate” and move into the next generation and modified while “Bad” individuals are removed from further consideration. In the case of Neuroevolution, these solutions are ANNs that represent the brain structure. These individual ANNs can mate and create offspring solutions with slightly different characteristics. These are then evaluated for fitness against some standard. GAs then seek to progressively keep increasingly fit individuals over several generations of mating and mutating to evolve an optimal solution. GAs, however, lack a formal structure to express intelligent decisions.
Unlike GAs, ANNs have a solid formal structure. ANNs base their legitimacy on the structure and connectivity of the human brain. Many tiny computational units called neurons work in parallel and are driven by inputs from the environment or other neurons. ANNs have the unique ability to be trained by exposing them to examples of a domain to be learned. They form one major branch of machine learning.

One resulting system from neuroevolution research has been the NeuroEvolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen 2002). NEAT evolves the brain from a simple form to a complex form for a given problem. It fuses the best of both worlds with a biological backing. NEAT uses the “DNA” of genetic algorithms to construct the “brain” and evolves it to meet the presented task. The concept is to start simple, present easy tasks for the evolving structure to conquer and then progressively “complexify” the brain, as the simple solutions no longer work as well in the environment when more difficult problems are presented to it. NEAT provides the basis for evolving and computing new agents.

For adaptation of the existing network, a method called PSO (Kennedy and Eberhart 1995) is applied to the network weights. PSO is a stochastic process of problem solving using Swarm Intelligence to adjust in real-time. The basic principle of PSO is to create a group of dissimilar individuals. Each individual wants to be like the best individual in the group, but also want remain close to their most successful past configuration. The process is stochastic but not specifically gradient behavior, thus avoiding getting stuck in local optima. Therefore, like Boids (Reynolds 1987), it includes simple rules that create complex emergent behaviors from a group of individuals. This is excellent for reacting in real-time, and is
mathematically simple. It can avoid non-optimal minima problem in the solution space. PSO is used to optimize the weights of an ANN.

PSO coupled to NEAT allows for adaptation over the lifetime of an individual. Humans are not purely based on DNA, but are instead combination of nature and nurture (Wong et al. 2005). The agents should be able to learn things in an environment in which they are not born with, similar to how an animal is born with instincts but can learn to do additional tasks. However, over its lifetime, the magnitude of the changes to an organism’s structure are limited (i.e., a monkey remains a monkey and a fish remains a fish within its lifetime). The weights of the network are adapted, not the structure of the genotype during lifetime of the agent. These weights exist as small perturbation in order to fix close but inappropriate behavior. It is necessary to adapt to real-time interaction with the user without stopping or re-running the method. Therefore, the PSO algorithm permits additional learning to take place.

5.2 Algorithmic Implementation

In this section, a brief overview of NEAT and PSO is given as implemented as a subset of the overall algorithm Particle swarm Intelligence and Genetic programming for the Evolution and Optimization of Neural networks (PIGEON). PIGEON is a hybrid algorithm that combines NEAT with PSO in order to take advantage key properties of each as written about in Section 4.2.
5.2.1 NEAT Implementation

NEAT is a machine learning algorithm that was introduced by Stanley in [Stanley and Miikkulainen 2002]. It is covered in more depth in [Stanley 2004]. Although genetic programming of neural networks has been done before, NEAT provides a novel method for preserving network novelty through speciation and recombination of networks by using innovation numbers. At its core the computational structure of NEAT is based on a MLP. The structure consists of input, output, and hidden neurons which are connected together by dendrites each of which has an associated weight.

A neuron is the basic computational element of the ANN. The output for the next reaction step is given by the summation of incoming weights multiplied by the output of a connected neuron. This value is then fed into an activation function to produce the final output. The activation function for the current implementation is the Sigmoid function, although other functions such as Gaussian and Arctangent have been examined.

\[
\text{for } j \in \text{Neurons do}
\]

\[\text{Val} = \sum_{i \in \text{Neurons}_j}(\text{Weight}_{ij} \ast \text{Out}_i(n))\]

\[\text{Out}_j(n + 1) = \frac{1.0}{1.0 + e^{-\text{Val}}}\]

\text{end}

An input neuron simply is a special case because its output is derived from external stimuli from the environment and directly injected into the network. This information may be preprocessed in order to make the inputs more acceptable for processing by scaling the input value first. For this implementation, the maximum input value was limited to ±π.

93
There is also a specialization of input neuron called the bias neuron that always outputs one (1.0). The output neuron is the same as a hidden neuron but its output is also used as actions to be output into the environment. These outputs may also need to be scaled to fix the particular action being done.

As an example for the Chaser problem defined in Section 6.1.1, the network consisted of a two inputs and two outputs with an additional bias neuron. A linear version with no hidden layer can be seen in Figure 5.1. More complex examples are also shown such as a feed-forward with one hidden layer in Figure 5.2 and a fully connected recurrent network with one hidden layer in Figure 5.3. For the purpose of reading the network graphs, neurons will have their ID number inside; inputs will always be on the bottom and shown as triangles; outputs will always be on the top and shown as squares; and hidden nodes are distributed in the middle and shown as ovals. Connecting the neurons, the dendrites are drawn as splines with an arrow terminating on the output of that neuron, an ID plus weight listed on the path, and the path is color coded green for positive and red for negative.

Figure 5.1: Linear Network for Chaser
Figure 5.2: Feed-forward Network for Chaser with One Hidden Layer

Figure 5.3: Recurrent Network for Chaser with One Hidden Layer Fully Connected
In order to be a GA/GP, there are several processes that must be met. These processes include crossover, mutation, and selection. Crossover is the mating process where the gene of the parents are combined to produce a new individual. Mutation is the random process of adding slight changes to an individual in order to explore the search space. Selection is the process of choosing the individuals that are designated to mate in order to produce the next generation. This implementation uses a fitness proportional selection that probabilistically selects individuals with the greatest fitness for mating. Additionally, the system also has elitism to carry over individual between generations. NEAT performs all of the actions of the GA on the structure of an ANN.

The mutations allowed on the ANN are: add a neuron, add a dendrite, cold mutations, and hot mutations. Each of these mutations happens with some probability during the mating process. The adding of structure allows for the complexification of the network to scale to the problem (Stanley and Miikkulainen 2002). When a neuron is connected, it is inserted to split a dendrite between two other neurons as seen in Figure 5.4. This is done in order to preserve the internal consistency. Additionally in this implementation, a connection is made between the new neuron and the bias input neuron. In experimentation, this change led to better performing networks when using the sigmoid activation function.

When adding a dendrite, a set of rules must be preserved in order to preserve consistency of the network. Only a single link can be made between two neurons, and no dendrites can feed back to the input neurons. Some work was done restricting the network to prohibit loops in the graph, but it did not prove to be advantageous. As seen in Figure 5.5, the types of
Dendrites can be between two neurons that were not previously connected in a feedforward path, connections among neurons on the same level, self-looping connection for a neuron, and connections that go from the output back into the network.

The Hot and Cold mutation have to do with the modification of the weight values associated with the dendrites. In a Hot Mutation, a weight is chosen at random and a small random number is added or subtracted to the weight. However, the weight is not allowed to grow past a certain value, in this case $\pm 3\pi$. In the case of a Cold mutation, the value for a particular weight is simply chosen at random in the entire range without regard to the
previous value. Cold mutations happen at a much lower frequency than Hot mutations, in this implementation a factor of 5 to 1.

Preserved in this implementation of NEAT, the GA is done in a structure that is held separate from the computational network structure. Similar to biology, there is similar distinction from the genotype and the phenotype. The genotype encodes the network in its “DNA” and contains all of the data needed for mating and tracking. This format provided greater ease to save and load networks, transfer them for clustering, and visualize them for inspection. The genotype is stored in the structure called the genome. The phenotype is the expression of the network and provides the actual execution of the network. The genotype of the network could then be turned into the phenotype for computation during the process called “genesis” (Stanley 2004). During the GA mating process, the genotypes of two individual “parents” are mated together to make a new individual for the next generation. This process in a non-defined structure ANN can be difficult, however, NEAT has a solution to this problem.

One of the most interesting features of NEAT, and one of the main advantages over other GA/ANN Hybrids, is the use of innovation numbers for the purposes of tracking both neurons and dendrites in a network. An issue identified typically with GA/ANN Hybrids, is how to perform the operation of crossover during the “mating” of two networks. As the system gets more complex, identifying the location from where to extract part of one network and replace with parts of the other becomes more difficult. In a GP using Koza trees (Koza 1989), it is not a problem to swap any branch because the culmination of every branch returns a single
number. The structure of an ANN is a graph with multiple input and output connections including recurrent connections. Simply replacing without recovering these links will break this structure. Innovation numbers fix this issue by tracking the creation of structure over time. Every time a new hidden node is created or a new link made, a list is checked to see if that structure has been made before. If it has not, it is given a unique id, else it is given the id of the known structure. Now when crossover is performed, the neurons and dendrites of the same id can be aligned for uniform crossover. During uniform crossover, the DNA of two parents are combined to create a child (See Figure 5.6). For those innovations that match, the neurons and dendrites are selected from the parents at random (See Figure 5.7). Those unique structures that are not matched can also be passed to the child based on the parent with the greatest fitness (See Figure 5.8). This process of structure matching in the genotype is similar to biological markers in DNA such that the coding region that creates a “leg” is only swapped with a “leg” from another individual, not an “arm.” In this implementation, the innovations numbers are kept for the entirety of a run.

Another interesting feature of NEAT is the use of “speciation” in order to preserve new innovations. The concept of speciation is to create “species” by clustering similar genomes together that can only mate with each other. The process of speciation is done by using the innovation numbers on the dendrites. To compare two individuals, the genomes are aligned by the id, if both individuals had the same dendrite the absolute value of the weight difference was recorded, else a count was kept of the disjoint dendrites between the sets. A delta difference between the two values could then be calculated by summing together the
Figure 5.6: Two Parent Genomes
Figure 5.7: Aligning the Two Parents, Swapping Matching Genes Uniformly

Figure 5.8: Resulting Child Genome
total weight differences and number of disjoint multiplied by constants. If the delta between two individuals is less than a threshold, those individuals will be considered part of the same species.

\[
\begin{align*}
Weight_\Delta &= \sum_{i \in (\text{Dendrites}_A \cap \text{Dendrites}_B)} |\text{Weight}_A - \text{Weight}_B| \\
\text{Disjoint}_\Delta &= |\text{Dendrites}_A \oplus \text{Dendrites}_B| \\
\text{Species}_\Delta &= 2.0 \ast \text{Disjoint}_\Delta + \frac{0.5 \ast \text{Weight}_\Delta}{|\text{Dendrites}_A \cap \text{Dendrites}_B|}
\end{align*}
\]

To speciate the entire population, a greedy algorithm is performed to put each individual into the closest bin based on the first individual in the bin. Because of the aforementioned elitism, the first individual in a species is typically the species champion from the last generation held over for another generation. This process of speciation protects innovation because a tolerance factor attempts to keep a certain number of species, typically ten (10), and the number of children that will be produced through mating is calculated for each species. The number of children produced for a species is equal to the average for the species divided by the sum of the species averages times the population size. In this implementations, the rounding leftovers go to the species with the overall champion. The speciation allows for different distinct ANN structures an opportunity to mate, even if the fitness is not near the best. Without speciation, the probability of mating when using fitness proportional selection would be low.
for $j \in Species$ do
  $Average_j = \frac{\sum_{i \in Species_j} Fitness_i}{|Species_j|}$
end

$TotalAverage = \sum_{j \in Species} Average_j$

for $j \in Species$ do
  $Offspring_j = \lfloor \frac{|Population| \cdot Average_j}{TotalAverage} \rfloor$
end

5.2.2 PSO Implementation

PSO at its heart is a physics-inspired social optimization system. The system, similar to a GA or GP, starts with a population of random individuals that each have a list of numbers which would represent something in the solution space. These real valued numbers could represent anything like the constants in a physics equation or the length of an antenna. For the purposes of a hybrid PSO/ANN system, these values represent the weights in an ANN between nodes.

As part of the physics model, the values can be represented as a position in N-dimensional space, where N is the number of values needed for a problem. The N-dimensional space is therefore the solution space, and any point in this space represents a solution for a given problem. From each solution, a fitness needs to be calculated on how well that solution solves a given problem. The way PSO works is to have each solution in a population represented by a particle with mass and this particle is influenced by outside forces that move the particle
through the solution space. As the particle moves, the new positions represent new solutions and therefore new values in the internal list. In theory, the particles should eventually converge to a location in the solution space that should give the optimal answer.

In the physics of the problem, the particles have mass, velocity, frictional drag, and outside forces. The outside forces for the problem are social forces based on moving “toward” other good solutions. In the standard PSO, the only social forces that are used are forces in the direction of the best individual and in the direction of where an individual particle was best at some point in the past. Every individual particle has memory of the position of when it received the best fitness. An attractive force is made to move in the direction of when it was personally best. If the particle ever gets better, it would replace the best position with the current. Another attractive force is made in the direction of the best individual of the population. The best can be found simply by evaluating the entire population for fitness and choosing the current best. The standard PSO uses only the globally best individual, however, there have been studies on using only the locally best based on some distance metric (Suganthan 1999).

These two force vectors are themselves not used directly. To make the process more stochastic, a small random numbers are multiplied by the forces in each of the N-orthogonal directions. This feature, along with the randomized initial position, eliminates a pure deterministic outcome of the algorithm. Since the influence of the position of the particle is done on the force level, a second order micro-simulation is done to solve for the next position of the particle at the next time step. Because the system is second order, the particles have ve-
locity and therefore momentum. The momentum, similar to backpropagation, is attributed to allow the particles to get out of local maxima. The frictional drag conversely slows down the particle to prevent instability and eventually stop a particle given no outside forces. The combination of these features allow the PSO to function as a stochastic non-linear optimization system which works well in several real valued domains ([Kennedy and Eberhart 1995]).

\[
\begin{align*}
\text{for } j \in \text{Positions do} & \\
\text{ForceSelf}_j &= c_1 \times \text{Rand} \times (\text{BestPosition}_j - \text{Position}_j(n)) \\
\text{ForceGlobal}_j &= c_2 \times \text{Rand} \times (\text{GlobalBestPosition}_j - \text{Position}_j(n)) \\
\text{Velocity}_j(n+1) &= \omega \times \text{Velocity}_j(n) + \text{ForceSelf}_j + \text{ForceGlobal}_j \\
\text{Position}_j(n+1) &= \text{Position}_j(n) + \text{Velocity}_j(n+1)
\end{align*}
\]

The social forces abstract the search for the solution from the actual problem being solved. The motion of the particles is based on how a set of individuals do without knowledge of the problem. The problem-specific parts relate to the representation, of which PSO has none of its own, and the calculation of the fitness.

The standard example of PSO is a population of bees. Imagine these bees are looking for a flower. The bees do not know the position of the flower, but they can sense how close they are based on a pollen sensor. They can also remember where they were when the pollen sensor was the highest. Finally, the bees release a pheromone based on their pollen sensor and other bees can detect their location based on it. The bees are initially scattered in an environment and each checks their own pollen sensor and pheromones released by others. The more bees, the greater possible area may be covered on the initial scattering. Each bee
makes an independent decision to turn towards the bee with the highest pheromones. Since the bees are already flying and limited in speed, they cannot turn instantly but begin to move in the direction they decided to go. On later time steps, some bees may detect that while moving to the best they are detecting less pollen than before. They therefore decide to split the different in a combination of going to the best and going where they detected more pollen. This new flying vector may lead them to new territory to become the best bee. This process is repeated continually until all the bees eventually swarm around the single flower. This solution was found without knowing the location of the flower, but instead from the simple intelligence of each bee and the group dynamics.

5.2.3 PIGEON Implementation

As referenced in Section 4.2.5, many authors have combined PSO and ANN as a means for training the network. Previous research in literature has found that PSO is able to converge quicker than backpropagation in some domains (Gudise and Venayagamoorthy 2003) and that the algorithm is invariant to the ANN structure of the network being solved (Zhang et al. 2000). Therefore, PSO is an attractive algorithm to us in combination with an ANN. The ANN structure provides the generalized computational framework that the PSO lacks. While previous authors have combined PSO and ANN using fixed networks (Gudise and Venayagamoorthy 2003), cascade networks (Zhang et al. 2000), recurrent networks, and in combination with ACO to find the structure and then PSO for the weights (Chen et al.)
no one, as of the timing of this dissertation, has combined together NEAT with PSO as an algorithm.

The new algorithm presented here called PIGEON, which stands for Particle swarm Intelligence and Genetic programming for the Evolution and Optimization of Neural networks, combines together the strong features of NEAT with the numerical optimization of PSO. The NEAT algorithm uses the same GA to compute an ANN to produce a solution in addition to the other good features such as innovation numbers, speciation, and complexification. The PSO is only used to adjust the weights of the dendrites between neurons. Since PSO does not have any structural information, the structure of the ANN for PSO to optimize would have to be assumed to be fully connected barring outside information. NEAT, however, would create the structure for the system.

The operation of PIGEON is tied to the way in which NEAT functions. In order to operate on the correct weights, the dendrites are lined up based on the innovation numbers to properly insure that corresponding dendrites are being combined (see Figure 5.9). The normal PSO algorithm can then be used based on the individuals best previous performance weights and the weights of the best individual. PIGEON retains the speciation of NEAT and limits the PSO combination of only those individuals that are of the same species, similar to the way NEAT only allows mating from the same species. Each species already has a champion, it naturally leads to be the best individual that gets an attractive social force. Additionally since those individuals are among the same species based on a “distance” delta, it also leads to the local champion concept since individuals will only be influenced by those
close in the solution space. Since individuals of a species may still have disjoint dendrite sets, dendrites that do not line up between the local best and the current individual will not have an attractive force. However, the weight will still be influenced by the attractive force of the past best values and momentum.

![Figure 5.9: Aligning the Individual with Best Self and Global Best for PSO](image)

Although PSO can be thought of as a social learning phase rather than biological one, the end effect is very similar to the way mating works except with a higher converging element of social forces rather than the chance of crossover and mutation. Since PSO is an optimization algorithm, the main emphasis is on convergence to a solution while a NEAT creates greater
complexity and moves into new solution spaces. At the creation of PIGEON, two different but similar algorithms were created from the combination of PSO and NEAT named Chain and Alternate.

The concept behind Chain was to allow the standard NEAT algorithm to attempt to find the correct structure and relatively optimal weights for a fixed amount of time. Then as a second stage run PSO to adjust the weights of the “correct” structure to find the optimal weights. One of the main influences of this method was from hybrid GP/ANN literature written about not evolving structure and weights at the same time (Yao 1993). Experiments were run on when the switch between NEAT and PSO should be for the problem set. These results are presented in Section 6.2.2.3.

Alternate, on the other hand, switches back and forth between NEAT and PSO evolution steps. NEAT would evolve structure normally for one generation; then the PSO method would be applied for a fixed amount of time; and then NEAT would again be run. Alternating back and forth between NEAT and PSO allowed new structure to be created then socially optimized. Experiments to determine how much PSO to run for every NEAT generation are presented in Section 6.2.2.3.
5.2.4 Code Implementation

The software was implemented in C++ using modular classes with an emphasis to keep the algorithms, learning methods, and simulation separated from each other. Heavy use of inheritance and polymorphism was used such that different sections can be interchanged easily for purposes of both testing, evaluation, and code use. For example, the algorithm can be replaced without affecting the simulation or training method. Since the all algorithms are related on the computational level, populations or individuals created under one algorithm can be loaded by another to continue to be optimized. This feature allows a set of agents trained under one learning approach with a specific algorithmic method to be loaded by another learning approach with another algorithmic method. This feature of the framework explicitly allows for multi-stage learning.

However, the most useful feature is the ability to change the simulation/problem independent from the rest of the system. To define a new problem, the number of inputs, outputs, and environmental variables need to be defined; the preprocessing stage to create inputs; the discrete state-space update for the physics model, and the fitness function for evaluation. The virtual functions are called using fixed calling structure from the base class, however, the proper simulation gets called. This allows for the addition of new problem to be solved by the algorithm and training system.

Additionally, the three dimensional (3D) OpenGL display is integrated to graphically represent the problem. A generic joystick library handles input axis values and force-feedback
for multiple devices at the same time. All the training methods, learning algorithms, simulations, and settings for each can be set from the command-line interface to allow complete configuration without a re-compile. The overall codebase for approach, algorithms, visual interface, and clustering currently approximately eighteen thousand (18000) executable lines of codes not considering comments, whitespace, and other C++ syntax such as brackets.

Finally, evolving a GA system can potentially take long periods of time to train. Fortunately, GAs that do not contain interaction of agents or co-evolution become embarrassing parallel problems that can be sped up by using a computing cluster. The system is programmed using a typical scatter/gather method ([Cutting et al., 1992] where individuals are farmed out to slave nodes on the cluster and the fitness results are collected on the master node to allow for sub-linear operation. This allows for almost linear speed up of the training process. For example, a typical evaluation of an individual for one simulation takes approximately 0.3 seconds. However, because random numbers are involved in the evaluation, the fitness of eight (8) evaluations are averaged together to get the fitness of an individual. The typical population size is 150 and a typical generation length is 1000. Additionally, because the results of a run are not deterministic, 30 runs are done per algorithm to get a better statistical comparison. These factors need to be multiplied together to obtain the total run time. Normally, if executed on one machine linearly this would result in a time to finish of 125 days of computation. However, because the available clusters have over 128 processors, this time is lowered to approximately one day, which is a more reasonable amount of time.
5.3 Learning Approaches Implemented

As mentioned in Section 5.2.4, it was important to have the system flexible and function under a single codebase. This allowed the same simulation code to be called regardless of the learning method to ensure a proper comparison. Additionally, since the same algorithmic framework is called, each of the following learning approaches can be tested using multiple agent training methods for comparison. These experiments are shown in Section 6.2.

The following learning stages share much of the same functionality, however, they differ mostly in way the agents are evaluated. In the Observational learning stage, the primary evaluation of the agent is based on the similarity to the observed human given the same environment. In the Instructional stage, the goal is to be the closest to the human by being the current best and receive the least penalty. In the Experiential stage, the goal is to gain the best fitness defined by the problem domain.

5.3.1 Observational Learning Stage

The Observational learning stage actually consists of two parts: the data collection phase and the agent training phase. In the data collection phase, a human expert is given several scenarios in which to perform a specified task. The human is in control of a single simulated entity and the goal of the system is to mimic the actions performed by the human in the same environment. The collection is done in real-time on the computer using the joystick interface.
for a short duration. This time is kept to 30 or 60 seconds depending on the experiment being performed. The initial environment is randomized to ensure a degree of generality and the expert is tested over a series of ten (10) trials. While the human is performing, the computer is collecting the joystick axis information at 100 Hz. The number of axes can differ based on the experiment being run. Additionally, the state of the agent, specified by the state space model, and any relevant environmental variables are also recorded at 100 Hz. At 100 Hz, the discrete time samples can almost be considered continuous for the purposes of simulation due to the nature of the physical system. This data sample stream is complete enough to entirely reconstruct the run for later playback. The human expert is given the option during recording to reject recording trials if they believe they performed too poorly. Although this part is not in the spirit of purely unobtrusive observation, it does uphold the goal of this dissertation of training an intelligent agent without programming, rather than the goal of explicitly matching an individual including their flaws.

Once this information is captured, it is stored on the disk and kept as a record of expert performance of a task. This information is used by the agent training stage as “truth” in performance, even if it might not be optimal. During the training phase, the computer agent is presented with the same environmental information that the human received during the run. The environmental information is typically not used directly by the network and must be preprocessed. The preprocessing stage must be implemented as part of the simulation. The typical operations are to convert obstacles into local coordinate space versus global, to filter potentially extraneous information, and to normalize the inputs to be in an acceptable
range for the network. Again, while this part can be considered work for the programmer or knowledge engineer, it is currently a necessary step in order to simplify the problem, however, this step would also be required for a hand written system.

Once the agent receives the processed information, the network is executed and an action is taken from the output neurons. These outputs directly equates to the human’s joystick axis operation. These values are therefore directly comparable in order to find the fitness of the individual. The fitness of the individual is determined by the sum of the fitness for every time step. On every time step, one hundredth (1/100) of a second, the difference in “joystick” actions is calculated as the sum of the squared error. The maximum fitness for a time step would be if all actions were exactly the same and given a value of one hundredth (1/100) of a point, while the exact opposite action would be given a zero. This is done to scale the fitness of the agent to a perfect score of the number of seconds of operation. The overall fitness for an agent is given by the average of the fitness across eight of the ten trial runs. The last two trial runs are not used in training and are kept as a validation comparison at the end of the run.

<table>
<thead>
<tr>
<th>Data: Action values limited from 0.0 to 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TotalFitness = 0</td>
</tr>
<tr>
<td>for t = 0:Dt:Time do</td>
</tr>
<tr>
<td>Error = ( \sum_{i \in \text{Actions}} (\text{HumanActions}_i - \text{AgentActions}_i)^2 )</td>
</tr>
<tr>
<td>Fitness(_t) = Dt \ast (1.0 - \sqrt[</td>
</tr>
<tr>
<td>TotalFitness = TotalFitness + Fitness(_t)</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>
All the algorithmic methods used involve populations of individuals and either a generational or iterative approach. Therefore, the fitness of the entire population of individuals must be taken, and the operations for learning must be performed as described in Section 5.2. Even though the evaluation time of a single agent happens faster than real-time, the total run typically takes much longer than the original observed data length. At the end of a run, a population of individuals exists that should perform the same actions as the human given the same environmental input. However, this does not necessarily mean that the agent could perform the task on unseen data or even whether the agent could perform the task at all. The population “champions” are tested both on the previously unseen validation data in “ghost mode” where actions are done by the computer but do not affect the simulation. Those actions are compared to the human and scored in a way similar to Observational learning, however, no learning is performed. This score is used to validate that the agent has matched an individual expert in operation and has generalized to other situations. Additionally, the agent is tested in actual performance by being placed in new, previously unseen situations and tested for competency using the fitness functions used in Experiential learning. These values are used as a point of reference for learning across the different stages and are given in Section 7.3.1.
5.3.2 Instructional Learning Stage

The Instructional learning stage is very similar to the Observational learning stage. The main differences stem from the training operation. Because this stage requires human interaction, the simulation operation must happen in real-time. The interaction is performed haptically by using force-feedback joysticks. A library was written to allow the force-feedback to be independently sent to each axis, given the desired position. A control loop implements the actions of the computer agent physically, which can then be felt by the human trainer. Based on much of the work listed in Section 2.3, the haptic interface can give an increased understanding of timing and motion in the training phase.

During this training phase, all of the agents interact with the environment simultaneously, in real-time. The necessity of this relates to the way in which the learning algorithm operates. Unlike the other stages of learning where there was on option of learning algorithm, the algorithmic method used for real-time operation was preselected and fixed. As part of PIGEON, the training is done using PSO only on the weights of the ANN. During training, the actions of all of the agents of the population are evaluated and sorted based on similarity to the human coach over a set synchronization time; one (1) second. The agent that best matches the human action is selected as the “champion agent.” Although the internal decision-making process of the coach is not known, the weights of the best individual are known during that one second period. The weights of all individuals can therefore be adjusted using PSO. The PSO-only technique was selected for its fast convergence properties and its
quick computation time. Both of these factors are needed for real-time learning because the generational computation of the other algorithms such as NEAT or the Chain/Alternate method do not lend themselves well to this type of operation. Additionally, techniques such as rtNEAT (Stanley et al. 2005b) are not applicable because of the synchronous nature of the problem. The agents experience real-time adaptation during training.

The feedback felt by the human coach is the output actions of the current champion agent. The agents operate themselves like “thirty monkeys attempting to learn to drive a bus” (Spector and Klein 2002) all vying to be the co-pilot. The human coach has the option of either allowing the current champion to drive or take control by holding the trigger button and moving the joystick. By allowing the agent to drive, the coach is effectively condoning the agents actions and providing a degree of validation. This validation may happen in states were the coach may have preferred a different move, but still have found the current move acceptable. If the move is unacceptable, the coach can overpower the computer agent and learning takes place. The coach has several responsibilities during training including the modification of the agents actions, determining the validation of the agent, and whether a training session is usable. While the coach is supposed to show the agent how to get out of tricky situations, the coach may deem the training session a failure and revert the population weights to the beginning of the session. On average, there are ten (10) usable real-time training sessions of an agent for the coach to give and receive feedback although the coach could determine more or less as needed.
The coach then fills out a subjective survey on the teaching experience. That information includes the perceived performance of the agent, the “intelligence” expressed by the agent, and the whether the force-feedback helped in the training process (see Table B.1). This information will be coupled with the objective fitness tests on the experiential fitness function to gain a comparable metric. Future work has been proposed to use the same coaching process in the opposite direction by instead assuming the computer agent is more proficient and training a human participant. This work however is outside the scope of this dissertation (see Section 8.4.1).

5.3.3 Experiential Learning Stage

The Experiential learning stage allows the agent to operate on its own and learns by attempting to maximize the fitness score for the given simulation. This learning stage does not involve a human either through direct interaction or through recorded data. The computer operates within the simulation receiving a fitness score that represents a score for proper operation.

The fitness score therefore is very important in steering the agent toward the correct solution. The fitness score cannot determine what action should be done in a given situation; however, it can grade the resulting situation for correctness. Since the fitness function is the only grading value, a properly written fitness function that can lead to the optimal answer is important. While this can be significant effort for the programmer to properly devise a
fitness function, it is necessary to implement one during the creation of simulation, as it will be very domain specific.

During training, the environment is randomized and the agent is placed into the simulation. The simulation can run faster than real-time because it does not involve human intervention. The individual agent’s fitness is the average of eight (8) evaluation runs to increase generality and help eliminate outliers. In addition, the number of runs was chosen to provide parity to the eight (8) evaluations in Observational learning. All agents “practice” in the environment and are graded by the fitness function. These agents are then processed by any of the algorithmic methods to produce the next agent set and the process is repeated for a fixed number of generations/iterations. The agents experience many unseen random situations in which they have not previously seen. At the end of the training, the population is saved and the best individuals are recorded.

The agents produced by this stage can be graded purely on the fitness function; however, these agents also need to be graded subjectively for competency. Agents produced through Experiential learning can sometimes exploit simulation features to produce high scoring agents based on fitness but not as expected. Additional information on these cases will be discussed in Section 7.3.3.
5.3.4 Combination of Stages

The software for the system has been designed to combine the multiple stages of learning together into a cohesive framework. This ability allows an agent trained in one stage to be loaded into the next to continue its training. Each stage, with its own flaws, is augmented by another stage, ultimately resulting in a better performing agent.

As mentioned in Section 1.2, humans follow a multi-stage learning process. Similarly for computers, it is hypothesized that a single approach will not yield an agent of human intelligence. The FALCONET system addresses these issues by feeding each stage into the next in a natural learning process. The physical implementation of each of the individual stages is listed in the previous sections.

The Observational stage is the bootstrapping process which can be run for a long period of time and on a large collection of expert human data. The output of this process is not only a competent agent, but a hopefully a competent population of agents. Because of some of the difficulty in potentially matching agents across runs since the innovation numbers no longer align, populations of agents are transferred between stages. Some analysis has been presented on the comparison of continuing to reuse a population or creating a population from a best agent seed in Section 6.3.4.3. Using a seed agent, a population is created using the same structure and initial weights, but each of the weights is perturbed by a hot weight mutation. Using this method, the entire population will not begin completely homogeneous.
The Observational-based agent can then be passed to either the Instructional or Experiential stage.

For the Instructional stage, the population of agents can be created from scratch. However, the structure of the agent would be unknown. Combined with the fact that the training method for the Instructional stage does not add new connections, the agents in the population would have to be created fully connected to allow them to learn. These limitations, coupled with the real-time nature of Instructional learning, would likely not lead to robust agents. Experiments were done to show this fact in Section 7.3.2. Typically, the Instructional stage would be run on an agent that is already competent. This agent would hopefully already have the structure and a close set of weights to the “optimal” agent. The Instructional stage is then used to refine undesirable traits in the agents. The coach will work with the population of agents until satisfied with the results. Again, the end result of this process is either a population of agents or a single best agent. The result is then passed to the Experiential stage.

The Experiential stage allows the population of agents to experience the simulation in an open training environment and to learn by practice. This process can be done using agents from scratch and can produce highly functional agents. However, it is desirable to use this stage to fill the gaps of the other stages by testing the agent in a wide variety of new scenarios. By experiencing the world in many different configurations than observation and instruction allow, the agents can gain more generality to the problem. The results of this stage can be used as the final resulting agent or fed back to the Instructional stage.
Some thought was put into creating a looping validation process between the Instructional and Experiential stage, however, the time and scope were deemed outside this research.

5.4 Implementation Summary

A software framework has been built to create autonomous agents. This framework used to implement the approach FALCONET and the method PIGEON is constructed of several smaller reusable modules that are to be tested and configured to meet the needs of the problem. PIGEON is a combination of NEAT and PSO used in different ways, depending on the learning approach. FALCONET is a learning approach that encompasses Observational, Instructional, and Experiential learning into one system that would learn in a way similar to the way a human would. Many experiments were run to find the best configuration of PIGEON for each learning stage. Additionally, many were executed to show that the proper order proposed in the hypothesis is better than any other combination. The following chapters will cover the testing and experimentation of the methods and approaches.
CHAPTER 6
ALGORITHM SELECTION

6.1 Experimental Simulations

The testbed domain for FALCONET, because of its inherent link to the human learning process, was selected to be simulations involving motor-skill transfer using haptic interfaces. This general domain covers a majority of problems where the only inputs to the environment are joysticks, steering wheels, and/or pedals. These input devices have variable input and, more importantly, can have variable feedback capabilities. The simulations do not include buttons, keyboards, mice, or any non-haptic feedback interfaces. It was deemed important to limit the human-controlled output actions into the simulation to analogues which the computer could control. This feature allowed the human and computer to have an equal interface to the simulation for fair comparison. For this particular simulation, all actions are done on joysticks because of their easy availability.

There are a wide variety of motor skill tasks that are normally done by humans. These tasks need to operate in real-time and can be very reactive in nature because of the continual stream of decisions being made. While some planning and long-term decision-making may be involved, dealing with the current situation reactively is the primary concern of this study.
The experiments selected were chosen in increasing levels of difficulty. They are designed to test the robustness of both the learning approaches and the algorithmic methods. The simulation domains chosen are: Chaser, Sheep, Crane, and Car. These are described in detail in the following sections.

6.1.1 The Chaser Simulation

The Chaser simulation is the simplest one in terms of both the environment and the goals. The task is to control an agent to chase an preprogrammed entity as close as possible. The agent has a direction and position and is governed by a simple first-order physics simulation. There are only two (2) control outputs represented by the two axes of the joystick, which represent linear and angular velocity. The fleeing entity is controlled by simple rules: attempt to move with a vector away from the chaser, move faster as the chaser approaches, and attempt to stay near the center of the playing field. Additionally, some random movement was added to the fleeing entity to add to the difficulty of the problem.

For this simulation, only two (2) environmental inputs are given to the agent through the preprocessing step: the relative distance and the relative angle difference based on heading. The human trainer is given a aerial view of the environment and of the chase. An example of the interface can be seen in Figure 6.1. Although the human might use other visual information, these inputs were determined sufficient for building an agent.
At the start of the game, the chasing agent and fleeing entity are placed randomly on the playing field. The chaser is represented by the red cube with the heading line while the fleer is simply a blue cube. The agent then must attempt to catch the fleer by staying as close as possible to it. The problem is very reactive in nature because it deals with the movement of the fleer. However, an intelligent agent can develop a control system that may predict motion as well as simple pursuit. The grading criterion during the simulation is based on the distance and angle between the chaser and fleer for every time step. A perfect score would be if the positions coincided with the chaser facing the fleer. The worst score would be to be on the other side of the map facing the opposite direction. A more formal definition is given in the following code block.
Data: Playing Field (3x3), Fleer Limited (2x2), Angles Limited ±π (rad)

\[
TotalFitness = 0
\]

\[
\text{for } t = 0 : Dt : \text{Time do}
\]

\[
\text{Error} = \sqrt{(\text{Fleer}_x - \text{Chaser}_x)^2 + (\text{Fleer}_y - \text{Chaser}_y)^2 + (\text{Fleer}_\theta - \text{Chaser}_\theta)^2}
\]

\[
\text{Fitness}_t = Dt * (1.0 - \sqrt{\text{Distance}} / \sqrt{2.5^2 + 2.5^2 + \pi^2})
\]

\[
\text{TotalFitness} = \text{TotalFitness} + \text{Fitness}_t
\]

\[
\text{end}
\]

6.1.2 Sheep Simulation

The Sheep simulation adds a level of complexity to the Chaser simulation. The task is to chase a herd of sheep into a predefined pen. The complexity comes from the multiple targets and the additional planning required to accomplish the goal. However, the joystick controls and physics are exactly the same as the Chaser simulation. There are two (2) control outputs: the linear and angular velocity. The sheep, which have direction in this problem, are given simple rules: run away from the agent, move faster if the agent gets close, and do not to bunch up with other sheep. A circular fence was added as an outside border to prevent the sheep from running off of the map or getting caught in a corner.

In the Sheep simulation, the agent is presented with eleven (11) environmental inputs from the preprocessor. The first three (3) inputs are for spatial awareness and include: the distance from the center of the pen, the relative angle to the center, and the current angle of the agent with respect to “north.” The relative distance and angle help with the approach of the pen while the actual angle could be used for setting up specific position-based maneuvers.
The other eight (8) inputs are based on the position of the sheep divided into pie wedges. Pie wedges are a common practice in dealing with multiple items in a scene without having the number inputs change with the number of items. The space is divided into eight (8) equal slices of 45 degrees each, and centered at the agent. These wedges are relative and move based on the location and heading of the agent. A visual example can be seen in Figure 6.2 with green pie wedges. The input for a particular wedge is based on the closest sheep inside the wedge. The input would be zero (0.0) if the sheep was next to the agent and a maximum size of two (2.0) if the sheep was outside the viewing radius.

![Figure 6.2: Example of Sheep Pie Wedges](image)

In the simulation, the agent is given by a red box with a heading line and the sheep are represented by blue boxes with heading lines. The black outer ring is the fence to stop the
sheep from leaving the screen while the red inner circle represents the holding pen. When a sheep enters the center holding pen, the sheep will turn black (see Figure 6.3).

The fitness of the Sheep simulation is based on the distance of the sheep from the center pen with full points awarded if the sheep enters the center pen. Once the sheep enter the center pen, they stop moving and are no longer considered by the pie wedge sensor. A full score would be if all sheep, normally sixteen (16), were captured in the pen within the 60 second time limit. However, to give more than 16 discrete scores, points are awarded in a way similar to partial credit if the agent brings the sheep near to the pen. The fitness is written in such a way that the fitness gained by bringing the sheep close to the pen is not more than capturing a sheep. The pseudo-code for the fitness is given the following code-block. This type of fitness was created to give a gradient pressuring toward moving
the sheep inward, even without capturing a sheep. This would reward an agent that almost
captures a sheep rather than no points at all. It has been found that rewarding the system
to perform a sub-goal, to bring the sheep toward the pen, can help the system learn the final
goal, to put the sheep in the pen.

| **Data:** Playing Field (Radius=1), Pen (Radius=.25) |
| **Data:** Distance are to Center |
| Captured = 0 |
| SheepScore = 0.0 |
| **for** i ∈ Sheep **do** |
| | if Distance\(_i\) < Pen\(_r\) **then** |
| | | Captured = Captured + 1 |
| | else |
| | | SheepScore = SheepScore + \(\frac{1.0 - \text{Distance}_i}{1.0 - \text{Pen}_r}\) |
| | end |
| end |
| **if** Captured ≠ |Sheep| **then** |
| | SheepNormal = \(\frac{\text{SheepScore}}{|\text{Sheep}|-\text{Captured}}\) |
| else |
| | SheepNormal = 0.0 |
| end |
| TotalFitness = Time * \(\frac{\text{Captured} + \text{SheepNormal}}{|\text{Sheep}|}\) |

### 6.1.3 Crane Simulation

The Crane simulation adds three new complexities to the problem: multiple joysticks, obsta-
cle avoidance, and multi-stage planning. The goal is to operate a loading crane at a seaport
and move multiple boxes into a specified area. The boxes, typically ten (10), are randomly
placed on the playing field and the drop off point is always in the same location (see Figure
6.4). The physics for this simulation is different in that there are now four (4) action outputs. One joystick controls the independent X-Y axes respectively, while the other joystick controls the Z axis and the gripper. Each action controls the velocity of a first order system for each state. An increased number of outputs mean more actions to consider and it also means larger networks. The reason for this is based on the structure of an ANN because an additional output requires incoming weights and connections. Adding twice the outputs to a feed-forward network would approximately double the size. For a fully connected network doubling the outputs increases the size by about factor of four. This is caused by the additional weights between other outputs, inputs, and hidden nodes to complete the new structure.

Additionally for this simulation, collision physics is considered for both the crane with boxes and boxes with each other. This requires avoidance of static obstacles to be considered when returning the payload. Finally, there is an element of multi-stage planning to consider when operating a crane. The first stage is to select which box to pick up and maneuvering the crane gripper to capture it. The second stage is once the payload box is picked up, to avoid other boxes and place the payload in the destination square.

There are twelve (12) inputs from the environment. The first four (4) are related to the crane state: angle to the center of the destination square, the distance to the center of the destination square, the height of the gripper, and a binary value that indicates whether the gripper is carrying a box. These inputs were selected as a near minimal set that the agent would need in order to perform the operation. The other eight (8) inputs are from
a similar pie wedge scheme that was used in the Sheep simulation. The human observer is
given two views of the same scene: a top down aerial perspective and a third person moving
perspective (see Figure 6.4). It was originally found that the human was having problems
lining up the crane above the box with just the third person view which necessitated an aerial
view. However after some visual queues and practice, the aerial view was not used very often
in later stages. The gripper is represented by the red C-shaped object, the blue cubes are
the payload boxes, and the green cube is a payload box being carried by the gripper. The
red outline square on the field is the destination area.

Figure 6.4: Example of Crane Interface

The fitness for the Crane game is based on the number of boxes that are brought into
the destination square and the Manhattan distance of the box from the destination square.
The number of boxes collected is the main metric, with the closeness being a partial credit
to bias the agent to moving the boxes toward the goal. The distance-based partial credit
never rewards more points than collecting a single box. The pseudo-code for the fitness is
given in the following code-block.

| Data: Playing Field (3x3), Destination Square (0.5x0.5) |
| Captured = 0 |
| BoxScore = 0.0 |
| for i ∈ Boxes do |
| if Box_i is in Destination Square then |
| Captured = Captured + 1 |
| Distance_i = 0.0 |
| else |
| Distance_i = Manhatten(Box_i) |
| BoxScore = BoxScore + \frac{3.75−Distance_i}{3.75} |
| end |
| end |
| if Captured ≠ |Boxes| then |
| BoxNormal = \frac{Error}{|Boxes|−Captured} |
| else |
| BoxNormal = 0.0 |
| end |
| TotalFitness = Time \times \frac{Captured+BoxNormal}{|Boxes|} |

6.1.4 Car Simulation

The Car simulation tests a different set of complexities: dynamic obstacles and a complex
physics model. The goal of this simulation is to be more similar to a real-world application.
While the environment gets more complex, the number of action outputs has been reduced
back to two (2): steering wheel and a gas/brake combination. Because of the unavailability
of a force-feedback steering wheel and pedals, the simulated car is operated with a joystick by having the X-axis operate the steering wheel and the Y-axis operate the gas in the positive position and brake in the negative. The Y-axis also operates a combination of reverse and brake if the vehicle is required to move backwards. The goal of the simulation is to operate a car around the loop of a track involving traffic traveling in different directions in their respective lanes (see Figure 6.5). The car must travel via the center road forcing the situations of merging into traffic from a side road and crossing a lane of traffic to a side road. The goal is to make as many laps as possible while staying in the proper lane without colliding with other cars. The computer controlled preprogrammed cars attempt to stay in their own lane at a constant speed, but will actively avoid collisions.

Figure 6.5: Example of Car Interface
The agent receives eleven (11) inputs from the environment. The first two (2) inputs are based on the X and Y locations of the vehicle on the track. This information can be used to differentiate turning operations or to regulate the speed of the car at different locations on the track. The next input is the relative angle of a “carrot point” down the road. A carrot point is located on the center of the lane at a location a fixed distance in front of the car. This point can be used for steering in order to stay in a lane. The other eight (8) inputs are from the pie wedges around the car. Each wedge gives the relative distance of another car from the agent within its discrete sensor range. These wedges can be used as the distance from other cars in both, the merging and normal driving situations. The wedges offer similar blind spots if a car is beyond another car and not directly visible. The human is presented with a top down view of the driving simulation (see Figure 6.5). Some testing was done using the first person perspective of normal driving, however, it was deemed difficult without better peripheral vision or a rear view “mirrors.”

In the simulation, the car being controlled is the green box with a heading line. The red boxes represent the cars traveling in the clockwise direction in the inner lane and are considered the oncoming traffic. The blue boxes represent the cars traveling in the counterclockwise direction on the outer lane. The center vertical lines represent the side road for merging into traffic or lane crossing. The agent is meant to make counter-clockwise loops on the right hand side of the course and center line by following the blue line (see Figure 6.5).

The fitness of the Car simulation is based on several factors, with the major one being the number of laps made. Because of the maximum speed and time allowed, the maximum
score received would be to make five (5) laps around the track. However, there are many penalties that can happen during driving. If the agent deviates more than a car width from the center of the lane, i.e. completely out of the lane, the agent does not get credit for that portion of the lap. The leaving the lane penalty should force the agent to stay in the lane to gain fitness. If the agent causes the slowing of other traffic or tailgates too close, the agent will get “honked at” by the other cars, and will receive a penalty, except for the specific situation in which the car must stop to make the left turn at the top of the track (see Figure 6.5). Consideration is made for this because this constitutes a proper turn and no penalty should be given. The agent is also penalized heavily if it ever hits any of the other cars on the road, which is enforced any time the collision detection routine is called.

The system is weighted in such a way that the maximum honking penalty can at most lose one (1) lap, while the maximum for hitting another car will be to lose all laps. The pseudo-code for the fitness is given in the following code block. The fitness score is devised to get the agent to originally stay in the lane making as many laps as possible, regardless of collision or being “honked at.” It then can improve its fitness by avoiding the heavy penalties of collision while maintaining driving distance. Finally if the system makes several laps and avoids collisions, it can then try to not receive the somewhat minor penalties for tailgating.
### Data

Angles are from Center of Right Loop

- $\text{HonkTime} = 0$
- $\text{HitTime} = 0$
- $\text{AngleDistance} = 0$

```
for t = 0 : Dt : Time do
    if Collision Detected then
        HitTime = HitTime + Dt
    end
    if Tailgating Or (Causing a Slowdown And Not Crossing) then
        HonkTime = HonkTime + Dt
    end
    if (Car In Valid Lane at t And t-Dt) And (Moving Counter-Clockwise) then
        AngleDelta = Angle_t - Angle_{t-1}
        AngleDistance = AngleDistance + AngleDelta
    end
end
```

$LapScore = \frac{\text{AngleDistance}}{2\pi} - \frac{\text{HonkTime} + 5\times \text{HitTime}}{\text{Time}}$

if $LapScore < 0.0$ then
    $LapScore = 0.0$
end

if $LapScore > 5.0$ then
    $LapScore = 5.0$
end

$TotalFitness = Time \times \frac{LapScore}{5.0}$

The physics used and the given problem are modeled after the DARPA Grand Challenge test track, more information can be found in (Patz et al. 2008).

### 6.2 Comparison of Algorithmic Methods

As a first stage of testing the FALCONET approach, it is important to identify the correct algorithm or collection of algorithms that will perform the best. It has been proposed in previous chapters that the PIGEON algorithm, which is a combination of NEAT and
PSO, would be the most appropriate choice. To support this claim, a series of experiments were performed to compare PIGEON versus NEAT and PSO individually. To make the comparison fair, each of the algorithms was tuned to identify which combinations of internal parameters perform best for a given domain. Since NEAT and PSO are subcomponents of the PIGEON algorithm, it is advantageous to optimize those algorithms in order to optimize PIGEON itself. Finally, because PIGEON is a new algorithm, there are parameters to investigate to determine their best combination.

Since the Chaser simulation was the simplest in terms of inputs, outputs, and goals, the run time of Chaser was less than the other simulations. It was therefore used as the primary testbed for early experimentation. While the standard “No Free Lunch” idea states that it would be unfair to compare across domains for different algorithms, some comparison needs to be made to get a general idea of the parameter space. After the parameter testing in Section 6.2.2, each algorithm is run on each of the four domains in Section 6.3. In each simulation, the maximum fitness is scaled to the time, therefore for Chaser the maximum fitness is 30 while for Sheep, Crane, and Car it is 60. For the Observational and Instructional stages, two test subjects, codenamed Orange and Violet, were used to ensure the generality of the algorithm.
6.2.1 Method of Comparison

The main method of comparison in the selection of an algorithm is the fitness. The fitness for an approach can be calculated over the course of many runs to obtain the mean and standard deviation. Using these statistics (instead of simply using the fitness of the best run) one can demonstrate that an algorithm will perform better on average in the future.

In all experimental comparisons, thirty (30) runs of each algorithm were performed and the statistics computed.

Using these statistics, a Student’s t-test (Burford 1968) can be performed with the Null Hypothesis that the means are equal. The Student’s t-distribution is used instead of the Normal distribution because the sample set is not large enough to reflect accurate statistics. In the Student’s t-test, the final result calculated is the p-value which relates to the probability that the mean of the two distributions are the same to a significant level. A small p-value of 0.05 would mean that there is a 95% probability that the mean of one distribution falls outside the other distribution. This level of significance would mean that one algorithm is statistically different than another algorithm.
\[ t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s^2_1}{n_1} + \frac{s^2_2}{n_2}}} \]

\[ \nu = \left( \frac{\frac{s^2_1}{n_1} + \frac{s^2_2}{n_2}}{n_1 - 1} + \frac{\frac{s^2_2}{n_2}}{n_2 - 1} \right)^2 \]

\[ \Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt \]

\[ t_{pdf}(t, \nu) = \frac{\Gamma \left( \frac{\nu+1}{2} \right)}{\sqrt{\nu\pi} \ \Gamma \left( \frac{\nu}{2} \right)} \left( 1 + \frac{t^2}{\nu} \right)^{-\left( \frac{\nu+1}{2} \right)} \]

\[ p_{value} = \int_t^\infty t_{pdf}(\tau, \nu)d\tau \]

\[ Percent = (1 - p_{value}) \times 100 \]

The mathematical treatment for calculating p-value from the Student’s t-test is given above. The \( \bar{X}_1 \) and \( \bar{X}_2 \) are the mean fitnesses, \( s_1 \) and \( s_2 \) are the standard deviations, \( n_1 \) and \( n_2 \) are the number of runs, and \( t \) and \( \nu \) are the test statistic and degrees of freedom respectively.

To read any tables later in Chapters 6 and except the summaries in Chapter 7, the tables will have the mean and standard deviation of the fitness of the 30 runs along with a p-value. The later stages of Chapter 7 will be the statistics on 100 runs. The p-value column is the main value to note when determining significance. Each group of experiments, usually assigned its own graph, is given part of a table. If there is no p-value listed in a row, it means that that method was the best performing for the group. If there is a p-value, it is the related to the probability of that row being the same as the best performing. A value less
than 0.05 would mean it was statistically different to a 95% confidence and, by definition, worse than the best. A value above 0.05 would indicate that there is not enough difference to say that the two methods are different at a 95% confidence.

6.2.2 Algorithm-Specific Parameters

Each algorithm has adjustable parameters that affect the performance. Some parameters can improve performance with an additional time cost, while others can be problem specific. Many of the parameters used were obtained from papers about the baseline algorithm that suggest particular values. In those cases, the default value is used and may have been tested superficially. In cases where the default was seen to be significantly worse, an in-depth analysis is done.

6.2.2.1 NEAT

Most of the default parameters were used for the NEAT algorithm. The population size was fixed to 150 individuals, the target number of species was set to 10, the total number of generations to 1000, and species stagnation to 50 generations. Furthermore, fixed mutation rates were set for add neuron (0.1%), add dendrite (0.5%), cold mutation (5%), and hot mutation (20%). Many of these constants were based on (Stanley 2004), although some were found by informal experimentation on previous unpublished works and test runs during
development. The number of generations was fixed such that a typical full run would take approximately one day on a 2.0 Ghz AMD Opteron.

### 6.2.2.2 PSO

The default values were tried for the PSO algorithm. However, several parameters of the PSO/MLP hybrid had to be tested. Because the testing time of a single evaluation of NEAT and PSO took similar time periods, the number of PSO iterations was also fixed to 1000. The main terms for PSO are the inertial constant $\omega$ and the convergence parameters ($c_1$ and $c_2$). It is common practice to select $\omega$ to be 0.9 and to have $c_1 = c_2 = 2.0$. However, through a typographical error when programming the system, it was found that $c_1 = c_2 = 0.9$ performed better.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Constant</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiential</td>
<td>0.9</td>
<td>28.09</td>
<td>0.13</td>
<td></td>
</tr>
<tr>
<td>Experiential</td>
<td>2</td>
<td>25.51</td>
<td>1.12</td>
<td>9.75e-14</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>0.9</td>
<td>22.18</td>
<td>0.16</td>
<td></td>
</tr>
<tr>
<td>Observational Violet</td>
<td>2</td>
<td>18.41</td>
<td>0.28</td>
<td>2.37e-47</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>0.9</td>
<td>24.46</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td>Observational Orange</td>
<td>2</td>
<td>20.54</td>
<td>0.23</td>
<td>4.8e-48</td>
</tr>
</tbody>
</table>

Table 6.1: Comparison of Different PSO Constants

As seen in Figure 6.6, Figure 6.7, and Figure 6.8, the PSO performed better with $c_1 = c_2 = 0.9$ on the Chaser simulation for both the Experiential and Observational stages. And according to Table 6.1, the p-values of the single tail test showed that 0.9 is signifi-
Figure 6.6: Chaser Simulation Experiential, PSO Constants

Figure 6.7: Chaser Simulation Observational Violet, PSO Constants
Figure 6.8: Chaser Simulation Observational Orange, PSO Constants

significantly better with all values being much less than 0.01 (1E-2). One possible explanation of this effect is that the normal PSO values with the default structure can be overly disruptive to the weights. The weight changes by PSO are like the Hot-Mutations, in that the social vector adds to the current value. With the value 2.0, the weights can receive a much higher speed for the particle and move past other optimal solutions. While hard to see in Figure 6.6, PSO finds a solution above 27.00 within the first ten (10) iterations. However, after that initial fast learning, the system decreases in performance. A slower speed from the $c_1=c_2=0.9$ seems to be a less destabilizing. The average best performance learns quickly and continues to increase in performance. Other informal tests were done that show that values greater than 1.0 start decreasing performance. However, no tests were done to determine whether 0.9 was the optimal value less than 1.0.
Although not the point of discussion in this section, it can also be seen from Figure 6.9, that the PSO constants affect later PIGEON performance. For the Alternate (left), there is no significant change; however, in Chain (right) the 2.0 constant decreased the fitness when the PSO was turned on versus increasing performance with 0.9. Therefore, the PSO constant of 0.9 was chosen for the best comparison.

Another point to analyze with PSO is the structure on which the weights are being optimized. Remember PSO itself has no structure. Its structure comes from the ANN. However, since no NEAT step is being performed in order to find the optimal structure, the structure of the ANN has to be assumed for the PSO only tests. Using best practices for input and output, such as in (Khaw et al. 1995), the number of nodes for the hidden layer can be computed. However, the connections themselves are not fixed and there are several options. It was found, but not shown here, that Recurrent Networks outperformed Feed-Forward-only networks. But, there were multiple options in choosing the structure for recurrent networks. Several example networks were created of where to connect the recurrent
loops. The networks tested included: 1) a fully connected network where all hidden layers connect to all other hidden layers and all outputs loop to other outputs called Loop All; 2) only self-loops on the output layer called Loop Output; and 3) only self-loops on the hidden layer called Loop Single. There are a multitude of potential networks from which to select and the proper structure would generally be unknown. That is the purpose of having NEAT complexify the proper structure.

![Fitness Statistics for Algorithm versus Iterations](image)

Figure 6.10: Chaser Simulation Experiential, PSO Structure

According to Table 6.2, the structure does play some importance in the fitness. For the Experiential approach, there is at least a 95% (p-value < .05) probability that Loop Output is better than Loop ALL, but not (p-value > .05) for Loop Single. While for the Observational approach, Loop Single performs best and is different to significant levels for both test subjects. These results would suggest that more connected links in the network do
Figure 6.11: Chaser Simulation Observational Violet, PSO Structure

Figure 6.12: Chaser Simulation Observational Orange, PSO Structure
<table>
<thead>
<tr>
<th>Approach</th>
<th>Structure</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiential</td>
<td>Loop All</td>
<td>28.09</td>
<td>0.13</td>
<td>6.62e-05</td>
</tr>
<tr>
<td>Experiential</td>
<td>Loop Output</td>
<td>28.21</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td>Experiential</td>
<td>Loop Single</td>
<td>28.09</td>
<td>0.37</td>
<td>0.0503</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>Loop All</td>
<td>22.18</td>
<td>0.16</td>
<td>0.000371</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>Loop Output</td>
<td>21.96</td>
<td>0.19</td>
<td>2.35e-11</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>Loop Single</td>
<td>22.31</td>
<td>0.12</td>
<td></td>
</tr>
<tr>
<td>Observational Orange</td>
<td>Loop All</td>
<td>24.46</td>
<td>0.11</td>
<td>0.00215</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>Loop Output</td>
<td>24.27</td>
<td>0.11</td>
<td>1.93e-09</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>Loop Single</td>
<td>24.59</td>
<td>0.21</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.2: Comparison of Different PSO Structure

not necessarily mean better performance. Although, it must be noted that as the number of links increase (66 for Loop All versus 32 and 36 for Loop Out and Loop Single respectively) the number of computations increases and, therefore, the run-time increases approximately linearly to approximately double the time.

Although all the PSO algorithms were similar in performance, Loop Single was technically better in both observation test cases and not technically worse in the Experiential test case. Therefore, for the PSO algorithm, it was decided to use $c_1=c_2=0.9$ for the constants and to use the Loop Single structure when comparing to other algorithms.

### 6.2.2.3 PIGEON

As mentioned in Section 5.2.3, there were two main theories on how to implement the NEAT/PSO combination referred to as PIGEON. The alternatives were either to execute
the NEAT algorithm for a set number of generations and then let PSO optimize the weights of the grown network by chaining them together, or to apply NEAT and PSO alternatively allowing the system to both grow and modify weights at the same time.

The Chain algorithm was originally selected to be the main way PIGEON would operate. The theory was that NEAT would find the optimal structure, then PSO would optimize the weights of that structure. However, since the algorithm is new, it is unknown how long to run each algorithm. Since the Generation of NEAT and the Iteration of PSO were set to be equal, the total number of Generations/Iterations was fixed to 1000 to make the algorithm comparable to both NEAT and PSO alone. Experiments were done to test when the cutoff point should be and therefore, what percentage of time each algorithm should be used. A cutoff percentage of how much to run each algorithm from 25% to 75% was used to identify how much of the time NEAT should be applied versus PSO.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Cutoff</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiential</td>
<td>25%</td>
<td>28.06</td>
<td>0.07</td>
<td>0.416</td>
</tr>
<tr>
<td>Experiential</td>
<td>50%</td>
<td>28.07</td>
<td>0.09</td>
<td></td>
</tr>
<tr>
<td>Experiential</td>
<td>75%</td>
<td>28.05</td>
<td>0.06</td>
<td>0.209</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>25%</td>
<td>21.51</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td>Observational Violet</td>
<td>50%</td>
<td>21.48</td>
<td>0.27</td>
<td>0.292</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>75%</td>
<td>21.48</td>
<td>0.25</td>
<td>0.275</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>25%</td>
<td>24.13</td>
<td>0.17</td>
<td></td>
</tr>
<tr>
<td>Observational Orange</td>
<td>50%</td>
<td>24.12</td>
<td>0.19</td>
<td>0.39</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>75%</td>
<td>24.05</td>
<td>0.14</td>
<td>0.0201</td>
</tr>
</tbody>
</table>

Table 6.3: Comparison of Different Chain Cutoff Percentages

It can be seen in Figure 6.13, that when the PSO algorithm was applied at the cutoff point there was a sudden improvement in performance of the overall algorithm. However,
Figure 6.13: Chaser Simulation Experiential, Chain Cutoff

Figure 6.14: Chaser Simulation Observational Violet, Chain Cutoff
by the end of the run, the fitness of each of the algorithms appears to converge. According to the statistics, the 50% cutoff outperformed the others, but not different to statistically significant levels (p-values > 0.05). It is interesting to note that the standard deviation for each fitness is very small ($\sigma < 0.1$), meaning that the algorithms converged to the same fitness each time.

In Figure 6.14 and Figure 6.15, similar results can be seen with the initial PSO addition increasing fitness when applied. However, among these results, the 25% cutoff outperformed the other two, but only sometimes was different to a statistically significant level. Another interesting result for observation is that there is a greater slope increase for the lower cutoff values. The lower the cutoff value, the less NEAT executes, thereby resulting in a less complex network. This simple network is likely to have been optimized fully early on, while
the more complex networks of 75% are more difficult, and still optimizing by the end of the run. Additionally, when compared against Experiential, the environment is more structured, which results in much smoother learning.

For Chain, it appears that a cutoff of 25% or 50% would be the best values. For experimental testing, none were different to a significant level than any other system and for Observational testing, 25% was technically better in some cases, especially for Orange. It is important to note that 0% PSO would degrade to running just NEAT and 100% PSO degrades to just running the PSO algorithm. These comparisons will be shown later in this Section 6.3.

The other PIGEON method was to alternate between using NEAT and PSO as the algorithm. This is done contrary to published literature that specifies to not adjust structure and weights at the same time (Yao 1993). Alternating the algorithms can make sense because of several properties used from NEAT such as speciation and innovation numbers. The speciation still only allows those individuals of similar structure to be optimized through PSO and speciation process also protects new innovations. The innovation numbers allow the PSO to properly match the weight values that are to be combined. The main aspect to be analyzed was how many iterations of PSO to run per Generation of NEAT. This value, labeled handoff, was examined on a range from 1 to 20.

In Figure 6.16, the Alternate algorithm for Experiential learning shows some definite trends. The handoff values of 1 and 2, values using the most NEAT versus PSO, converged to much lower points than the handoff values of 5, 10, and 20, which use less NEAT versus
Figure 6.16: Chaser Simulation Experiential, Alternate Handoff

Figure 6.17: Chaser Simulation Observational Violet, Alternate Handoff
Figure 6.18: Chaser Simulation Observational Orange, Alternate Handoff

<table>
<thead>
<tr>
<th>Approach</th>
<th>Handoff</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiential</td>
<td>1</td>
<td>27.88</td>
<td>0.12</td>
<td>2.09e-14</td>
</tr>
<tr>
<td>Experiential</td>
<td>2</td>
<td>27.84</td>
<td>0.14</td>
<td>1.67e-14</td>
</tr>
<tr>
<td>Experiential</td>
<td>5</td>
<td>28.13</td>
<td>0.10</td>
<td>0.106</td>
</tr>
<tr>
<td>Experiential</td>
<td>10</td>
<td>28.13</td>
<td>0.07</td>
<td>0.0881</td>
</tr>
<tr>
<td>Experiential</td>
<td>20</td>
<td>28.15</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>Observational Violet</td>
<td>1</td>
<td>21.37</td>
<td>0.26</td>
<td>2.47e-11</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>2</td>
<td>21.87</td>
<td>0.20</td>
<td></td>
</tr>
<tr>
<td>Observational Violet</td>
<td>5</td>
<td>21.80</td>
<td>0.29</td>
<td>0.149</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>10</td>
<td>21.74</td>
<td>0.28</td>
<td>0.0226</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>20</td>
<td>21.49</td>
<td>0.30</td>
<td>3.63e-07</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>1</td>
<td>24.02</td>
<td>0.14</td>
<td>1.59e-12</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>2</td>
<td>24.33</td>
<td>0.14</td>
<td></td>
</tr>
<tr>
<td>Observational Orange</td>
<td>5</td>
<td>24.30</td>
<td>0.15</td>
<td>0.183</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>10</td>
<td>24.21</td>
<td>0.18</td>
<td>0.00191</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>20</td>
<td>24.04</td>
<td>0.20</td>
<td>1.44e-08</td>
</tr>
</tbody>
</table>

Table 6.4: Comparison of Different Alternate Handoff Values
PSO. Additionally, the lower used NEAT values seem to converge more quickly. This could be attributed to the complexity of the underlying neural-network. The worst system, handoff 2, averaged 20.2 Neurons and 67.7 Dendrites while the best system, handoff 20, averaged 6.5 Neurons and 11.4 Dendrites. In the handoff 2 case, NEAT could be producing additional complexity beyond what is needed for the problem. However, NEAT alone without PSO averaged 12.5 Neurons and 44.3 Dendrites. The PSO weight changes could then be attributed to the additional growth. In this experiment, handoff 20 was the best performer. It was significantly different than all (p-value < 0.05) except for handoff 5 and handoff 10. Of those two, handoff 5 performed better with a p-value of 0.106 and therefore, not even be considered different to a 90% confidence.

In Figure 6.17 and Figure 6.18, the Alternate testing for Observational learning shows the opposite of Experiential learning. For both Observational test cases, the handoff value of 2 outperforms all and is different to a significant degree (p-value < 0.05) except for handoff 5. For Observational learning, the increased NEAT values seem to function better by learning faster and appearing to continue learning while other values seem to have converged to a lower value with smaller slopes. It is interesting to note that handoff 1 performed the worst in each case. This could be attributed to the fact that with handoff 1 several factors of PSO could not operate appropriately such as momentum and personal best scores. With only one step of PSO before a step of NEAT, the momentum terms and last personal best terms of PSO learning could not be calculated because NEAT likely replaced individuals through
mating. When mating occurs and new individuals are created, there is no history of old values and no past velocity information.

For Alternate, the handoff value of 5 appears to be the best option. While it did not perform the best in either Experiential or Observational, it was, however, not significantly different (p-values of 0.106, 0.149, and 0.183) than the best in any of the cases. It was therefore the most consistent in testing, which is desirable. The handoff values that were best in one case were usually near worst in the other.

6.3 Algorithm Selection per Stage

The end result of optimizing the different algorithms is to select which algorithm did best on the test simulations. In order to make the selection, the top performing parameters for each individual algorithm were graphed and compared to each other for all four of the simulations: Chaser, Sheep, Crane, and Car. The algorithm that performed best for each stage was then used as the main algorithm for all further testing stages. The final algorithms used for comparison are: NEAT with standard values, PSO with $c_1=c_2=0.9$ and Loop Single structure, Chain with 50% Cutoff, and Alternate with Handoff of 5.
6.3.1 Observational Comparison

For Observational learning, a comparison was done across the all four domains for each algorithm. Furthermore, the comparison was done on both test subjects, Violet and Orange. As a preface to the results discussion, the test subjects themselves acted differently in how they approached each simulation. Orange was very consistent in the training by using the same technique throughout. Violet changed behaviors several times during the training, trying to improve performance. This is believed to account for the consistently higher scores for every algorithm on Observational learning based on Orange versus those based on Violet.

6.3.1.1 Chaser Observational

<table>
<thead>
<tr>
<th>Approach</th>
<th>Algorithm</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observational Violet</td>
<td>NEAT</td>
<td>21.33</td>
<td>0.25</td>
<td>8.17e-21</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>PSO</td>
<td>22.18</td>
<td>0.16</td>
<td></td>
</tr>
<tr>
<td>Observational Violet</td>
<td>Chain</td>
<td>21.48</td>
<td>0.27</td>
<td>1.2e-16</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>Alternate</td>
<td>21.80</td>
<td>0.29</td>
<td>5.48e-08</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>NEAT</td>
<td>23.98</td>
<td>0.16</td>
<td>1.04e-18</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>PSO</td>
<td>24.46</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td>Observational Orange</td>
<td>Chain</td>
<td>24.12</td>
<td>0.19</td>
<td>2.37e-11</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>Alternate</td>
<td>24.30</td>
<td>0.15</td>
<td>9.06e-06</td>
</tr>
</tbody>
</table>

Table 6.5: Observational Learning - Comparison of Algorithms on Chaser

In both Figure 6.19 and Figure 6.20, it can be seen that PSO outperformed the other algorithms in the Chaser simulation with Alternate being second. According to Table 6.5, PSO was better and different to a significant level (p-value < 0.05) in each case. For Violet,
Figure 6.19: Chaser Simulation Observational Violet, Algorithm Comparison

Figure 6.20: Chaser Simulation Observational Orange, Algorithm Comparison
PSO learned more quickly and gained an early lead. For Orange, PSO kept an offset above both NEAT and Chain, but Alternate had an accelerated learning event approximately around iteration/generation 300. In both cases, Chain tracks NEAT almost exactly, as it should since the algorithm runs NEAT only, until the cutoff is reached. After the cutoff, a slight improvement can be seen. Examining both Table 6.5 and looking at the raw population statistics, PSO had a lower standard deviation. It is possible that PSO would quickly optimize to a local maximum and become stuck. Finally, PSO also had a much more complex network with a Neuron/Dendrite count of 11/66 versus 13/34 for Alternate. The additional weights create a harder problem. Although during the 1000 generation/iteration run, PSO appears to be the best choice for Observational learning.

For the Chaser experiment for subject Violet, it appears that Alternate still has an upward slope, unlike PSO which seems to have stagnated. A study was done to see these effects long term if Alternate would continue learning, this is discussed in Section 6.3.4.2.

6.3.1.2 Sheep Observational

In Figure 6.21 and Figure 6.22 it can be seen that PSO performs well in both Observational experiments for Sheep. However, similar to Chaser, it is able to learn quickly in the first 20 iterations but then becomes stuck. NEAT and Chain track together and Chain improves dramatically at the Cutoff value of 50% and is able to maintain an offset over NEAT. Again, Alternate is able to learn quickly and perform better than NEAT and Chain. However, by
Figure 6.21: Sheep Simulation Observational Violet, Algorithm Comparison

Figure 6.22: Sheep Simulation Observational Orange, Algorithm Comparison
<table>
<thead>
<tr>
<th>Approach</th>
<th>Algorithm</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observational Violet</td>
<td>NEAT</td>
<td>47.26</td>
<td>0.46</td>
<td>1.02e-27</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>PSO</td>
<td>49.84</td>
<td>0.19</td>
<td></td>
</tr>
<tr>
<td>Observational Violet</td>
<td>Chain</td>
<td>47.89</td>
<td>0.06</td>
<td>1.18e-35</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>Alternate</td>
<td>48.11</td>
<td>0.37</td>
<td>4.05e-26</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>NEAT</td>
<td>51.41</td>
<td>0.52</td>
<td>1.71e-14</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>PSO</td>
<td>52.65</td>
<td>0.14</td>
<td></td>
</tr>
<tr>
<td>Observational Orange</td>
<td>Chain</td>
<td>52.18</td>
<td>0.06</td>
<td>1.41e-19</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>Alternate</td>
<td>52.32</td>
<td>0.17</td>
<td>1.66e-11</td>
</tr>
</tbody>
</table>

Table 6.6: Observational Learning - Comparison of Algorithms on Sheep

the end of the run, Alternate and Chain were at similar levels. According to Table 6.6, PSO outperformed the other algorithms and was different to a significant level for both test subjects, Violet and Orange.

6.3.1.3 Crane Observational

<table>
<thead>
<tr>
<th>Approach</th>
<th>Algorithm</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observational Violet</td>
<td>NEAT</td>
<td>40.86</td>
<td>0.80</td>
<td>6.03e-17</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>PSO</td>
<td>44.59</td>
<td>1.40</td>
<td></td>
</tr>
<tr>
<td>Observational Violet</td>
<td>Chain</td>
<td>42.71</td>
<td>0.79</td>
<td>3.57e-08</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>Alternate</td>
<td>43.79</td>
<td>1.52</td>
<td>0.0196</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>NEAT</td>
<td>38.57</td>
<td>0.67</td>
<td>1.43e-19</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>PSO</td>
<td>45.01</td>
<td>1.89</td>
<td></td>
</tr>
<tr>
<td>Observational Orange</td>
<td>Chain</td>
<td>40.43</td>
<td>0.69</td>
<td>5.02e-15</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>Alternate</td>
<td>42.29</td>
<td>1.31</td>
<td>1.79e-08</td>
</tr>
</tbody>
</table>

Table 6.7: Observational Learning - Comparison of Algorithms on Crane
Figure 6.23: Crane Simulation Observational Violet, Algorithm Comparison

Figure 6.24: Crane Simulation Observational Orange, Algorithm Comparison
In Figure 6.23 and Figure 6.24, the various attributes of the different algorithms are seemingly magnified. PSO can be clearly seen to learn quickly and obtain a fitness of near 40 almost instantaneously, although it is interesting that the PSO values are noisy for the first 500 iterations. Chain gets an instant performance boost over NEAT at the cutoff value and maintains a strong lead. Alternate has shown that it is able to learn much more quickly than the other NEAT-based algorithms, even passing PSO for subject Violet. According to Table 6.7, PSO finished significantly different and better on average with Alternate being second and close to at least a 99% confidence for Violet (p-value > 0.01). The Crane simulation had more actions to match than the other simulations but observationally, the system was able to match them to a similar score to the other simulations.

### 6.3.1.4 Car Observational

<table>
<thead>
<tr>
<th>Approach</th>
<th>Algorithm</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observational Violet</td>
<td>NEAT</td>
<td>50.55</td>
<td>0.39</td>
<td>3.17e-35</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>PSO</td>
<td>53.25</td>
<td>0.26</td>
<td></td>
</tr>
<tr>
<td>Observational Violet</td>
<td>Chain</td>
<td>51.57</td>
<td>0.50</td>
<td>1.79e-20</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>Alternate</td>
<td>52.84</td>
<td>0.77</td>
<td>0.0047</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>NEAT</td>
<td>51.30</td>
<td>0.30</td>
<td>1.65e-42</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>PSO</td>
<td>54.11</td>
<td>0.27</td>
<td></td>
</tr>
<tr>
<td>Observational Orange</td>
<td>Chain</td>
<td>51.98</td>
<td>0.46</td>
<td>1.41e-26</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>Alternate</td>
<td>53.14</td>
<td>0.74</td>
<td>3.82e-08</td>
</tr>
</tbody>
</table>

Table 6.8: Observational Learning - Comparison of Algorithms on Car
Figure 6.25: Car Simulation Observational Violet, Algorithm Comparison

Figure 6.26: Car Simulation Observational Orange, Algorithm Comparison
In Figure 6.25 and Figure 6.26, the trends continue similar to the other simulations for observation. PSO outperformed the other three algorithms and was different to a significant level, PSO learns early and levels off; Chain gains a lead on NEAT at the cutoff point and Alternate is able to learn faster.

6.3.1.5 Observational Summary

Over the course of four different simulation domains, the trends were readily seen for Observational learning. PSO would initially learn very quickly, optimizing its weights using a much larger, fully connected network. However, it would learn to a point, over-fit the values, and appeared to not have a steep slope and stagnated. This over-fitting will relate later to Section 6.3.4.1. With Chain, the PSO’s quick learning ability is able to give an initial quick boost to the performance over NEAT by itself. The weights of the network get optimized quickly at the cutoff point and end with a higher fitness than NEAT. However, NEAT could pass Chain if more generations were run because it seemed to have a slightly upward slope in some simulations. Alternate seemed to have the best combination of PSO and NEAT. Alternate would learn quickly and not stagnate fully because when the next NEAT generation happens after the handoff every five iterations. This is believed to happen because of NEAT’s ability to switch the problem space by complexifying the problem that PSO might have originally over-fit. The moving target actually seems to improve performance. In general, PSO seems to be the algorithmic method of choice for Observational learning. It was
able to outperform the other learning algorithms and be statistically different to a significant level for all 8 experiments. However, because the purpose of this dissertation is to produce well performing agents from observation, and not match the human inputs like an advisory system, the experiential score needs to be examined for those agents created through observation. The results of these experiments are in Section 6.3.4.1. A possible extension to this section would be to increase the number of generations/iteration to take advantage of the continued learning, this is investigated further in Section 6.3.4.2.

### 6.3.2 Instructional Comparison

The reader should remember that Instructional learning takes place on-line, in real-time, as an interaction between the trainer and the agent. It is a cooperative learning effort where the trainer must teach and fix the agent by adapting what it is doing. This is done through the haptic feedback element of the current best agent. As part of the learning approach, the agents are continually being given learning rewards and penalties by the trainer. The agents are sorted based on these scores, and the best agent is at the top of the list. However, there are competing ideas for implementation to show the benefits of Instructional learning. Should all the reward be given at the end of a training session in a batch fashion, or should the reward be distributed during the run? Giving the reward at the end is very similar to an interactive Observational learning. The summed reward could be considered the fitness of the individual. While reward-during is the definition of Instructional learning because it
involves real-time feedback to the algorithm for adaptation. Regardless of the method of giving rewards, the agents are trained using the PIEGON algorithm with a fixed network during the run and weights adjusted using PSO. The approach is explained in Section 5.3.2.

The reward, re-sorting, and algorithm iteration happen once a second. The comparison of these approaches was done for three runs each consisting of 20 training session for agents learning from an initial random population. Because training must happen in real-time and each run would take 20 minutes, the training was limited to only three runs. Furthermore, with two different methods and four different domains, it would take a minimum of eight continuous hours of human time to train everything. The Student’s t-test, given in Section 6.2.1, reflect the smaller number of samples by recalculating the degrees of freedom. This still provided a fair way to calculate significance.

### 6.3.2.1 Chaser Instructional

For the Chaser Instructional training, the agent would initially perform rather randomly. This caused the trainer to almost continually penalize the agents. By the 12th training iteration, the agents would begin to perform rationally. The trainer could therefore allow the agent to operate normally, and only intercede when necessary.

In Figure 6.27, it can be seen that the reward during the training appears to give better fitness for the population. Additionally in Table 6.9, the reward during approach seems to outperform the reward at the end approach to a significant level and have a lower standard
Figure 6.27: Chaser Simulation Instructional Green, Two Methods

<table>
<thead>
<tr>
<th>Approach</th>
<th>Method</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instructional Green</td>
<td>Reward At End</td>
<td>22.90</td>
<td>2.89</td>
<td>0.0302</td>
</tr>
<tr>
<td>Instructional Green</td>
<td>Reward During</td>
<td>28.52</td>
<td>1.31</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.9: Instructional Learning - Comparison of Two Methods on Chaser
deviation. The fitness appears to improve over the 20 training sessions. The agents trained with reward during the run could eventually follow the fleer while those trained with reward at the end were not always able to accomplish this feat.

6.3.2.2 Sheep Instructional

![Fitness Statistics for Algorithm versus Generations/Iterations](image)

Figure 6.28: Sheep Simulation Instructional Green, Two Methods

<table>
<thead>
<tr>
<th>Approach</th>
<th>Method</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instructional Green</td>
<td>Reward At End</td>
<td>33.75</td>
<td>1.35</td>
<td>0.000104</td>
</tr>
<tr>
<td>Instructional Green</td>
<td>Reward During</td>
<td>50.46</td>
<td>1.64</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.10: Instructional Learning - Comparison of Two Methods on Sheep
For the Sheep simulation shown in Figure 6.28, the reward-during runs significantly outperforms the reward-at-end. The reward-during run continues to have a smoother curve with an upward trend. For the Sheep simulation, agents of both training methods did not seem to totally grasp the concept, usually not chasing the sheep directly. This can be understandable since the domain is more complex than Chaser and the training is being done on agents with originally random weights. The agents produced from reward during the run did act less erratically than agents produced from reward at the end of the run. The reward-at-end agents would typically move back and forth severely when it would get close to sheep. This type of action would be undesirable if attempting to appear intelligent.

### 6.3.2.3 Crane Instructional

<table>
<thead>
<tr>
<th>Approach</th>
<th>Method</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instructional Green</td>
<td>Reward At End</td>
<td>31.64</td>
<td>1.26</td>
<td>0.00121</td>
</tr>
<tr>
<td>Instructional Green</td>
<td>Reward During</td>
<td>38.34</td>
<td>0.97</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.11: Instructional Learning - Comparison of Two Methods on Crane

It can be seen in Figure 6.29, that the results are much closer in the Crane simulation than with Sheep. Nevertheless, according to Table 6.11, the agents produced with reward-during-run were still able to significantly outperform agents produced with reward-at-end. Both agent sets required considerable effort to train in this domain. The agents behaved rather poorly, causing the trainer to almost continually provide negative reinforcement. Ad-
ditionally, the agent’s actions, felt through the force-feedback, made it difficult for the trainer to perform the task. However, the reward during set of agents eventually learned to move toward the closest block and to return to base once the object was picked up. It did not, however, learn how to pick up the block or drop the block once it returned to base. The reward at the end set of agents were unable to find a path to the block and would constantly “vibrate” as it would want to go full left or right rapidly.

6.3.2.4 Car Instructional

In Figure 6.30, it appears that agents trained with reward during operation outperform agent trained with reward at the end by a good margin. However in Table 6.12, it is revealed that it
Figure 6.30: Car Simulation Instructional Green, Two Methods

<table>
<thead>
<tr>
<th>Approach</th>
<th>Method</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instructional Green</td>
<td>Reward At End</td>
<td>32.26</td>
<td>9.97</td>
<td>0.0409</td>
</tr>
<tr>
<td>Instructional Green</td>
<td>Reward During</td>
<td>50.84</td>
<td>1.35</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.12: Instructional Learning - Comparison of Two Methods on Car
is barely a significant different (p-value < 0.05) because of the very high standard deviation of the reward-at-end approach. In the complex domain of driving a car, the agents produced with reward-during-run eventually learned to stay in the lane after the trainer brought the agent to the center of the lane. They were unable to deal with traffic well, both when merging into the center lane and when the cars were in the same lane. The agents produced with reward at the end were not even able to learn to stay in the lane.

6.3.2.5 Instructional Summary

In general, for Instructional learning, the agents trained with reward-during-run operation approach seemed to always outperform the agents produced with reward-at-end, and this was true to a significant level in all four simulation domains. These results lend credence to the Instructional learning approach by showing its benefits over an approach more similar to observation learning. One of the main reasons for this is believed to be the number of algorithm updates to the population of individuals. Even though the individuals are evaluated on every time step, the reward-during receives many more PSO iterations for training the network. Effectively, the reward-at-end only does one algorithm iteration per training session. Each domain received 20 training session, thus PSO only received 20 iterations. For reward-during-run, a iteration would happen once a second during the run for the 20 iterations. It therefore received 600 iterations for the 30 second Chaser runs and 1200 iterations for the 60 seconds of the other domains. These additional iterations allowed each of the
“monkeys on the bus” (Spector and Klein 2002) to jostle for position more often since the sorting for position happens at the time of the iteration. This allows them to influence each other more during the PSO update. For these reasons, reward-during-run was selected as the method of choice for Instructional learning.

6.3.3 Experiential Comparison

Experiential learning operates by allowing an agent to interact with an environment and scoring the agent purely based on performance. This score directly becomes the fitness of an individual. The goal of the system is to optimize the fitness score for a given simulation. In this set of experiments, all four algorithms are compared in the four simulation scenarios. In each case the algorithms were given 1,000 generations/iterations of learning steps and every agent in the population was presented with eight random training scenarios.

6.3.3.1 Chaser Experiential

<table>
<thead>
<tr>
<th>Approach</th>
<th>Algorithm</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiential</td>
<td>NEAT</td>
<td>27.92</td>
<td>0.09</td>
<td>7.13e-12</td>
</tr>
<tr>
<td>Experiential</td>
<td>PSO</td>
<td>28.09</td>
<td>0.13</td>
<td>0.0918</td>
</tr>
<tr>
<td>Experiential</td>
<td>Chain</td>
<td>28.07</td>
<td>0.09</td>
<td>0.00968</td>
</tr>
<tr>
<td>Experiential</td>
<td>Alternate</td>
<td>28.13</td>
<td>0.10</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.13: Experiential Learning - Comparison of Algorithms on Chaser
In Figure 6.31, it can be seen that all four algorithms were able to do well on the simple Chaser simulation, receiving a score of approximately 28 out of 30. The fitness trends found in Experiential training are similar to Observational learning in several respects. Chain and NEAT are nearly the same until the cutoff point where Chain gains a quick lead and is able to maintain the lead through the rest of the run. However, for Experiential learning, Alternate learns more quickly and performs the best, even more so than PSO. But no additional learning appears to take place after the first 300 generations/iterations. According to Table 6.13, Alternate is significantly different and outperforms the other algorithms (p-value < 0.05) except PSO. The performance of each is rather close and learned quickly by all. This can be likely attributed to the simple nature of the experiment.
6.3.3.2 Sheep Experiential

Figure 6.32: Sheep Simulation Experiential, Algorithm Comparison

<table>
<thead>
<tr>
<th>Approach</th>
<th>Algorithm</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiential</td>
<td>NEAT</td>
<td>59.45</td>
<td>0.86</td>
<td></td>
</tr>
<tr>
<td>Experiential</td>
<td>PSO</td>
<td>54.82</td>
<td>4.10</td>
<td>4.9e-07</td>
</tr>
<tr>
<td>Experiential</td>
<td>Chain</td>
<td>59.17</td>
<td>0.94</td>
<td>0.118</td>
</tr>
<tr>
<td>Experiential</td>
<td>Alternate</td>
<td>58.73</td>
<td>1.56</td>
<td>0.0158</td>
</tr>
</tbody>
</table>

Table 6.14: Experiential Learning - Comparison of Algorithms on Sheep

In Figure 6.31, the graph of the fitness is rather interesting because it is different than some of the other experiments. In the Sheep simulation, PSO is unable to optimize as well as the other algorithms, however still receives a fitness of 54.82 out of 60. This score equates to catching approximately 14 of the 16 sheep on average. The other three algorithms with a fitness of almost 60 means that they generally caught all 16 sheep, thereby effectively
solving the problem. In this simulation, Alternate was able to learn more quickly in the first 30 generations/iterations, but appears to fall off early. NEAT did the best with Chain tracking almost the entire way except for the last 50 generations/iterations. Chain received only a small boost at the cutoff, however, perhaps because PSO did so poorly. According to Table 6.14, NEAT technically was significantly different and better than the other algorithms (p-value < 0.05) except Chain and it was very close in this case to the other NEAT-based algorithms.

6.3.3.3 Crane Experiential

![Figure 6.33: Crane Simulation Experiential, Algorithm Comparison](image)
<table>
<thead>
<tr>
<th>Approach</th>
<th>Algorithm</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiential</td>
<td>NEAT</td>
<td>3.96</td>
<td>0.05</td>
<td>0.00558</td>
</tr>
<tr>
<td>Experiential</td>
<td>PSO</td>
<td>3.97</td>
<td>0.04</td>
<td>0.00875</td>
</tr>
<tr>
<td>Experiential</td>
<td>Chain</td>
<td>4.00</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>Experiential</td>
<td>Alternate</td>
<td>3.97</td>
<td>0.06</td>
<td>0.0253</td>
</tr>
</tbody>
</table>

Table 6.15: Experiential Learning - Comparison of Algorithms on Crane

The Crane results in Figure 6.33 are unfortunate because no algorithm was experientially able to accomplish the task. The results cannot be properly analyzed because no algorithm was able to pick up a single block. This could be blamed on the complexity of the domain. The other testing domains are very reactive in nature and a single agent without different modes can be used to accomplish the task. In Crane, there are really multiple states and different sets of actions that need to be performed. The algorithms could have benefited by dividing the domain into different contexts. The contextual idea needed for Crane is presented in Section 8.2.1 as part of future work.

### 6.3.3.4 Car Experiential

<table>
<thead>
<tr>
<th>Approach</th>
<th>Algorithm</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiential</td>
<td>NEAT</td>
<td>26.50</td>
<td>9.30</td>
<td>0.0212</td>
</tr>
<tr>
<td>Experiential</td>
<td>PSO</td>
<td>31.22</td>
<td>12.37</td>
<td>0.339</td>
</tr>
<tr>
<td>Experiential</td>
<td>Chain</td>
<td>29.45</td>
<td>12.33</td>
<td>0.171</td>
</tr>
<tr>
<td>Experiential</td>
<td>Alternate</td>
<td>32.61</td>
<td>13.09</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.16: Experiential Learning - Comparison of Algorithms on Car
In Figure 6.34, the results are more promising in the complex domain of Car. Each of the algorithms was able to drive the car for an average of 2.5 to 3.0 laps. In this experiment, PSO was the slowest learner, but does continue to improve over the course of the run. NEAT and Chain, again, do the same on average until the cutoff point of 50% at generation 500 when the PSO part takes over. It is interesting to note that in this domain, Chain shows signs of its typical stagnation, but experiences another learning step at iteration 800. NEAT is unable to pass Chain by the end of the run. Finally, Alternate outperforms all the other algorithms, but is not significantly different as seen in Table 6.16. The large standard deviation of the fitness scores made it hard to prove significance because of the large overlap of the fitness distributions. Alternate learns the quickest, learning up to a 28 fitness in the first 100 generations/iterations. It also appears to continue to learn over the course of the
generations/iterations and could continue to learn beyond that point because of the upward trend. This would make sense because when compared to the other domains, it would be a more difficult task to learn and therefore should take longer to find the optimal solution.

6.3.3.5 Experiential Summary

For Experiential learning, the results show that the PIGEON Alternate learning algorithm is better for both Chaser and Car, while slightly worse than NEAT on Sheep. No results could be gained from Crane, as no algorithm was able to find an acceptable solution. While there does not appear to be a clear winner for every domain, the Alternate method was consistently near the top or at the top in most of the testing. The properties of Alternate also appear to be well-suited to Experiential learning. Therefore, Alternate is the algorithm of choice for Experiential learning.

6.3.4 Additional Experimental Testing

6.3.4.1 Algorithm Experiential Performance

The previous section in this chapter focused on the scores received during each respective phase of learning (Observational, Instructional, and Experiential) for the purpose of algorithm selection. Another factor that must be analyzed is the performance of the agent
produced with that phase once left on its own to operate in the domain. The main example of this difference is related to Observational learning. A high score in Observational learning means that the agent has best matched those actions of the human test subject on the observed cases but does not mean the system has generalized to other cases. A produced agent must be able to intelligently operate on its own.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Subject</th>
<th>Method</th>
<th>Observation</th>
<th>Exp Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chaser</td>
<td>Violet</td>
<td>PSO</td>
<td>22.18</td>
<td>26.18</td>
<td>0.39</td>
<td>5.05e-14</td>
</tr>
<tr>
<td>Chaser</td>
<td>Violet</td>
<td>Alternate</td>
<td>21.80</td>
<td>27.10</td>
<td>0.34</td>
<td></td>
</tr>
<tr>
<td>Chaser</td>
<td>Orange</td>
<td>PSO</td>
<td>24.46</td>
<td>26.75</td>
<td>0.35</td>
<td></td>
</tr>
<tr>
<td>Chaser</td>
<td>Orange</td>
<td>Alternate</td>
<td>24.30</td>
<td>26.65</td>
<td>0.20</td>
<td>0.0912</td>
</tr>
<tr>
<td>Sheep</td>
<td>Violet</td>
<td>PSO</td>
<td>49.84</td>
<td>47.87</td>
<td>9.28</td>
<td></td>
</tr>
<tr>
<td>Sheep</td>
<td>Violet</td>
<td>Alternate</td>
<td>48.11</td>
<td>47.33</td>
<td>12.81</td>
<td>0.428</td>
</tr>
<tr>
<td>Sheep</td>
<td>Orange</td>
<td>PSO</td>
<td>52.65</td>
<td>44.45</td>
<td>17.98</td>
<td></td>
</tr>
<tr>
<td>Sheep</td>
<td>Orange</td>
<td>Alternate</td>
<td>52.32</td>
<td>41.40</td>
<td>19.49</td>
<td>0.267</td>
</tr>
<tr>
<td>Crane</td>
<td>Violet</td>
<td>PSO</td>
<td>44.59</td>
<td>3.87</td>
<td>0.36</td>
<td>0.32</td>
</tr>
<tr>
<td>Crane</td>
<td>Violet</td>
<td>Alternate</td>
<td>43.79</td>
<td>3.91</td>
<td>0.29</td>
<td></td>
</tr>
<tr>
<td>Crane</td>
<td>Orange</td>
<td>PSO</td>
<td>45.01</td>
<td>3.91</td>
<td>0.29</td>
<td></td>
</tr>
<tr>
<td>Crane</td>
<td>Orange</td>
<td>Alternate</td>
<td>42.29</td>
<td>3.87</td>
<td>0.21</td>
<td>0.273</td>
</tr>
<tr>
<td>Car</td>
<td>Violet</td>
<td>PSO</td>
<td>53.25</td>
<td>12.77</td>
<td>18.80</td>
<td>0.000599</td>
</tr>
<tr>
<td>Car</td>
<td>Violet</td>
<td>Alternate</td>
<td>52.84</td>
<td>30.35</td>
<td>21.02</td>
<td></td>
</tr>
<tr>
<td>Car</td>
<td>Orange</td>
<td>PSO</td>
<td>54.11</td>
<td>5.60</td>
<td>12.23</td>
<td>2.74e-06</td>
</tr>
<tr>
<td>Car</td>
<td>Orange</td>
<td>Alternate</td>
<td>53.14</td>
<td>30.56</td>
<td>23.39</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.17: Compare Observational on Experiential Fitness PSO versus Alternate

Table 6.17 represents a full layout of observation fitness and the experiential fitness (Exp Mean) for the best agents produced through Observational learning for both the PSO and Alternate algorithms. In the chart it can be seen that PSO has outperformed Alternate in every case for Observational learning. However, PSO only outperforms Alternate experientially in three cases (Chaser Orange and Both Sheep) but is not different to significant
levels (p-value > 0.05). Alternate does outperform PSO in three other cases (Chaser Violet and Both Car) and is different to significant levels (p-value < 0.05). In the Car domain especially, Alternate does much better, scoring double the value of PSO. This table supports the Alternate would be a better choice than PSO when producing an agent possibly because of the way in which PSO might over-fit the observational data.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Method</th>
<th>Neurons</th>
<th>Dendrites</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chaser</td>
<td>PSO</td>
<td>11</td>
<td>68</td>
</tr>
<tr>
<td>Chaser</td>
<td>Alternate</td>
<td>12</td>
<td>29.5</td>
</tr>
<tr>
<td>Sheep</td>
<td>PSO</td>
<td>38</td>
<td>912</td>
</tr>
<tr>
<td>Sheep</td>
<td>Alternate</td>
<td>19</td>
<td>43</td>
</tr>
<tr>
<td>Crane</td>
<td>PSO</td>
<td>69</td>
<td>3588</td>
</tr>
<tr>
<td>Crane</td>
<td>Alternate</td>
<td>22</td>
<td>69.5</td>
</tr>
<tr>
<td>Car</td>
<td>PSO</td>
<td>38</td>
<td>912</td>
</tr>
<tr>
<td>Car</td>
<td>Alternate</td>
<td>17.5</td>
<td>37.5</td>
</tr>
</tbody>
</table>

Table 6.18: Size Comparison PSO versus Alternate

Additionally in a subjective observation, the agent produced with PSO was less steady in the actions performed. For example in the Car domain, the Alternate-taught agent would stay in the lane and produce smooth turns while the PSO-taught agent was continually moving back and forth in the lane. While these two motions would receive equal score for staying in the lane, the Alternate agent would externally appear more intelligent. A possible explanation of this behavior is likely explained by the structure of the agent’s internal networks. Table 6.18 gives the average sizes for Alternate and PSO. The optimized network produced through Alternate is about half as small as PSO for Chaser but over a 20\textsuperscript{th} of the size on both Sheep and Car and a 50\textsuperscript{th} of the size for Crane. Since PSO uses a fully
connected network, it has many more neurons and dendrites than necessary. The extraneous weights with feedback can make the network more unstable and difficult to optimize. An additional effect of the large networks relates to performance. The additional structure took extra time to compute. For example on Car, PSO took approximately 6.7 versus 5.8 days for Alternate. Although the majority of the computation time was related to the simulation. The approximately one extra day of computation was required because of the additional network size.

The experiential fitness of the Observationally-taught agent and the network size therefore, show that Alternate would be the better choice for learning from observation. This is an important selection because the method for learning for observation affects later learning phases as will be shown in Chapter 7.

6.3.4.2 Increased Generations

After viewing the results of the Alternate experiments for Observational learning, it could be seen that the results had not converged to a point after 1,000 generations/iterations. It would therefore be interesting to know how many generations/iterations it would take to converge. The Violet test case for Chaser was chosen to be studied because it appeared to have the largest slope out of the observational cases. The experiments were run again for 10,000 generations/iterations - ten times the original number.
Figure 6.35: Chaser Simulation Observational Violet, Alternate Long Run

<table>
<thead>
<tr>
<th>Approach</th>
<th>Handoff</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observational Violet</td>
<td>1</td>
<td>22.27</td>
<td>0.12</td>
<td>2.36e-18</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>2</td>
<td>22.76</td>
<td>0.17</td>
<td></td>
</tr>
<tr>
<td>Observational Violet</td>
<td>5</td>
<td>22.53</td>
<td>0.39</td>
<td>0.00261</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>10</td>
<td>22.45</td>
<td>0.30</td>
<td>6.9e-06</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>20</td>
<td>22.39</td>
<td>0.33</td>
<td>1.49e-06</td>
</tr>
</tbody>
</table>

Table 6.19: Comparison of Different Alternate Handoff Values, Long Term
As seen in Figure 6.35, the Alternate system continues to learn well past the original 1,000 generations/iterations and by 10,000 generations/iterations it appears to still not to have fully converged. The trends are the same as the shorter runs in Section 6.2.2.3 with handoff 2 being best, handoff 1 worst and decreasing performance with additional NEAT generations. This time handoff 2 is significantly different than handoff 5 (p-value < 0.05), however, handoff 5 is the second best. The increased generations/iterations also lead to a longer run-time. While the original 1,000 generations/iterations run was approximately limited to one day for Chaser, it took approximately 12 days to compute 10,000 generations/iterations. This is to be expected because a fixed population size would be approximately linear in generations/iterations, except that the networks became much more complex, which increases the run time. At 1,000, the best network had 12.0 Neurons and 33.3 Dendrites on average while at 10,000 the best network had 50.2 Neurons and 219.1 Dendrites.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Gen/Iter</th>
<th>Diff</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observational Violet</td>
<td>1000</td>
<td>2.13</td>
<td>21.83</td>
<td>0.26</td>
<td>3.69e-11</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>2000</td>
<td>0.22</td>
<td>22.05</td>
<td>0.29</td>
<td>8.09e-07</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>3000</td>
<td>0.17</td>
<td>22.22</td>
<td>0.32</td>
<td>0.000678</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>4000</td>
<td>0.07</td>
<td>22.29</td>
<td>0.35</td>
<td>0.00709</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>5000</td>
<td>0.04</td>
<td>22.33</td>
<td>0.34</td>
<td>0.0191</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>6000</td>
<td>0.03</td>
<td>22.36</td>
<td>0.34</td>
<td>0.038</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>7000</td>
<td>0.05</td>
<td>22.41</td>
<td>0.37</td>
<td>0.12</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>8000</td>
<td>0.04</td>
<td>22.45</td>
<td>0.39</td>
<td>0.226</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>9000</td>
<td>0.03</td>
<td>22.48</td>
<td>0.39</td>
<td>0.311</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>10000</td>
<td>0.05</td>
<td>22.53</td>
<td>0.39</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.20: Comparison of Generations/Iterations Over Time, Long Term
Comparing these values over time condensed in Table 6.20, it can be seen that learning and improvement is still happening well into the 10,000th generation/iteration and the fitness is still increasing at approximately 0.05 points per 1,000 generations/iterations. However, according to the p-value, no significant learning took place after 6,000 generations/iterations (p-values > 0.05). It is obvious that a majority of the learning took place in the first 1,000 generations/iterations. If the sum of the increase in fitness of 2.83 overall is said to be 100% learning, then 75.3% occurred in the first time period with 7.8% in the second, and decreasing from there. With diminishing returns after the first 3,000 generations/iteration, each additional day of training only nets approximately 1.5% more fitness. Through an informal cost-benefit analysis, the added time past 3,000, or three days, is likely unnecessary for the Observational stage. For time constraints, it was decided to not run any additional training past 1,000 generations/iterations for the other domains. However as part of future work, it would be beneficial to increase the Observational learning phase to 2,000 generations/iterations to improve fitness without overly increasing the processing time. Additionally, some thought has been put into a stopping criteria to eliminate a fixed number of generations (see Section 8.1.2).

6.3.4.3 Transfer Between Approaches

When transferring intelligent agents between learning approaches, there were two different approaches implemented, as mentioned in Section 5.3.4. The best-agent-transfer method only
takes the best agent from the previous stage, which for this experiment was Observationally-taught, and seeds the next stage, in this case Experiential learning. The best agent is then slightly modified through mutation to create the new population. The theory is that the best agent is close to the solution and only some additional learning from another phase is needed. The other method was to transfer the entire population that contained the best agent. The theory is that the best agent might be over-fit to Observational learning, but possibly another agent that is slightly worse at observation may work better when acting experientially.

![Fitness Statistics for Algorithm versus Generations/Iterations](image)

Figure 6.36: Chaser Simulation Observational then Experiential Violet

As can be seen in Figure 6.36 and Figure 6.37, both methods of transfer, Best and Population, were able to learn Experientially after the initial Observational learning. The additional learning stage increased the fitness. For both test subjects, the transferring of
Figure 6.37: Chaser Simulation Observational then Experiential Orange

<table>
<thead>
<tr>
<th>Approach</th>
<th>Transfer</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observational Violet</td>
<td>Best</td>
<td>28.08</td>
<td>0.12</td>
<td>0.339</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>Population</td>
<td>28.09</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td>Observational Orange</td>
<td>Best</td>
<td>28.02</td>
<td>0.09</td>
<td>0.00464</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>Population</td>
<td>28.08</td>
<td>0.10</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.21: Comparison of Different Transfer Methods
the entire population outperformed transferring only the best individual, but only slightly, although for test subject Orange, there was a difference to a significant level (p-value < 0.05). It can be seen from the graphs, however, that during the initial 200 generations/iterations the population transfer was able to learn faster and to a higher level. Even though both methods could conceivably converge to the same point eventually, it suggests that training with population transfer could take less time or be done for a shorter period of time. It was therefore decided that population transfer was the best method to use between learning phases.

6.4 Summary of Method Experiments

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO Constants</td>
<td>$c_1 = c_2 = 0.9$</td>
</tr>
<tr>
<td>PSO Structure</td>
<td>Loop Single</td>
</tr>
<tr>
<td>PIGEON Chain</td>
<td>Cutoff 50%</td>
</tr>
<tr>
<td>PIGEON Alternate</td>
<td>Handoff 5</td>
</tr>
<tr>
<td>Increased Generations</td>
<td>1,000</td>
</tr>
<tr>
<td>Transfer</td>
<td>Population-Based</td>
</tr>
<tr>
<td>Approach</td>
<td>Method</td>
</tr>
<tr>
<td>Observational</td>
<td>Alternate Handoff 5</td>
</tr>
<tr>
<td>Instructional</td>
<td>Reward-During</td>
</tr>
<tr>
<td>Experiential</td>
<td>Alternate Handoff 5</td>
</tr>
</tbody>
</table>

Table 6.22: Final Algorithm Choices

For the initial stage of testing for algorithm selection, each prospective algorithm (NEAT, PSO, Chain, and Alternate) was tested for performance. Table 6.22 lists the decisions made
as a result of the extensive testing carried out in this chapter. Each of the algorithms had parameters tuned in order to find the best choice for two of the learning approaches (Observational and Experiential). It was found that PIGEON method of Alternate with a handoff of 5 (five PSO steps for every one NEAT step) was the best algorithm for Experiential. Alternate performs significantly different to a 95% confidence level and is better on more domains than the other algorithms. It was originally found that PSO outperformed Alternate on Observational learning; however, PSO over-fit the data set. Alternate was shown to produce better performing agents that generalized to the domain and reacted in a more intelligent manner. Additionally, it was shown that Alternate could receive benefit from running for additional generations/iteration because the fitness did not seem to level off for Observational learning. Because of the time investment, the longer runs were not be done for all domains, but it does lend credence to the learning abilities in the future with faster processing. Finally, it was found that transferring an entire population between learning approaches outperformed transferring a single individual due to the increased diversity.
CHAPTER 7
EXPERIMENTAL TESTING

7.1 Comparison and Combination of Learning Approaches

The experimental results of the previous chapter showed that as an algorithmic method, the PIEGON Alternate proved to be the best performing one across a range of domains and learning approaches. Now that the algorithm has been chosen, a comparison of the approaches can be made in order to find the best combination of learning that will produce a high performing agent that will act in an intelligent, human-like manner. The hypothesis posed in this dissertation is that the three-stage FALCONET approach that performs learning in a sequential and natural manner of Observational Learning followed by Instructional Learning, and finally with Experiential Learning should produce the best agents. For the case of comparison, the best agent is defined as an agent that:

- Functions near-optimally in performance
- Exhibits human-like behavior.

The agents produced by FALCONET are compared to a virtual power set of approaches in order to prove their validity.
Therefore, to evaluate the advantages of FALCONET, experiments were run on:

- Observational Only
- Instructional Only
- Experiential Only
- Observational then Instructional
- Observational then Experiential
- Instructional then Experiential
- Observational then Instructional then Experiential

All these approaches can be tested under the common framework using the best practices algorithms selected after the testing of Chapter 6. The order of operations of the tested groups was fixed because of several reasons. One factor is that some of the approaches depend on other approaches. For example, Observational then Instructional learning requires that Observational learning be performed first. This is because Instructional learning assumes a non-random model of the human performance already in effect that it can coach. Another factor was that some other sequences would not have particularly made sense. An example of this would be first running Experiential then running Observational. While this is physically possible to do under the framework, taking an agent that has learned on its own, followed by having it match a particular individual seems an unnatural learning order. A person that
has developed a strategy on his/her own, especially one that works well, would likely not
deviate to match someone else’s behavior.

Results that need to be analyzed vary between approaches. For example in Observational
Learning, the produced agents will be compared against the training sets observed from the
human, but will also need to be compared to the two human observation validation sets
that were not presented as part of the training set. However, the main point of comparison
between the different combinations of approaches will be based on the experiential fitness
values. The experiential fitness is an objective metric of performance that is fixed to a partic-
ular experimental domain and can be used to show the merits of a given approach because
it explicatively acts as a score of how well a task is performed. While the term experiential
fitness may sound similar to Experiential learning, experiential fitness is how well an agent
performs within a domain, while Experiential learning is the method through which an agent
learns by experiencing the domain. The fitness for a given domain is described in Section
6.1. In every case, the best agent produced by the approach is selected and run 100 times
on a randomly generated simulation of a domain to show generality and consistency as well
as to have enough data to show statistical significance. Individual experimental results for
an approach and related values are in each section. The final result tables for all approaches
together in a sorted list is given in Section 7.6.

As a reminder to the reader, the maximum possible fitness is based on the number of
seconds per simulation, therefore the maximum fitness for the Chaser simulation is 30, while
the maximum fitness for the Sheep, Crane, and Car simulations is 60. The higher the score, the better the individual is at performing the task.

The second way to compare and contrast between learning approaches is to examine the human-like qualities of the agents performing the task. Human-like is defined as a qualitative metric of performance based on the observation of the execution of the agents by a human observer. During the observation, the human notes actions that would not be considered by a reasonable person, or could physically not be done by a human in the given domain. These include actions unrelated to the task at hand and/or unnatural control schemes to implement a motion. Examples of unrelated actions would be: moving in a fixed direction unrelated to the fleer in Chaser, running into the wall and staying there or spinning in circles in Sheep, and driving off of the road or stopping randomly even when no other vehicles are present in Car. With respect to movement: moving the joystick back and forth to the extremes or moving in straight lines with hard turns to linearly to approximate a curve in any domain. These are “computer-like” moves that would seem unnatural to observe or implement physically on a joystick. Although it is difficult to describe how an agent performs in a human-like manner, it can be defined by the absence of actions that a human would not do when performing the task. The desire is to have high performing agents that also act in a human-like manner.
7.2 External Results

7.2.1 Human Performance

To help gain a baseline performance for comparison, the performance of the original human trainers can be calculated. These fitness scores are based on the runs they did for the observation sets using the formula for grading computer agents. The purpose of this experiment was to see how well a human could complete the task and observe the actions done while performing it.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Subject</th>
<th>Mean</th>
<th>StdDev</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chaser</td>
<td>Violet</td>
<td>22.72</td>
<td>0.62</td>
<td>21.66</td>
<td>23.48</td>
</tr>
<tr>
<td>Chaser</td>
<td>Orange</td>
<td>24.61</td>
<td>0.73</td>
<td>23.53</td>
<td>25.75</td>
</tr>
<tr>
<td>Sheep</td>
<td>Violet</td>
<td>29.71</td>
<td>11.07</td>
<td>12.60</td>
<td>43.64</td>
</tr>
<tr>
<td>Sheep</td>
<td>Orange</td>
<td>43.52</td>
<td>9.09</td>
<td>30.64</td>
<td>60.00</td>
</tr>
<tr>
<td>Crane</td>
<td>Violet</td>
<td>38.56</td>
<td>3.44</td>
<td>33.02</td>
<td>45.12</td>
</tr>
<tr>
<td>Crane</td>
<td>Orange</td>
<td>42.15</td>
<td>6.59</td>
<td>32.94</td>
<td>51.42</td>
</tr>
<tr>
<td>Car</td>
<td>Violet</td>
<td>26.59</td>
<td>7.87</td>
<td>11.51</td>
<td>34.72</td>
</tr>
<tr>
<td>Car</td>
<td>Orange</td>
<td>39.37</td>
<td>7.40</td>
<td>25.52</td>
<td>48.79</td>
</tr>
</tbody>
</table>

Table 7.1: Human Performance on Simulations

As a summary of the results in Table 7.1, two different test subjects had their actions recorded in the different simulations. The human subject were explained the rules of the simulation and given ample time to practice with the joystick. They had the option of throwing out runs with which they were not satisfied. However, they were not given the numeric value of the fitness of their runs. This was partially done to encourage normal behavior instead
of trying to only optimize the fitness score. Test subject Orange outperformed Violet on average and was more consistent in the recorded data by maintaining a consistent strategy between test runs.

The human performance can be used to gauge the agent performance. The humans performed competently, successfully accomplishing each task without visual errors. However, except for a single case of subject Orange in the Sheep domain, there were no perfect scores. Looking only at the fitness, one could say that the human’s performance was sub-optimal, being nearly 25% off a perfect score on average. This could be for a variety of reasons:

- In Chaser, the human reaction time likely lessened the score. From watching the runs, it could be seen that the humans were slow to react to sudden movements of the fleer. The human would typically overshoot and make a long sweeping turn to face the fleer in order to recover. The scoring system would therefore deduct points for moving in the wrong direction and not facing the fleer. While the task of pursuing was done, the human subjects’ techniques were clearly not optimal.

- In Sheep, the humans sometimes had difficulty dealing with multiple sheep. Most human techniques focused on a single sheep or small groups of sheep at a time. With this singular focus, the end result typically was to run out of time before all sheep were collected into the pen. Additionally, this behavior resulted in the human continuing to chase a single sheep into a pen, especially if they just missed in capturing one, even though there was opportunity to capture several elsewhere. Since the goal was to
capture the most sheep possible, the optimal techniques were those that rounded up
large groups of sheep and drove them all toward the pen.

• In Crane, the humans sometimes had difficulty in grabbing the boxes. In order to pick
up a box, the crane gripper had to be open wide enough to lower down over the box
without hitting it and then close the gripper on the box to secure it. In several cases,
the humans would line themselves above the box and lower the grippers, but not have
the gripper sufficiently open or begin to close the gripper too late while raising it back
up and miss grabbing the box. Most of these problem are indicative of being in a rush
or a simple mistake; however, each missing box cost time and points.

• In Car, the main penalty for humans seemed to be for not staying in the lane, losing
points for not being within a car’s width. The points for the Car simulation were based
on how many laps were performed within a time span, but these laps had to be within
the lane in order to fully count. An optimal, and therefore high score, would be to
make the same number of laps but stay in the lane. The lane deviations reduced the
score significantly.

The humans were not behaving in an optimal manner while operating in the domains.
However, by watching the runs, an external observer would still say that they were able to
adequately complete the task as the object score were 75% of perfect and the movement to
implement the task were visually appropriate.
7.2.2 Preprogrammed Agents

As another point of comparison, a set of agents were created by hand that followed a particular paradigm that fit the task at hand. These preprogrammed agents represent several hours of development work each, especially in the process of setting them up and optimizing them. They were used to see how well a typical agent would perform, but also to stress test the simulations. The handmade agents were not given any special knowledge about the simulation and were given only the same environmental inputs as the learning agents. These agents represent the typical way in which a programmer might address a given task when coding an AI without any machine-learning.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Method</th>
<th>Mean</th>
<th>StdDev</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chaser</td>
<td>PD Controller</td>
<td>26.48</td>
<td>0.18</td>
<td>26.05</td>
<td>26.72</td>
</tr>
<tr>
<td>Sheep</td>
<td>Angle Difference</td>
<td>37.42</td>
<td>13.46</td>
<td>5.28</td>
<td>60.00</td>
</tr>
<tr>
<td>Crane</td>
<td>State Machine</td>
<td>41.64</td>
<td>7.15</td>
<td>9.93</td>
<td>56.16</td>
</tr>
<tr>
<td>Car</td>
<td>Carrot Follow</td>
<td>47.26</td>
<td>17.60</td>
<td>0.00</td>
<td>59.28</td>
</tr>
</tbody>
</table>

Table 7.2: Hand-Made Performance on Simulations

For Chaser, it was determined that a simple PD controller on the angle difference would be able to follow the fleer fairly well. For Sheep, a controller that found the closest sheep and attempted to minimize the difference between the angle to the sheep and the angle to the pen also seemed to work well. This would eventually drive the sheep being chased into the pen. While a PD controller provided the minimization, several rules needed to be fired to select the correct sheep. For Crane, it was found that a state machine was needed to
accomplish the task. A state was made to find the closest box and grabbing it while another state was used to pick up the box and return to the destination. It proved difficult to create a single stateless agent that could accomplish both tasks. This could lead to the failure mode that will be seen in future learning agents and will be discussed in more depth in Section 8.2.1. For Car, an agent similar to the KnightRider autopilot system (Patz et al., 2008) was employed that used a Carrot Following algorithm. A PD controller minimizes the error between a future carrot point down the track and the heading of the vehicle to maintain lane control. While a separate controller handled the gas/brake combination to maintain the highest speed based on the car’s sensor cone to avoid collision. In this case the cone is represented by the pie wedges given by the agent preprocessing.

These agents gave programmatic insight into the nature of the simulation and the inputs given to the agents by the preprocessing. The nature of the simulations devised for motor control tasks could be mostly accomplished using reactive control and rules. However, Crane required additional planning or separation of control between states. This would prove to be difficult for the learning agents.
7.3 Single Phase Learning

7.3.1 Observation Only

Observational learning is an approach for training agents that uses a database of unobtrusively collected observations of the test subject’s actions and the environment in which he/she operated. For this dissertation, two human test subjects were monitored, each participating in ten trial runs, in each of the four different domains. Of those ten trial runs, eight were selected at random to be the training set for agent creation, while the other two were kept as a validation set. The computer agents were trained based on a comparison between its own actions and the actions of a human in the same situation. The human and the agent are synchronized together continually during the training of the agent. Because of this fact, the agents who may perform extremely well in observation may not perform well in operation alone because the simulations are non-linear complex systems. The experiments for Observational learning were created to see if the best Observational fitness necessarily means the best performing agent, whether an agent created on the training set will match the validation set from the same human test subject, how does the test subject affect performance, and how intelligent the produced agents appear upon inspection by a human.

As a point of comparison, the results (see Figure 7.1) of the best population for the Chaser simulation are compared between the fitness received for Observational learning versus the experiential fitness. It can be observed that the when the fitness for observation are sorted
from worst to best, the experiential fitness also appears to improve. From informal visual analysis of the graph between the two different fitnesses of a single individual, it can be weakly inferred that the agents acting most like the human test subject will perform better experientially. However, it can also be seen that the entire population from the end of the Observational learning run is relatively well fit.

![Graph showing Fitness Statistics for Algorithm versus Individuals](image)

**Figure 7.1: Chaser Simulation - Observational versus Experiential**

Approximately 10 individuals belonging to the top cluster of best individuals did not actually contain the best agent when graded experientially. Meaning that the agents that best matched the human (observational fitness) did not score the best when graded on its own (experiential fitness). The best agent experientially actually ranked 119 out of 150 (Figure 7.1 the center spike of three on the left). These results, which also hold true in the other domains, suggest that the agents can become over-fit to observation, actually becoming less
optimal at the task by acting too similar to the sub-optimal human. Since the main aspect of this dissertation is to create optimal yet human-like agents, the representative agent for Observational learning will not be the agent with the top Observational fitness score, but instead, the agent with the best experiential fitness score produced through Observational learning.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Subject</th>
<th>Human</th>
<th>Obs Training</th>
<th>Obs Validation</th>
<th>Experiential</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chaser</td>
<td>Violet</td>
<td>22.72</td>
<td>19.51</td>
<td>18.24</td>
<td>27.03</td>
</tr>
<tr>
<td>Chaser</td>
<td>Orange</td>
<td>24.61</td>
<td>22.42</td>
<td>22.97</td>
<td>26.67</td>
</tr>
<tr>
<td>Sheep</td>
<td>Violet</td>
<td>29.71</td>
<td>47.67</td>
<td>46.71</td>
<td>46.71</td>
</tr>
<tr>
<td>Sheep</td>
<td>Orange</td>
<td>43.52</td>
<td>49.60</td>
<td>47.73</td>
<td>44.84</td>
</tr>
<tr>
<td>Crane</td>
<td>Violet</td>
<td>38.56</td>
<td>44.49</td>
<td>41.88</td>
<td>3.98</td>
</tr>
<tr>
<td>Crane</td>
<td>Orange</td>
<td>42.15</td>
<td>41.33</td>
<td>39.69</td>
<td>3.91</td>
</tr>
<tr>
<td>Car</td>
<td>Violet</td>
<td>26.59</td>
<td>51.44</td>
<td>51.89</td>
<td>17.89</td>
</tr>
<tr>
<td>Car</td>
<td>Orange</td>
<td>39.37</td>
<td>50.80</td>
<td>51.42</td>
<td>31.48</td>
</tr>
</tbody>
</table>

Table 7.3: Comprehensive Table of Observational Learning Fitness

The total results for Observational Learning are condensed in Table 7.3. All results are based on the best agent in terms of experiential fitness score, the higher the better. The column labeled “Human” is the experiential score of the human test subject. The columns labeled “Obs Training” and “Obs Validation” are fitnesses in comparison to the test subject for the Observational training and validation sets respectively, using the observational score. Finally, the “Experiential” column is the experiential score of the best agent produced through Observational learning.

When compared to their particular human trainer, the agents effectively agreed with the human 77.6% and 78.17% of the time on the training set for Violet and Orange. Recall that
this score is based on the agent that performed best on its own (Experientially) and not the top match to the joystick (Observationally) because the desire to have high performing agents. In comparison, the agents which were best at observation were closer to 82% in matching the human’s actions. To provide insight in how the agents generalized to match their trainer, two runs of human-observed data were kept from the training set. These run can then be presented to the agents as a validation set. The best agents received a score of 75.58% and 77.05% for Violet and Orange respectively. These percentages are fairly close to the training set. These results support the notion that these agents can generalize to the person whose observed performance was used to train them.

However, the focus for this dissertation is on creating agents that can perform on their own in a randomly generated environment. The agents were originally only scored in their ability to match the actions of the human trainer. No specific consideration, with the exception of the final agent selection which did use the experiential score, was given to how well the agents performed in a simulation based on the goals of the given task. The agent is only presented with the human’s actions and the environmental stimuli that caused those actions. Those agents are then put into the simulator to react completely on their own to accomplish the task. In Table 7.3, the results of how well each best Observationally-trained agent performed is given. In three out of the four domains, the agents were able to successfully learn how to appropriately function at or above the level of a human. The notable exception is the Crane domain where no agent successfully learned how to operate the crane. For the
other domains, it shows the ability for Observational learning to successfully transition from observed information to independent agent that generalize to problem.

In the domains where the agents worked well, the fitness score for the produced agent were similar to the performance of the original trainer and in some cases better. In the Chaser domain, the agents were able to outperform the original human trainers. This can be attributed to the way the agents made fewer small mistakes when chasing. The humans would sometimes overshoot the target or make mistakes they would have to later correct. The agents tended to not make these mistakes and could generally chase more smoothly.

For Sheep, the agents were able to capture approximately 12 of the 16 sheep on average. In the case of Violet, the agent performed significantly better than the human trainer. Violet’s Sheep runs were inconsistent and in some cases only captured three (3) sheep in the 60 second time limit. In Car, the agent based on Orange was able to perform approximately three (3) laps on average, actually doing better than Violet. In this case, the agent based on Violet did not perform as well, averaging only 1.5 laps.

Another interesting outcome to note, the agent based on Violet outperformed the agent based on Orange in both Chaser and Sheep even though Violet did worse than Orange in both domains. It is theorized that this is similar to the traits noticed by Pomerleau in (Pomerleau 1991). Violet was more inconsistent, but that also meant that he made more mistakes from which he needed to recover. This, in turn, provided the agent observable examples of recovery to learn. This allowed to the agent to learn to recover where the agent based on Orange had to learn from a consistent good performance without mistakes. As a
result, agent based on Orange was unable to recover from bad situations. However in the Car domain, the agent based on Violet made more mistakes from which it could possibly recover, but mistakes when driving are very costly. An agent running into another car is severally penalized. The agent based on Orange that stayed in its lane better was able to score higher.

It is important to cover the failure of the agents on the Crane domain (experiential fitness of 3.95 out of 60) even though the agents received relatively high scores on observation (observational fitness of 42.91 out of 60). This can be partially attributed to the way Observational learning was performed. The agents did do the same action as the human in a majority of the cases, but there were still approximately 25% of the time when the agent did not. Since the domains are complex non-linear problems that evolve over time, small differences in actions early on will affect all states in the environment later on. An early mistake can cause many problems later on. In Observational learning, the positions of the computer agent and of the human were synchronized to make the environments compatible. In the synchronization process, the agent would be given the same state as the human at regular intervals. Therefore, if a human had a box when the synchronization happened, the agent would be given a box and brought to the same position. However, this effectively saved the computer agent from bad situations it created. Therefore, it can be observed that the agent could usually get to a box or, once a box was picked up, it could be returned to base. However, the agent never learned to appropriately pick up a box. The agent being synchronized with the human created the situation in which the agent who did not pick up
a box would suddenly have a box because the human had a box when synchronization took place. Once the agent had a box, it could then learn how to return the box. Since all observed information was put in one database and the picking up of boxes was in the minority of time in that database, the agent seemed to optimize for the majority of situations, namely finding and returning boxes. If the observational data had been manually partitioned into contexts such as in (Fernlund et al. 2006) or automatically such as in (Trinh 2009), it is possible that the agent could have learned the three separate maneuvers independently. The context-based concept is mentioned as a future work in Section 8.2.1.

The agents created from Observational learning performed rather well in three of the domains. A questionnaire was given to the original human trainers to observe the agents based on themselves using Observational learning and answer a series of questions about the behavior of the agents. The best agent was loaded into a randomly generated environment for each domain and the human observed the agent moving on the screen. When the trainers were asked on how much they felt the agents’ actions were similar to their own, the agents scored a 4.17 out of 5.0, where 5 was perfectly like themselves. The humans would notice similar traits to their own in several of the domains. For example in the Sheep domain, Violet typically chased the sheep in a clockwise manner while Orange would chase them in a counter-clockwise manner. This type of movement was reflected in the agent. Another example was the agent behavior in the Car domain. Violet was more cautious and would stop and wait for cars at the bottom intersection (see Figure 6.5). On the other hand, Orange
would typically dart into any open gap in the cars in which he could fit. These behaviors are actively reflected in the agent produced from Observational learning.

![Joystick Motion over Time](image)

**Figure 7.2: Chaser Simulation - Joystick Movement Comparison**

To evaluate human-like performance, there were two main experiments performed: analysis of data and human inspection. In Figure 7.2, the joystick data for turning are displayed for a Human (red) and the Observationally-taught agent (green) for a 10 second time period in the Chaser domain. To read the plot, the X-axis is time while the Y-axis is the absolute position of the joystick with 0.0 full left, 0.5 middle, and 1.0 full right. When chasing, the human would perform smooth motions moving from one direction to the other within a certain reaction time. The motion of the agent was very similar with comparable reaction times and motions, with some additional “rough” points.
The other method for detecting human-like qualities is using human inspection of the agent performance. The trainers were asked if the agents performed in a human-like manner visually. For this question, the agents scored a 4.5 out of 5.0 (5 being perfectly human-like). This question, similar in some ways to a “Turing Test,” was to see whether the agents’ movements appeared like a human while operating in the domain. This could be important in the future for agents used in a simulator to emulate people or simply agents that appear competent, though sub-optimal. The trainers were also asked to observe the agents’ performance while the joysticks physically implemented the actions of the agents. The joysticks would move to the position needed to implement the desired action. The human was asked to hold onto the joystick to sense haptically the actions of the agents as well as allowed to view the performance on the screen. When asked whether they felt that the agent performed haptically human-like, meaning the agent’s movements were similar to a human, the agent was rated 4.0 out of 5.0 on average. This question, which is related to a question in Section 7.3.3, was designed to determine whether the actions seemed reasonable. An example of this would be if one was driving a car one could move the steering wheel back and forth quickly and still maintain a straight line. An external observer only looking at the car motion, but not the tires, would say that visually nothing was abnormal with the car. But an individual in the car would say that the steering wheel movement was abnormal and not human-like. This is the difference between the input to the simulation (actions) and the output of the simulation (observables). The agents produced by the means developed in
this dissertation were designed to both, make intelligent actions and to observably appear intelligent.

For the agents trained using only Observational learning, the end product was very promising. An agent was able to be trained to originally match a human’s actions when presented with similar stimuli in the environment. The produced agent was able to make use of that knowledge to appropriately operate in three of the four domains (Crane being the exception). That agent was judged to have performed at a level similar to a human. The agent also retained traits of the human trainer and was said to behave in a human-like manner, both visually and haptically. Observational learning should provide a strong basis for the initial stage of FALCONET.

7.3.2 Instructional Only

Instructional learning is an approach for creating agents purely based on the interaction with a human trainer. The approach is explained in depth for execution and scoring in Section 5.3.2, however, a brief overview follows.

The Instructional approach uses a population of agents which are graded together by the human coach, who uses positive and negative rewards by providing corrective counterforce on the haptic joysticks. The collection of agents eventually learns the task by maximizing reward from the coach. This is different from Experiential learning, which maximizes the
fitness from interaction with the environment. The algorithm for the approach is based on PSO and its ability to learn quickly. This feature is important since the haptic interaction with the trainer must happen in real-time. The experiments are devised to see how quickly Instructional can learn, whether an agent can be taught purely by Instruction, and the level of performance of an agent produced through Instruction.

An example population of Chaser can be seen in Figure 7.3. The algorithm is able to optimize rather quickly from an original set of random agents and eventually receiving close to full reward. Since the PSO algorithm and learning happens on a population, it can be seen that not only does the best agent get better, but the worst agent in the population improves as well (25 to 29 for best and 15 to 27 for worst). Even within the 20 coaching sessions, the entire population begins to converge to a point. This will be important for later training stages and when this stage is used as the middle step of FALCONET.

The original plan for the dissertation research did not include doing Instructional learning by itself. It was deemed too difficult to train an agent from a random population in a reasonable amount of time. However, during some initial testing, it was found that the agents would respond well to coaching in a short period of time for the Chaser problem. From the original random agents that would just “rattle” the joysticks in the beginning, the agents would begin to follow the fleer when brought close after some instructional intervention by the human trainer. Then the agent would smooth out its movement and follow the fleer more often, even when turned away. Finally by the 20th session, the agent would typically follow the fleer from almost any initial position and require very little correction. The
human trainers, Violet and Orange, were tasked with the job of creating an agent from scratch using Instructional learning. They found that the “online, hands-on training [was] surprisingly responsive.” And when asked if they believed they could train an agent only haptically they responded positively with a 4.5 out of 5.0. Therefore, for this reason, it was decided to include a comprehensive evaluation of Instructional-only learning.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Ins Best</th>
<th>Ins Exp</th>
<th>Experiential</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chaser</td>
<td>29.10</td>
<td>29.10</td>
<td>26.04</td>
</tr>
<tr>
<td>Sheep</td>
<td>51.54</td>
<td>43.18</td>
<td>25.35</td>
</tr>
<tr>
<td>Crane</td>
<td>37.90</td>
<td>34.21</td>
<td>3.98</td>
</tr>
<tr>
<td>Car</td>
<td>50.89</td>
<td>47.55</td>
<td>1.29</td>
</tr>
</tbody>
</table>

Table 7.4: Comprehensive Table of Instructional Learning Fitness
The fitness of agents can be found in Table 7.4. “Ins Best” represents the max Instructional fitness of the agent at the top of the list. “Ins Exp” represents the Instructional fitness of the agent that did best experientially and “Experiential” was the experiential score for that individual. Similar to Observational learning, it can be seen that the top Instructional agent was not the always the agent that performed best experientially. The individual which received the most reward may not have been able to operate on its own. A problem could have come from the Instructional learning method. During the instruction of an agent for Car, the agent could be forced by the trainer to get on to the right loop (see Figure 6.5) after which the agent would follow the road and make the top turn into the middle road. However, these agents when operating on their own could not properly make the initial left turn from the middle road into traffic, either hitting a car or missing the turn and driving off the track. The way Instructional learning works, the agents were rewarded for following the road by the coach, of which the most time is spent on the loop. To optimize the reward, the agents would ignore punishment of actions that did not constitute much physical time. This problem should be solved by initially coaching agents that have some “common sense” which is the purpose of Section 7.4.1, Observational then Instructional learning.

By looking at the “Experiential” column of Table 7.4, the agents can be seen to succeed in Chaser and Sheep. The agents actually were able to outperform even humans on Chaser while the agent was approximately as good as Violet on Sheep, capturing seven sheep on average. The Instructional-only agents did not perform well on Crane or Car. These were the two most complex problems. Crane likely did not work for the same reasons as Observational
learning - the lack of defined separate states. While the agents’ failure in the Car domain was likely the result of the aforementioned problem with the amount of time spent on an individual situation as mentioned in the previous paragraph.

In general, it was surprising that Instructional learning was actually able to make an acceptable agent at all. The method was effectively only shown the task 20 times, with a learning event once a second. The haptic-based training proved to be a quick way to teach an agent, but it did not seem to be able to learn complex domains on its own. However, the properties of Instructional learning should prove to be a good middle stage for FALCONET by learning quickly.

7.3.3 Experiential Only

Experiential learning by itself is different from the other forms of learning. There is no human interaction with the system, as it only attempts to optimize the fitness for a given domain (experiential fitness). This can lead to some unpredictable behavior because the agent is graded on its performance in the domain and not on its actions with respect to human-like qualities. The experiments are to examine how human-like the behavior of a learning system is without human interaction and the performance of the agent produced by this approach.

In the comparison to human-like performance, the joystick movements of a human (observed) are compared to the agent’s behavior. The human data are in fact the same as
Figure 7.4: Chaser Simulation - Joystick Movement Comparison

Figure 7.5: Chaser Simulation - Position Comparison
those described in Section 7.3.1, and are characterized by the human exhibiting smooth behavior with a normal reaction time. While the Experiential-taught agent moves almost instantaneously back in forth in order to implement the same type of motion. Because of the nature of the simulation, these actions would be integrated mathematically and produce a the desired motion (see Figure 7.5). For this reason, it is important to not only view the observable motion of the agent, but gauge the humanness of its actions as well.

On the “Turing Test” incorporated in the human trainer questionnaire, the agents were graded rather low for performing in a human-like manner from a visual point of view, receiving a score of 2.5 out of 5.0. The human trainers were subsequently given the opportunity to determine how well the agent performed in a human-like manner by haptically feeling how the agent acted through the joystick. In this evaluation, the agent then scored even lower, receiving a 1.5 out of 5.0. Through human inspection of both of the movement and the actions of the agent, it can be readily seen how human-like an agent performs. Therefore, it is unnecessary to view the joystick movement plots for future comparison.

The main reason for this performance difference is that the agents have learned some extreme behaviors to accomplish tasks. For example, in Chaser the agent developed into a “Bang-Bang” controller, which repeatedly slams the joystick quickly to the left and then to the right in order to drive. This maneuver is very un-human-like, but a “Bang-Bang” controller is considered an optimal control with respect to time because it always uses maximum acceleration (Bryson and Ho 1975). If a controller always uses maximum acceleration and deceleration, the object being moved would have to reach the destination in the shortest
amount of time, but it in no way constitutes a smooth or desirable motion. Another example of this can be found in Car where the agent will accelerate at full speed until it reaches the point considered to be tail-gating the car in front, and then slam on the brakes to avoid the tail-gating penalty. Again, this behavior may be optimal for making the most laps, but it is not very human-like.

The other issue with Experiential learning is the lack of “common sense” in terms of the actions performed. Similar to the “Bang-Bang” issue above, the agents sometimes learned interesting, but not technically incorrect behavior. An example of this comes in the normal human concepts of movement and direction. In the Car domain, the agents are tasked with accomplishing the most laps, staying within the proper lane, and not hitting or interfering with other cars. One of the best Experiential agents learned how to accomplish this task, but drove the entire track in reverse. The agent did meet all the criteria of the fitness function, but if viewed externally this behavior would be considered inappropriate. Similarly, in Chaser and Sheep, the agent would sometimes chase the fleer while moving backwards, or drive the sheep into the pen backwards. The Experiential agents do not have a concept of “proper” movements. Since the agents are spawned from random population moving about in a random way, the early generations found some method that gave a reasonable fitness and which subsequently influences all children of that fit individual. This problem is solved with FALCONET’s bootstrapping procedure as listed in Section 7.4.2.

While many of the faults of the Experiential agents have been discussed, it is difficult to ignore the results in terms of how well the agents performed (see Table 7.5). The Experiential
agents outperformed the best human, the hand-made agent, and the best observational agent in two out of the four tasks. In Chaser, the agent was able to almost always keep close to the fleer, learning a behavior of staying slightly to the outside of the fleer and forcing it into a circular pattern around the center. This self-learned procedure maximized points by forcing the fleer into a known pattern from any random starting position. In Sheep, the agent learned a spiral pattern where the agent would make circles around the pen and on each pass move in slightly. Using this behavior, the agent was almost always able to capture all the sheep. For Car, the agent consistently stayed in the lane while moving around the track. But to accomplish this, the agent used the aforementioned “Bang-Bang” controller to turn the “steering wheel” left and right hard while also being hard on the Gas/Brake. The agent was able to average almost four (4) out of a maximum five (5) laps every time, but being “Honked at” extensively as a very aggressive driver.

The Experiential agent, however, did not learn how to properly operate the crane. This is thought to be related to the lack of contextual decomposition as discussed in Section 7.2.2. Another possibility could be the fitness function itself, as it may not award enough “partial credit” to give a hint as to which direction to optimize. The fitness was based on the number

Table 7.5: Comprehensive Table of Experiential Learning Fitness

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Best Human</th>
<th>Hand Made</th>
<th>Best Obs</th>
<th>Experiential</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chaser</td>
<td>24.61</td>
<td>26.48</td>
<td>27.03</td>
<td>28.20</td>
</tr>
<tr>
<td>Sheep</td>
<td>43.52</td>
<td>37.42</td>
<td>46.71</td>
<td>58.31</td>
</tr>
<tr>
<td>Crane</td>
<td>42.15</td>
<td>41.64</td>
<td>3.98</td>
<td>3.91</td>
</tr>
<tr>
<td>Car</td>
<td>39.37</td>
<td>47.26</td>
<td>31.48</td>
<td>40.56</td>
</tr>
</tbody>
</table>
of boxes collected and the distance the boxes were moved toward the destination. However, no points were rewarded for simply moving toward a box or touching a box. Without this reward, the Crane agent, in its early random stage, never learned to pick up the box. A description of possible future work to determine whether this in fact was the problem is presented in Section 8.3.1.

Experiential learning has shown that it can make very successful agents - ones that can maximize the fitness score for a given domain. However, Experiential learning has also shown that the highly performing agents created may behave in a very unhuman-like manner. These agents, therefore, would not be appropriate to insert into a simulation to impersonate a human or in a physical system were it could damage the mechanics or electronics as a result of the rough actions of the agent. The Experiential agents may perform quite well, but they do not appear to perform intelligently. However, the Experiential learning phase has shown it can augment agents by having them improve their fitness over many simulations that are faster than real-time. This should be an excellent final phase for FALCONET to improve the agents beyond the original human teachings.

### 7.3.4 Single Phase Summary

FALCONET is comprised of the three distinct learning phases: Observational, Instructional, and Experiential. It was found that each method on its own was able to produce intelligent agents to varying degrees. Observational learning could take in recorded observations of a
human trainer and, from this information alone, create an intelligent agent that exhibited the traits of the trainer and could successfully operate in three of the domains (Chaser, Sheep, and Car) to the level of the original trainer. This agent was shown to not only perform well, but was also capable of exhibiting human-like behavior when accomplishing the task. Instructional learning was shown that it could produce an intelligent agent for the two simpler domains (Chaser and Sheep) in a very short period of time, using only the haptic feedback in real-time. It was found that with enough time and effort, one could teach the agent from scratch; however, this was not the original goal of this phase. Finally, Experiential learning was shown to be able to produce very highly performing agents in three of the four tasks (Chaser, Sheep, and Car), significantly beyond the other two approaches. However, it was deemed that the agents produced did not react or move in a human-like manner. Each of these stages individually has shown promise. Our hypothesis is that combining these different approaches together should produce even better results.

### 7.4 Two-Phase Learning

The main objective of this dissertation is to show that FALCONET, the combination of the three learning approaches, would be the best way to train an agent. To accomplish this task, different multi-phase approaches needed to be executed to provide a comparison, not only with the other multi-phase learning processes, but also with the single-phase ones described in the previous section. Each of the following methods takes the best population
of the previous phase and continues the learning process using a second approach. The
result of this second phase is another population that can be loaded into the third phase.
In each section, the multi-phase method is compared to its sub-components. For example,
the agents produced by the Observational then Experiential learning process is compared to
Observational learning and Experiential learning agents to see if there was an improvement.
We begin by evaluating two-phase learning methods, ultimately followed by the three-phase
learning method of Observational then Instructional then Experiential.

7.4.1 Observational then Instructional

The purpose of combining Observational and Instructional learning was to initially base an
agent on a human test subject and then, after the Observational training was finished, to al-
low the same human trainer to refine the agent by adjusting for errors or inconsistencies. It is
not possible in Observational learning to observe the human perform every possible situation.
Moreover, it can also be difficult to correct for the improper action of an observation-taught
agent if it did not react properly for a particular situation. The Instructional learning is
then used to correct for flaws in the observationally-taught agent. Experiments were done in
order to determine whether the haptic feedback helped the human trainer coach the agent,
then coaching occurred in every domain, and then the experiential fitness is compared to see
if the agent improved with the additional training step.
The Instructional process incorporated haptics with a joystick to give the human trainer additional insight into the motions of the agent. When the agent was operating, the top-performing agent, as a result of coaching reward (see Section 5.3.2), would have physical control of the joystick. If the trainer thought that the motion was inappropriate, the human could hold down the trigger button and force the agent to do a different action by moving his/her joystick to indicate the appropriate one. If the trainer thought that the action was appropriate, he/she could simply let the agent continue performing the action undisturbed, for which the agent would receive reward and partial validation.

When the trainers were asked whether the haptic interaction with the agent during task performance (by the agent) gave a greater understanding of the agent’s decisions, they responded with a 5.0 out of 5.0. One trainer felt that “the haptics does give an extra sense of what the agent is doing ... without the haptics, it would be more difficult to determine how to correct its behavior.”

They did find some flaws with the method, however. One trainer felt that “contradictory haptic feedback made movement and control that required accuracy and precision difficult.” With the way the haptics was implemented, the agents’ actions would still be felt even if the human wanted control. While the goal of the haptics was to feel the differences in agent intent, the human would have to correct for the movement, now with an additional disturbance. For example, if the human was trying to operate the crane and the agent was still acting erratically, the joystick would be moving back and forth quickly. The trainer would attempt to pick up the box, a skilled maneuver, but would sometimes miss or drop
the box because of the extra joystick movement. A study should be done to determine whether this effect is detrimental to training for the human. However, that is outside the scope of this research and is left for future research.

During the experiments, the human trainers were given an agent based on their own previously-observed runs. They were to correct the agent when they felt it was behaving inappropriately or if the agent was not performing as well as desired. The plots of the rewards over the 20 training sessions for each of the four domains is given below.

![Fitness Statistics for Algorithm versus Generations/Iterations](image)

**Figure 7.6: Chaser Simulation - Observational then Instructional**

It can be seen that for Chaser in Figure 7.6, the observationally-trained agent was already performing quite well. Trainer Orange was more even handed in correcting for small differences. Trainer Violet, on the other hand, tended to over-correct the agent. The trainer found that “sometimes the agent will make a mistake, which I would try to correct with
unusual actions, which the agent picks up as a regular behavior." This relates to the quick learning rate purposely incorporated within Instructional learning. The agent trusts the trainer rather implicitly. The agent does need to learn how to extract itself from problems, but the "cure" could have detrimental effects on other aspects of the performance. Much of the rest of the training run was spent trying to undo what was unintentionally taught. Although in the case of Chaser, the learning system could still overcome these problems.

Figure 7.7: Sheep Simulation - Observational then Instructional

In Sheep, the agent also performed quite well. In this case, the Orange trainer found that "It’s difficult to know when to correct since its action at any point in time could be part of a larger sequence of movements. Altering a particular action could invalidate an otherwise sound strategic situation." The Orange trainer actually felt he was making the agent perform worse with additional training. Conversely, the Violet trainer felt the agent
was better than himself. These feelings are confirmed in the reward plot (Figure 7.7) by the consistently higher Instructional fitness scores being given by Violet.

![Fitness Statistics for Algorithm versus Generations/Iterations](image)

Figure 7.8: Crane Simulation - Observational then Instructional

Crane was a different situation, however. The original observationally-taught agent was unable to learn the task on its own, and therefore the Instructional training proved to be difficult. This is reflected in Figure 7.8 where the human trainers do not give very positive rewards, averaging close to 37.5 out of 60.0. However, the Violet trainer found that if he picked up the box himself, the agent would drop off the block in the correct location. The Orange trainer found another interesting aspect with respect to the two joysticks used in this problem. He felt that “the two sets of controls interfered with each other. When I only controlled the movement, the crane control would become more correct and would drop down.” It was requested by the trainer whether it was possible to only correct for
one joystick at a time and let the agent operate the other or potentially use a two different
PIGEON agents, one for each joystick. Both of these ideas could potentially improve complex
Instructional learning. These concepts are discussed as future work in Section 8.2.2. In
general, both trainers found that they could get the agent to accomplish part of the task,
but the trainer had to get the agent in the proper situation. It is possible that with additional
training sessions (beyond 20) the human trainers could fix the agent, but the time involved
because of the real-time nature of the training made that situation not logistically feasible.
The additional instruction on top of the observationally-taught agent did not fix the agent
in the Crane domain, however, the process did trigger new extensions of research.

![Fitness Statistics for Algorithm versus Generations/Iterations](image)

**Figure 7.9: Car Simulation - Observational then Instructional**

In Figure 7.9, the reward for the Car domain had mixed results, with the reward mostly
distributed above 45.0 and a single case of 60.0, but without consistency. On the perfect
60.0 occasion, the Violet trainer trusted the agent enough to completely allow control of
the car for the entire training session. This domain, because of its complexity, also caused
some training problems. If an agent collided with another car, the trainer had to do some
interesting maneuvers to correct the situation in real-time, such as move in reverse down
the road or pull off the lane until the traffic passed and then pull back into the lane. These
behaviors were learned by the agent and sometimes detrimentally affect its performance.
Finally, the trainers noticed that in the Car simulation the agent would sometimes behave
erratically at the beginning of the run. This could be because the agents have not yet been
sorted out properly during the start of the run, as all agents in the population are vying for
control. If they are tied at the beginning, an agent is selected at random, regardless of its
performance (see Section 5.3.2). At the very beginning of the run, the agent typically needs to
wait for traffic. During this time, the agent should not be moving. Since reward is distributed
once a second, the agents all have the same reward before the first second. Therefore, an
agent is selected at random from the population to be the representative one whose actions
are felt through the haptic interface. However, after the first second, the human holding the
agent back would penalize all agents desiring to go forward, the population would re-sort,
and the agent felt through the haptics would seem to be more sane. This aspect could be
corrected by remembering the sorted list from the end of the previous run.

The results in Table 7.6 are unfortunately inconclusive. While for some domains and
human trainers, there were minor improvements, other domains saw some slight performance
drops. The extra Instructional phase showed improvements for Orange in Chaser, Sheep,
Car. The agents that learned from Violet, on the other hand, had decreases in performance for Chaser, Sheep, and Car. Although, each of these was not different to a statistically meaningful degree (p-value < 0.05) except for the Chaser domain. Therefore, it cannot be concluded that Instructional definitely improved an already good agent produced through Observational learning. Nevertheless, it could possibly be said that the more even-handed training technique of test subject Orange showed improvements. This phase can be highly influenced by the trainer. Further research with more human subjects should be performed in the future.

The final Instructional reward scores (column “Instructional”) for Chaser, Sheep, and Car were all rather large, meaning that the human trainer accepted the majority of the agents’ actions. Therefore, at the end the coaching session, the coaches validated the actions of the agents produced through this two-phase approach for all domains except Crane. Additionally, when the human trainers were asked whether they felt that the agent improved over the course of the haptic training, the trainers responded positively with a 4.5 out of 5.0.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Subject</th>
<th>Instructional</th>
<th>Obs Only</th>
<th>Obs-Ins</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chaser</td>
<td>Violet</td>
<td>30.00</td>
<td>27.03</td>
<td>26.28</td>
<td>4.09e-23</td>
</tr>
<tr>
<td>Chaser</td>
<td>Orange</td>
<td>30.00</td>
<td>26.67</td>
<td>27.00</td>
<td>1.98e-26</td>
</tr>
<tr>
<td>Sheep</td>
<td>Violet</td>
<td>59.01</td>
<td>46.71</td>
<td>45.17</td>
<td>0.251</td>
</tr>
<tr>
<td>Sheep</td>
<td>Orange</td>
<td>56.01</td>
<td>44.84</td>
<td>45.09</td>
<td>0.456</td>
</tr>
<tr>
<td>Crane</td>
<td>Violet</td>
<td>40.58</td>
<td>3.98</td>
<td>3.95</td>
<td>0.246</td>
</tr>
<tr>
<td>Crane</td>
<td>Orange</td>
<td>37.24</td>
<td>3.91</td>
<td>3.94</td>
<td>0.249</td>
</tr>
<tr>
<td>Car</td>
<td>Violet</td>
<td>52.99</td>
<td>17.89</td>
<td>14.92</td>
<td>0.146</td>
</tr>
<tr>
<td>Car</td>
<td>Orange</td>
<td>52.89</td>
<td>31.48</td>
<td>36.48</td>
<td>0.0655</td>
</tr>
</tbody>
</table>

Table 7.6: Comprehensive Table of Observational then Instructional Fitness
The performance was within a 5.0% difference between the observationally-taught agents and the observation-instructionally-taught agents in the three working domains. The agents did not appear to have made significant advancements through Instructional learning, but the method was not detrimental. The final assessment on whether Instructional learning was truly helpful will be seen in Section 7.5.1.

Other aspects of Instructional learning were useful by allowing the human to get a better understanding of the agent and to gain greater insight in how to train the agent better. The method also allowed for validation of an already produced agent. The human trainer could be used to decide in which sections certain actions were appropriate and attempt to train the agent only in those sections where the actions were not appropriate. This could help the trainer create boundaries to identify any sections where agents need to most urgently be retrained. These could potentially become contexts (see Section 8.2.1). Or the training could use sub-agents and use modularity to solve some issues (see Section 8.2.2). However, both of these items will be left for future work. Overall, the human trainers weakly agreed that haptics helped the training process with a score of 3.5 out of 5.0. With some changes to the way in which the haptics-based training was applied, including separating the joysticks and reducing agent feedback while the coach was in control, Instructional learning could be made to overcome some of the problems identified by the test subjects and improve the agent training process.
7.4.2 Observational then Experiential

The concept behind Observational then Experiential learning was to take the relatively good agents produced in Section 7.3.1 using Observational learning and allow the system to optimize and improve agents on their own through Experiential learning. The system should produce better agents, as it takes in a neural network given competence by being bootstrapped on human observation and then adds additional practice. Of particular interest here is a comparison to Section 7.3.3 where only Experiential learning was used, with no human interaction. The main experiments, when combining the Experimental stage, were to test whether there was an increase in performance with the additional practice and to see if the Experiential stage removed any of human-like mannerisms gained from the original Observational stage.

In Figure 7.10, the first item to note is the original fitness of the individuals in the initial generations/iterations. The Observational learning phase produced rather fit agents all on its own. Furthermore, the agents based on observation of trainer Violet, which were originally better than the agents based on Orange, continued to perform better over time in the Experiential learning phase. Some additional performance gains were made through Experiential learning; however, upon viewing the produced runs, the agents produced by the end are less smooth in their motion. As mentioned in Section 7.3.1, while the agent becomes more proficient, it sometime acts in a less human-like manner when reacting to
the environment. However, the agents are still much less chaotic than the Experiential-only agents.

In Figure 7.11, the results are quite promising. Both sets of agents (those based on Violet and Orange), starting from the observational training, can already capture approximately 12 sheep within the time period. Through Experiential practice alone, the agents have seemed to solve the problem by generation/iteration number 500, where the agents are consistently capturing all 16 sheep. The additional Experiential phase has given the average agent produced by the two-stage method higher fitness than the best agent from Observational learning. It will be seen in Table 7.7 that the learning system is fully generalized. Similar to the Chaser game, the agents have become more proficient, but now make more
un-human-like “jittery” moves when near sheep when compared to the Observational-only agent on which it was based.

In Figure 7.12, neither of the agent sets (Violet and Orange), were able to function successfully after observation. Viewing the agents’ operation, the agents based on Violet typically found the ability to raise up, open the gripper, and hover above a box or move back and forth between two boxes. The agents based on Orange typically lower the gripper to the ground, close the gripper, and run into some boxes. Both sets were unable to pick up any boxes but they contain most of the concepts, such as finding a box or returning a box. We have already discussed in other sections some theories on why this occurs. These theories relate to multiple contexts and the fitness function.
Figure 7.12: Crane Simulation - Observational then Experiential

![Fitness Statistics for Algorithm versus Generations/Iterations](image1)

Figure 7.13: Car Simulation - Observational then Experiential

![Fitness Statistics for Algorithm versus Generations/Iterations](image2)
Figure 7.13 also shows a good deal of promise. The agents from Observation are able to achieve a fitness in the 30’s (approximately 2.5 laps) and Experiential learning is able to learn into the 40’s (approximately 3.3 laps). The combined effort from both sets is able to learn about the 55 mark out of 60 (approximately 4.5 laps or possibly 5 laps with several honks). The agents retained many of the human characteristics of the original trainer. Agents based on Violet are more tentative in merging into traffic and those based on Orange are more aggressive and merge at almost any opening. The Orange-based agent also kept an interesting feature where it would over-shoot the center lane and readjust. This does not affect the fitness of the individual, but does reflect the retention of a certain style of driving. The agents were able to drive well, but began to add some driving features that did not exist in the original observation-taught agents. The agents adopted a policy of slowing down when presented with oncoming traffic in the opposite lane. While this behavior is not strictly improper and probably helps the fitness in certain situations, it does produce some unexpected braking. This would relate to a possible future work to feed this agent back to Instructional learning to smooth out these behaviors.

The overall comparisons are listed in Table 7.7. The results show that Observational then Experiential learning (Obs-Exp) improved the agents of Observational only learning (Obs) in every working domain to a significant level (p-value < 0.05 in P-value Obs), except for the Sheep domain when agents based on Violet decreased in performance. The explanation of this oddity is on how the final scores are computed. When graded during Experiential learning, the agents are based on eight runs, of which the top agent based on Violet received a
Table 7.7: Comprehensive Table of Observational then Experiential Fitness

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Subject</th>
<th>Obs</th>
<th>Exp</th>
<th>Obs-Exp</th>
<th>P-value Obs</th>
<th>P-value Exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chaser</td>
<td>Violet</td>
<td>27.03</td>
<td>28.20</td>
<td>28.15</td>
<td>3.99e-58</td>
<td>0.0365</td>
</tr>
<tr>
<td>Chaser</td>
<td>Orange</td>
<td>26.67</td>
<td>28.20</td>
<td>27.84</td>
<td>7.26e-57</td>
<td>7.48e-14</td>
</tr>
<tr>
<td>Sheep</td>
<td>Violet</td>
<td>46.71</td>
<td>58.31</td>
<td>36.66</td>
<td>1.63e-06</td>
<td>9.89e-29</td>
</tr>
<tr>
<td>Sheep</td>
<td>Orange</td>
<td>42.19</td>
<td>58.31</td>
<td>57.66</td>
<td>2.22e-11</td>
<td>0.228</td>
</tr>
<tr>
<td>Crane</td>
<td>Violet</td>
<td>3.98</td>
<td>3.91</td>
<td>3.99</td>
<td>0.407</td>
<td>0.0286</td>
</tr>
<tr>
<td>Crane</td>
<td>Orange</td>
<td>3.91</td>
<td>3.91</td>
<td>3.96</td>
<td>0.117</td>
<td>0.113</td>
</tr>
<tr>
<td>Car</td>
<td>Violet</td>
<td>17.89</td>
<td>40.56</td>
<td>49.31</td>
<td>4.66e-25</td>
<td>0.000812</td>
</tr>
<tr>
<td>Car</td>
<td>Orange</td>
<td>31.48</td>
<td>40.56</td>
<td>51.75</td>
<td>1.98e-12</td>
<td>4.95e-06</td>
</tr>
</tbody>
</table>

The table also shows that Experiential-only was statistically significantly different (p-value < 0.05 in P-value Exp) and slightly better in the simple Chaser domain and was better in the Sheep domain but not different to a significant level in the case of agents based on Orange. Overall, the agents were much better than their original observationally-taught counterparts.
on Orange. However in the Car domain, Observational then Experiential was much better than Experiential-only and averaged more than half a lap longer. It should be emphasized here that these agents not only performed better than Experiential-only but also appear more human-like in both movements and actions. However, as noted in the previous paragraphs, the agents had to forgo some human-like qualities to achieve those performance gains.

Overall, Observational then Experiential learning was able to produce very high performing agents. These agents retained some of the human traits from the Observational training they received, and were able to significantly improve their performance. The additional Experiential phase did add some computer-like qualities, but the agent would still be externally recognized as intelligent. This system consists of two of the major parts of FALCONET, Observational and Experiential learning, and results lend credence to the hypothesis that multiple different learning approaches combined together do better than any single learning approach alone.

7.4.3 Instructional then Experiential

Instructional then Experiential learning is an interesting approach mainly because it is based on an approach that was original assumed to not work. Instructional learning alone takes a randomly created agent and uses the coaching process to only give rewards to the agent. The agent was able to successfully learn some of the simpler domains, but did not perform well in the Car or the Crane domains. The Experiential training should be able to take the basic
agents with some of the human concepts gained through coaching and improve upon them. The experiments were devised to see if the extra Experiential step increased performance while still maintaining the effect of the human influence on the training.

Similar to Observational then Experiential learning, the Instructional then Experiential learning process was seeded with a competent agent for Chaser from the Instructional learning phase. It was found in the Instructional-only learning phase that an agent could be trained from scratch for Chaser. The Experiential phase was able to take this agent and improve upon it. It can be seen in Figure 7.14 that the population as a whole was able to improve to a similar level. This would result in a convergence where not much further improvement could be made. Subjectively, the agents produced through this approach were able to chase at similar performance to Experiential alone, however the agent was less “jerky”

Figure 7.14: Chaser Simulation - Instructional then Experiential
in the movements. The human influence in the original Instructional process appears to have eliminated some of the extreme movements.

Figure 7.15: Sheep Simulation - Instructional then Experiential

Although Instructional learning alone was only able to train an agent to capture approximately half the sheep, the extra Experiential stage was able to improve the best agent to a perfect score for the eight trial runs by generation/iteration number 400. Similar to Chaser, the system was able to bring the average fitness to the point that it was able to capture approximately 15 sheep. The agent retained some of the “darting in” behavior of the original human trainer to capture sheep. The system did have some “jerky” movement when sheep were detected, but not to the same degree of the Experientially-taught agent.

As seen in Figure 7.16, Instructional then Experiential was unable to perform successfully in the Crane simulation. The general operation of the agent was similar to Observational
Figure 7.16: Crane Simulation - Instructional then Experiential

then Experiential for Trainer Orange, where the agent would lower itself to the ground, close the gripper, and run into the closest box it could find. This does not result in a picked up box, but it can be said to be somewhat close to the desired set of actions.

Figure 7.17 shows a dramatic increase in performance as the system produces an agent that makes a discovery around generation/iteration number 400 that increases its performance by a factor of four (4). Originally, it appears that Instructional learning hurt performance as Experiential-only learning is able to obtain a fitness of 25 (2 laps) in the first 200 generations/iterations, but Instructional learning is unable to move beyond 3 (0.25 laps). The original Instructional-only agent on which this multi-phase system is based would originally miss the first left-hand turn. However, it was observed in the Instructional phase that the agent could stay in the lane if it was brought to the main road. At the point of the
performance spike, the agent learned how to make the left turn and used the previously-acquired lane-following information. Additionally, while in the lane, the agent was relatively smooth with respect to actions versus the Experiential-only agent. The agent retained some of the original human training on which it was based.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Ins</th>
<th>Exp</th>
<th>Ins-Exp</th>
<th>P-value Ins</th>
<th>P-value Exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chaser</td>
<td>26.04</td>
<td>28.20</td>
<td>28.08</td>
<td>1.38e-103</td>
<td>4.11e-06</td>
</tr>
<tr>
<td>Sheep</td>
<td>25.35</td>
<td>58.31</td>
<td>58.51</td>
<td>1.3e-42</td>
<td>0.401</td>
</tr>
<tr>
<td>Crane</td>
<td>3.98</td>
<td>3.91</td>
<td>3.94</td>
<td>0.171</td>
<td>0.238</td>
</tr>
<tr>
<td>Car</td>
<td>1.29</td>
<td>40.56</td>
<td>22.69</td>
<td>1.65e-14</td>
<td>5.26e-08</td>
</tr>
</tbody>
</table>

Table 7.8: Comprehensive Table of Instructional then Experiential Fitness

Table 7.8 shows rather conclusively that the additional Experiential phase improved Instructional learning. In each of the three working domains, the Instructional then Experien-
tial learning improved the performance in a significant way (p-value << 0.05 in P-value Ins). However, it was still found that the main contribution of Instructional learning was to help the agent react more smoothly and externally appear more intelligence versus Experiential learning alone when the agents are viewed performing. The agent improved upon themes trained in Chaser and greatly improved Sheep and Car.

In summary, the multi-phase (Instructional then Experiential) system was unable to outperform the Experiential-only agent in Chaser and Car. It was also able to outperform on Sheep, but the means were not different to a significant degree (p-value > 0.05). The system did significantly improve upon Instructional-only. However, the Instructional then Experiential did not in general improve performance over Experiential, but did inject human-like traits from the Instructional training.

For a system based on a learning phase that was not originally thought to work, the system was able to perform at least as well as a human in most cases and produced one of the highest Sheep fitnesses of all approaches. This multi-phase learning also supports the combining of multiple approaches. Experiential practice can actively improve another stage, but still retain training from the previous phase in the weights and structure. This system is very similar to the overall FALCONET approach except that its lacks the original off-line Observational training stage.
7.4.4 Two-Phase Learning Summary

While the benefits of FALCONET comprise the three learning phases as one overall approach (Observational then Instructional then Experiential), other less extensive combinations of two of these approaches were executed and analyzed. The results of Observational then Instructional were inconclusive, as no trend was found in the objective fitness scores. However, it was found through the human trainers that the system did give insight into how the original observation-based agents acted through the haptics. The agent received validation for certain actions, while other actions could be identified for improvement. This could lead to future improvements to the system in which certain areas would be focused on. Both systems that ended with an Experiential training achieved significant improvement. Observational then Experiential learning was able to take an already fit agent produced from Observational learning and significantly improve upon it. This was done while retaining much of the actions originally observed in the test subject. However, as the system improved, it did lose some of the human-like qualities. Nevertheless, in spite of this, it would not be likely to be mistaken for an Experiential-only system. Instructional then Experiential also made significant improvements over the original instruction-taught agents. The brief training session provided an initial frame for the Experiential phase to retain human-like qualities. The human bootstrapping processes are able to give some common sense and appropriate actions to the agent rather than the agent having to discover these from scratch. In the case of Observational then Experiential learning, the system was able to possess the
human qualities and also outperform the Experientially-only-taught agent. The two-phase learning approaches were generally shown to work better than the two single-phase learning approaches from which they were constructed. The final FALCONET approach contains all three of these approaches combined together and is hypothesized to be able to outperform each of the two-phase approaches.

7.5 Three-Phase Learning

As mentioned several times before, FALCONET ideally contains three phases of learning: Observational then Instructional then Experiential. Each phase in the learning process feeds the next by transferring the entire best population of individuals. It has been shown in the previous chapters that each of the phases has been improved by adding Experiential learning. However, another desirable feature was that improved agents retained much of the original human tendencies. The following section focuses specifically on Observational then Instructional then Experiential learning. The comparison of all approaches together is found in Section 7.6. The experiments performed were designed to determine whether the three-phase approach was able to outperform the two-phase approaches and to observe the behavior of the agents while performing the task to see if the multiple steps of human interaction had influence on the actions.
7.5.1 Observational then Instructional then Experiential

The final, three-phase FALCONET approach (Observational then Instructional then Experiential) should contain the same benefits of a two-stage learning process. This method is an extension of the Observational then Experiential learning that was shown to be able to produce very capable agents by taking a human-based agent and allowing it to practice in a simulated environment. The agent for this method uses the population created by Observational then Instructional. During that stage, the results were inconclusive on whether the Instructional phase increased performance, but it did allow the trainer to have a greater understanding of the agents’ actions. The produced starting populations were no worse than Observational-only agents and the increased human interaction should influence the system to be more human-like. This is because a human was able to sort through the agents during the Instructional rewarding process. This full three-phase learning process is tied to the human trainer as two phases of learning intimately involve human trainers. One (Observational) is based on data observed from a human performing the task while another (Instructional) provides a real-time ability for the human to physically interact with the agent population. The performance of the human instructors and their ability to train affects the overall outcome. The experiments were performed to compare the human-like qualities and the performance of the FALCONET agent across all simulations.

To provide a comparison of the human-like qualities of the different agents, two different studies were performed. The first study is designed to compare the joystick move-
ment of a human and a trained agent by analyzing the Y-axis values recorded during the Chaser Simulation (Figure 7.18). This comparison is similar to the method performed for Observational-only and Experiential-only learning. When comparing to the human (red), the agent’s actions (green) do show signs of computer-determined optimal control techniques; however not to the extent of the Experientially-taught agents (refer back to Figure 7.4). These actions are physically expressed on the joystick as jittery motions in place, and not unstable chaotic motions.

The second study is designed to provide a subjective questionnaire to several human trainers in a preliminary blind study. The second study had them visually monitor the agent movement and feel the actions through the haptic interface without the tester being told who or what was performing the actions. The answers were on a scale of one (1) to five (5), where

Figure 7.18: Chaser Simulation - Joystick Movement Comparison
Table 7.9: Blind Study of Human-like Qualities

<table>
<thead>
<tr>
<th>Entity</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human Violet</td>
<td>4.67</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>4.17</td>
</tr>
<tr>
<td>FALCONET Violet</td>
<td>3.17</td>
</tr>
<tr>
<td>Experiential</td>
<td>1.67</td>
</tr>
</tbody>
</table>

five (5) was the best. The results are averaged across responses for the different simulations (excluding Crane) in Table 7.9. The humans observation runs received the highest score with 4.67 while the Experientially-taught agents received the lowest score with 1.67. As a point of comparison, the Observationally-taught agents received a score of 4.17, the highest human-like score for the learning systems. The FALCONET-taught agent received a slightly lower score of 3.17. FALCONET’s human-like score was much higher than Experiential-only learning, but less than Observational-only learning. Overall, the agents produced through the three-stage system, which involved human-based training, were closer to a human than a system that did not involve human interaction (Experiential).

In Figure 7.19, it can be seen that the process was able to improve the performance of the original agents, but those agents were already at a high level of performance to begin with. The agents are visually very effective, following the fleer very closely by both chasing closely and adjusting speed. One of the major advancements seen in this agent was the ability to back up if the fleer is initially behind the agent. Like the other approaches that contained Experiential learning, the augmentation process caused the agent to make very quick actions. These actions produced movements which while appearing acceptable externally, cause the
joystick to move in a jerky fashion that would seem un-human-like. However overall, the agents performed to a high degree of proficiency with smooth movements.

In Figure 7.20, the agent populations were able to fully optimize (capture all 16 sheep) rather early on, for those based on the Orange trainer, within the first 200 generations/iterations. The agents had been trained through Observational then Instructional learning, and that agent already had been able to capture a majority of the sheep. The produced agents seem to have produced a combination of the strategies of the humans and the Experiential learner. This is attributed to the agents making spirals as learned experientially, chasing in sheep on each pass. However, unlike the Experientially-taught agent, the movement was much less jerky when a new sheep was discovered.
Figure 7.20: Sheep Simulation - Observational then Instructional then Experiential

Figure 7.21: Crane Simulation - Observational then Instructional then Experiential
In Figure 7.21, the full system continues the trend of being unable to solve the crane problem. It is believed that a combination of fitness function (Section 8.3.1) and use of contexts (Section 8.2.1) could solve this problem. It was unfortunate that FALCONET was unable to properly operate in this domain, but it will lead to future work.

![Fitness Statistics for Algorithm versus Generations/Iterations](image)

Figure 7.22: Car Simulation - Observational then Instructional then Experiential

In Figure 7.21, the agents for the Car domain were able to optimize to a high value of 57 out of 60 (4.75 laps or 5 laps with honk penalties, see Section 6.1.4). The agents were able to properly make the left turns for merging into lanes and across traffic. One movement notably lacking in this set of agents was the sudden braking with oncoming traffic that was seen in Experiential-only and Observational then Experiential. Because the human agents did not perform that behavior in the Instructional stage, it is possible that the additional human interaction “cured” the agent of this behavior. This is supported by the data that the
three-stage approach which contained Instructional did not have the sudden stops while the
two-stage approach without Instructional did. The agents were able to drive in a competent
manner around the track and was even observed to be able to extract itself out of a traffic
jam caused by aggressive driving.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Subject</th>
<th>Obs-Ins</th>
<th>Obs-Exp</th>
<th>Full</th>
<th>P-value OI</th>
<th>P-value OE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chaser</td>
<td>Violet</td>
<td>26.28</td>
<td>28.15</td>
<td>28.23</td>
<td>6.59e-63</td>
<td>0.00114</td>
</tr>
<tr>
<td>Chaser</td>
<td>Orange</td>
<td>27.00</td>
<td>27.84</td>
<td>27.80</td>
<td>6.9e-35</td>
<td>0.253</td>
</tr>
<tr>
<td>Sheep</td>
<td>Violet</td>
<td>45.17</td>
<td>36.66</td>
<td>57.55</td>
<td>1.12e-10</td>
<td>3.07e-27</td>
</tr>
<tr>
<td>Sheep</td>
<td>Orange</td>
<td>45.09</td>
<td>57.66</td>
<td>56.04</td>
<td>2.63e-09</td>
<td>0.105</td>
</tr>
<tr>
<td>Crane</td>
<td>Violet</td>
<td>3.95</td>
<td>3.99</td>
<td>4.00</td>
<td>0.14</td>
<td>0.413</td>
</tr>
<tr>
<td>Crane</td>
<td>Orange</td>
<td>3.94</td>
<td>3.96</td>
<td>3.94</td>
<td>0.502</td>
<td>0.318</td>
</tr>
<tr>
<td>Car</td>
<td>Violet</td>
<td>14.92</td>
<td>49.31</td>
<td>48.46</td>
<td>1.27e-25</td>
<td>0.317</td>
</tr>
<tr>
<td>Car</td>
<td>Orange</td>
<td>36.48</td>
<td>51.75</td>
<td>48.06</td>
<td>2.36e-05</td>
<td>0.0638</td>
</tr>
</tbody>
</table>

Table 7.10: Comprehensive Table of Full System Fitness

Table 7.10 gives a comparison of the full three-phase learning process versus two of the
two-phase approaches that made up the full approach. For space considerations and the
performance issues, Instructional then Experiential is not listed in the table. The first observ-
ervation is that the Experiential phase of learning was able to improve the base Observational
then Instructional agent in each of the working simulations to a significant degree (p-value
<< in P-value OI). The Experiential learning performance gains support the multi-phase
approach obtaining close to 90% of the optimal score in each domain (30 for Chaser and 60
for the other domains).

Also listed in the table is the comparison to Observational then Experiential. The main
difference between these two approaches is the extra Instructional phase. As seen in Sec-
tion 7.4.1, the Instructional phase was not conclusively shown to improve Observational-only learning. This experiment shows similar results with the fitnesses being statistically indistinguishable between the two approaches in the majority of the domains (p-value > 0.05 in P-value OE). However, the two notable exceptions are for the Violet test subject whose Chaser and Sheep agents were able to be significantly different (p-value < 0.05) and slightly better. The coaching provided by Violet was able to help build a better agent. This is seen most in the Sheep domain where the coaching was able to overcome the inconsistencies seen in the Observational then Experiential agent.

Therefore, the full FALCONET approach was able to outperform the other multi-stage methods that were contained within it or be statistically not different to a 95% confidence level. The actual operation of the agents was observed to react similarly to a human in the Chaser, Sheep, and Car domains. However, the system still was unable to solve the Crane domain. The agents, who were created with multiple phases of human learning, appeared to be more consistent and did not make as many jerky movements as the other approaches. The results support that the three-stage approach of FALCONET can produce high performing, human-like agents.

7.6 Final Analysis Comparison

As a final comparison, the fitness scores for every single approach are compiled into a single sorted table for each domain. The table includes all learning methods with one, two, or all
three approaches along with the original human scores and the preprogrammed agents hand
optimized for each domain. The data in this table include the mean, standard deviation, and
p-value for significance versus the best in order to provide full comparison. The data in these
tables are based on taking the agent with the best experiential fitness for each approach and
running it 100 times on randomly generated scenarios.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observational Instructional Experiential Violet</td>
<td>28.23</td>
<td>0.15</td>
<td>0.102</td>
</tr>
<tr>
<td>Experiential</td>
<td>28.20</td>
<td>0.18</td>
<td>0.00114</td>
</tr>
<tr>
<td>Observational Experiential Violet</td>
<td>28.15</td>
<td>0.21</td>
<td>1.83e-09</td>
</tr>
<tr>
<td>Instructional Experiential</td>
<td>28.08</td>
<td>0.19</td>
<td>7.22e-16</td>
</tr>
<tr>
<td>Observational Experiential Orange</td>
<td>27.84</td>
<td>0.40</td>
<td>2.61e-59</td>
</tr>
<tr>
<td>Observational Instructional Experiential Orange</td>
<td>27.80</td>
<td>0.44</td>
<td>4.85e-16</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>27.03</td>
<td>0.38</td>
<td>5.17e-112</td>
</tr>
<tr>
<td>Observational Instructional Orange</td>
<td>27.00</td>
<td>0.19</td>
<td>5.22e-144</td>
</tr>
<tr>
<td>Hand-Made</td>
<td>26.48</td>
<td>0.18</td>
<td>2.37e-130</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>26.67</td>
<td>0.19</td>
<td>6.59e-63</td>
</tr>
<tr>
<td>Observational Instructional Violet</td>
<td>26.28</td>
<td>0.54</td>
<td>1.46e-101</td>
</tr>
<tr>
<td>Instructional</td>
<td>26.04</td>
<td>0.33</td>
<td>3.4e-75</td>
</tr>
<tr>
<td>Human Orange</td>
<td>24.61</td>
<td>0.73</td>
<td>1.31e-103</td>
</tr>
<tr>
<td>Human Violet</td>
<td>22.72</td>
<td>0.62</td>
<td>1.31e-103</td>
</tr>
</tbody>
</table>

Table 7.11: Final Chaser Table Sorted

For the Chaser domain, Table 7.11 show the best agent on average was produced through
the FALCONET three-stage approach for test subject Violet. This method was significantly
different than all the other approaches except for the Experiential-only agent and was nu-
merically better. The significance of this feat is that the FALCONET agent was the best
but still had the human-like movements. Another interesting item to note is that the worst
scores were actually the human test subjects. They were unable to have the reaction time
or consistency of their computer counterparts. Additionally, the majority of the learning systems were able to outperform even the hand-made system in this domain. The control systems created for heading control, velocity control, and fleer prediction were able to outperform a reactive only system. It can also be seen that the trainer mattered. The agents based on Violet plus an Experiential stage were able to outperform the agents based on Orange. The major reason for this is the inconsistency of the Violet test subject. These observations gave the agent more of the domain to learn from and mistakes to fix. Violet still provided competent actions in the data, but the addition of showing the agent additional tactics appears to improve performance.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instructional Experiential</td>
<td>58.51</td>
<td>4.49</td>
<td></td>
</tr>
<tr>
<td>Experiential</td>
<td>58.31</td>
<td>6.43</td>
<td>0.401</td>
</tr>
<tr>
<td>Observational Experiential Orange</td>
<td>57.66</td>
<td>5.80</td>
<td>0.125</td>
</tr>
<tr>
<td>Observational Instructional Experiential Violet</td>
<td>57.55</td>
<td>7.18</td>
<td>0.13</td>
</tr>
<tr>
<td>Observational Instructional Experiential Orange</td>
<td>56.04</td>
<td>11.44</td>
<td>0.0235</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>46.71</td>
<td>15.58</td>
<td>2.29e-11</td>
</tr>
<tr>
<td>Observational Instructional Violet</td>
<td>45.17</td>
<td>16.54</td>
<td>1.84e-12</td>
</tr>
<tr>
<td>Observational Instructional Orange</td>
<td>45.09</td>
<td>13.75</td>
<td>4.67e-16</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>44.84</td>
<td>16.73</td>
<td>1.07e-12</td>
</tr>
<tr>
<td>Human Orange</td>
<td>43.52</td>
<td>9.09</td>
<td>5.28e-31</td>
</tr>
<tr>
<td>Hand-Made</td>
<td>37.42</td>
<td>13.46</td>
<td>2.45e-29</td>
</tr>
<tr>
<td>Observational Experiential Violet</td>
<td>36.66</td>
<td>14.01</td>
<td>3.62e-29</td>
</tr>
<tr>
<td>Human Violet</td>
<td>29.71</td>
<td>11.07</td>
<td>3.05e-50</td>
</tr>
<tr>
<td>Instructional</td>
<td>25.35</td>
<td>14.81</td>
<td>1.3e-42</td>
</tr>
</tbody>
</table>

Table 7.12: Final Sheep Table Sorted

For the Sheep domain, Table 7.12 shows that the Instructional then Experiential agent was able to produce the best agent on average. It was not significantly different (p-value
than Experiential, Observational then Experiential Orange, and Observational then Instructional then Experiential Violet. Observational then Instructional then Experiential Orange was worse and statistically different at a 95% confidence level, but not a 99% level (p-value > 0.01) and the next best was significantly worse to a much greater degree. The Instructional than Experiential victory is very interesting because while it produced a very fit agent, it was not statistically different than Experiential-only learning. This could mean that the Instructional phase had no effect. However by observing the behavior of the Instructional then Experiential agent, the actions of the controller are much smoother and this could be attributed to the human influence. The other item to note is that the FALCONET agent’s performance was ranked in the top five agents. This shows that the process can also produce high performing agents and these agents behave even more like the human counterparts. In this domain, the human test subjects are also in the bottom half of the table.

For the Crane domain, Table 7.13 shows that no learning agent were able to complete the task of picking up the boxes. In this domain, the human was the best performer being significantly better than all but the hand-made state machine agent. This domain suggests that there are some tasks that humans excel at: operating multiple joysticks and planning for multiple targets. Both human test subjects could, by the problem definition and practice, properly operate in this domain. Humans have the ability to use previous knowledge of similar event and can break down the tasks properly. The approximately 25 years the human trainers have been doing motor-skill related tasks has given them an advantage over a training system. However, this is a discussion outside the scope of this research.
<table>
<thead>
<tr>
<th>Approach</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human Orange</td>
<td>42.15</td>
<td>6.59</td>
<td></td>
</tr>
<tr>
<td>Hand-Made</td>
<td>41.64</td>
<td>7.15</td>
<td>0.302</td>
</tr>
<tr>
<td>Human Violet</td>
<td>38.56</td>
<td>3.44</td>
<td>1.72e-06</td>
</tr>
<tr>
<td>Observational Instructional Experiential Violet</td>
<td>4.00</td>
<td>0.33</td>
<td>1.11e-78</td>
</tr>
<tr>
<td>Observational Experiential Violet</td>
<td>3.99</td>
<td>0.30</td>
<td>1.18e-78</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>3.98</td>
<td>0.29</td>
<td>1.19e-78</td>
</tr>
<tr>
<td>Instructional</td>
<td>3.98</td>
<td>0.29</td>
<td>1.19e-78</td>
</tr>
<tr>
<td>Observational Experiential Orange</td>
<td>3.96</td>
<td>0.29</td>
<td>1.13e-78</td>
</tr>
<tr>
<td>Observational Instructional Violet</td>
<td>3.95</td>
<td>0.32</td>
<td>1.01e-78</td>
</tr>
<tr>
<td>Observational Instructional Orange</td>
<td>3.94</td>
<td>0.32</td>
<td>9.83e-79</td>
</tr>
<tr>
<td>Observational Instructional Experiential Orange</td>
<td>3.94</td>
<td>0.30</td>
<td>1.04e-78</td>
</tr>
<tr>
<td>Instructional Experiential</td>
<td>3.94</td>
<td>0.30</td>
<td>1.04e-78</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>3.91</td>
<td>0.30</td>
<td>9.67e-79</td>
</tr>
<tr>
<td>Experiential</td>
<td>3.91</td>
<td>0.29</td>
<td>9.95e-79</td>
</tr>
</tbody>
</table>

Table 7.13: Final Crane Table Sorted

Works chapter discusses and performs several experiments on several potential reasons why the agents could not learn the Crane simulation including: the fitness function, the multiple state/contextual nature of the problem, and the possible difficulty in dealing with additional joystick/actions with one large network. Each of these items represents potential future work that should be investigated.

For the Car domain, Table 7.14 shows that the learning approaches that contained human observation data (Observational) and practice (Experiential) were significantly different to a 95% confidence level (p-value < 0.05) and able to outperform the other learning approach, humans, and even the hand-made agent. Overall, it appears that both Observational then Experientially-taught agents did better than the full FALCONET-trained agents on average; however it was not significantly different (p-value > 0.05). As mentioned in Section 7.5.1,
<table>
<thead>
<tr>
<th>Approach</th>
<th>Mean</th>
<th>StdDev</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observational Experiential Orange</td>
<td>51.75</td>
<td>11.61</td>
<td></td>
</tr>
<tr>
<td>Observational Experiential Violet</td>
<td>49.31</td>
<td>16.84</td>
<td>0.118</td>
</tr>
<tr>
<td>Observational Instructional Experiential Orange</td>
<td>48.46</td>
<td>18.00</td>
<td>0.0638</td>
</tr>
<tr>
<td>Observational Instructional Experiential Violet</td>
<td>48.06</td>
<td>19.82</td>
<td>0.0556</td>
</tr>
<tr>
<td>Hand-Made</td>
<td>47.26</td>
<td>17.60</td>
<td>0.0175</td>
</tr>
<tr>
<td>Experiential</td>
<td>40.56</td>
<td>21.51</td>
<td>4.95e-06</td>
</tr>
<tr>
<td>Human Orange</td>
<td>39.38</td>
<td>7.41</td>
<td>2.71e-16</td>
</tr>
<tr>
<td>Observational Instructional Orange</td>
<td>36.48</td>
<td>22.39</td>
<td>5.64e-09</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>31.48</td>
<td>24.06</td>
<td>1.98e-12</td>
</tr>
<tr>
<td>Human</td>
<td>26.64</td>
<td>7.89</td>
<td>1.35e-41</td>
</tr>
<tr>
<td>Instructional Experiential</td>
<td>22.69</td>
<td>24.13</td>
<td>1.16e-20</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>17.89</td>
<td>20.36</td>
<td>6.22e-31</td>
</tr>
<tr>
<td>Observational Instructional Violet</td>
<td>14.92</td>
<td>19.17</td>
<td>1.05e-36</td>
</tr>
<tr>
<td>Instructional</td>
<td>1.29</td>
<td>0.73</td>
<td>7.28e-67</td>
</tr>
</tbody>
</table>

Table 7.14: Final Car Table Sorted

both sets of agents had similar performance, but the full FALCONET agents appeared more human-like in behavior with respect to dealing with oncoming traffic (by not stopping). The Car problem is a complex domain with moving traffic, multiple actuations (steering and gas/brake), and complex non-linear physics model. In this situation it was shown that the Experiential agent alone could not learn on its own to the level of agents that included human influence, and the agent that was produced did not perform in a human-like fashion by acting in a jerky manner. The domain also shows that Observational-only learning was unable to train to even half of the perfect score (60), but was able to operate at approximately a human level.
7.7 Approach Experimental Summary

In a summary of the above tables and the total of approach experiments, the results fully support the multi-stage learning process. No individual single phase approach was able to produce an agent that consistently performed well across the three domains (excluding Crane). Observation alone does not produce the best performing agents, but can produce an agent that performs close to or better than a human. The observation-based agent does act the most human-like out of all the learning agents. Therefore, the Observational-only approach does have a place in mimicking human behavior and acting as entities in simulators, but it does not produce high performing agents that would be desirable for performing a task.

The Instructional-only agent was consistently near the bottom of the list for performance. This was to be expected, as it was not meant to be a complete learning method on its own. Instructional learning was devised to augment other learning approaches by providing a validation procedure and to help the human trainers. However, from the results of Chaser and Sheep, it was shown that agents produced through Instructional learning can in fact teach a randomly-created agent to function in simple domains.

Finally, Experiential-only learning is known to be able to produce agents that perform well in simple domains. However, it was shown that practice alone makes it difficult to optimize in a complex domain such as Car. Additionally, the Experiential agent performed in a very un-human-like manner that cannot always be perceived as intelligent externally.
The haptic-based joysticks were able to be used to determine intelligence of Experiential-only agents by the physical actions of the agent as well as the agent’s motion (see Figure 7.4). The agent also lacked the level of common-sense expected in an intelligent entity.

Each of the single-phase methods could generally produce competent agents in three of the domains (excluding Crane), but the multi-stage method generally resulted in improvement increases over the original single-stage approaches on which they were based. Adding an additional practice stage (Experiential) was able to improve the performance of every agent. The original Instructional agent, which was usually the worst performer, was able to significantly improve in every working domain with practice. During this training, the original human training that rewarded the population for proper actions and motions was able to create an initial starting point that was able to be improved but still perform in a human-like manner. The Observational then Experiential approach exhibited this trait even more pronouncedly by producing one of the highest performing agents, but still acted human-like by reducing the twitching effects introduced by Experiential learning. By bootstrapping an agent with human knowledge, the agent is given a starting point with proper actions and strategies. The practice gained in Experiential learning then allows those systems to improve beyond the human.

The Instructional stage on top of the Observational-taught agent did not initially show any performance improvements. The human coaches were able to gain insights from the haptic joysticks on what the agent was doing wrong and provide validation of the already existing agents. The Instructional phase was meant to identify situations in which the agent
has not properly learned and correct them. The agents are sorted by the reward from the coach. Because the agent with the most reward will rise to the top, the coach can effectively select the best performer from the population. The coach is integral to this process as poor coaching can decrease performance.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Average Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observational Instructional Experiential Violet</td>
<td>3</td>
</tr>
<tr>
<td>Observational Experiential Orange</td>
<td>3</td>
</tr>
<tr>
<td>Experiential</td>
<td>3.33</td>
</tr>
<tr>
<td>Observational Instructional Experiential Orange</td>
<td>4.67</td>
</tr>
<tr>
<td>Instructional Experiential</td>
<td>5.33</td>
</tr>
<tr>
<td>Observational Experiential Violet</td>
<td>5.67</td>
</tr>
<tr>
<td>Observational Instructional Orange</td>
<td>8</td>
</tr>
<tr>
<td>Observational Violet</td>
<td>8.33</td>
</tr>
<tr>
<td>Hand-Made</td>
<td>8.67</td>
</tr>
<tr>
<td>Observational Orange</td>
<td>9</td>
</tr>
<tr>
<td>Human Orange</td>
<td>10</td>
</tr>
<tr>
<td>Observational Instructional Violet</td>
<td>10.33</td>
</tr>
<tr>
<td>Human Violet</td>
<td>12.33</td>
</tr>
<tr>
<td>Instructional</td>
<td>13.33</td>
</tr>
</tbody>
</table>

Table 7.15: Ranked Results on the Three Working Domains

The final FALCONET approach (the three other approaches in sequence) was consistently able to produce high performing agents that would behave in a human-like manner. For the successful testbed domains (Chaser, Sheep, and Car), the Observational then Instructional then Experiential agents were able to always score in the top five and the mean of performance fitness not be considered statistically different than the best performing agent at a 95% confidence level. One way to provide a final summary of performance for each approach is to calculate the average rank in each of the three working domains (Table 7.15). The average
The natural learning process of a human is believed to be to observe someone already competent at the task (Observational), then receive coaching to reduce inconsistencies (Instructional), and finally to practice to improve performance (Experiential). It was hypothesized that a learning agent that was trained in this same process would be better than just learning from any of the individual learning processes. The experimentation conducted in this chapter has supported the hypothesis. The agent produced through FALCONET was able to outperform or not be statistically different to a 95% confidence level than any method in the power set of subcomponents, although to a practical significance it would be difficult to differentiate between the top methods in terms of performance. Additionally, the top methods with Observational training were able to retain human-like qualities while performing tasks versus Experiential-only which did not involve human interaction and tended to act in an erratic fashion.
To accomplish this task, the PIGEON Alternate algorithm was identified to be able to learn in each approach by being a hybrid of Neural Networks, Genetic Algorithms, and Particle Swarm Optimization. PIGEON was also better than all its subcomponents in producing consistent high performing agents.

The haptic interface was also shown to help the training process by giving the human a better understanding of the agents’ actions and provide a mechanism to improve and validate the performance of the agents.

The total combination of the FALCONET approach and the PIGEON method has proved the hypothesis and increased the state of art for creating agents for motor-skill tasks.
CHAPTER 8
FUTURE WORK

This chapter covers various observations of problems and potential solutions to the approach created in this dissertation. There are many alternatives and combinations of approaches that could be run for experimentation but a defined subset was selected as ample to support the hypothesis. Some future work topics stem from alternative approaches developed after the main experimentation was well underway and others were simply out of the scope of research.

8.1 Reduce Effort

8.1.1 Computer-Based Training

One concept that was evaluated was a machine-to-machine teaching process. It was identified during the Observational and Instructional runs that there was technically no requirement for the observations to be based on a human. A hand-made agent, using whatever paradigm desired, could be created to accomplish the task. The hand-made agent does not need to be optimal in any way, but competent enough to be able to give the learning agent “common sense” actions to follow. The idea is that the hand-made agent cannot learn, and therefore
cannot get better. However, the learning agent bootstrapped with the hand-made agent could perform at least as well and continue to get better with an Experiential stage. This process would remove the human from the loop for observation data and could still make high performing agents. The only issue is with the removal of the human element, it is possible the agents would not act in a human-like fashion. It would be entirely based on how the original hand-made agent was constructed.

8.1.2 Stopping Criteria

It was identified that the agents produced by PIGEON can continue to learn after a large number of generations/iterations. However in some cases, the population of agents stagnated after a period of time or did not significantly gain in performance. There are several stopping criteria that exist in the literature which could have been used. A cross-validation set, which did exist for the observation case, could have been used to prevent the system from over-fitting the training data such as in (Prechelt 1998). Since Observational learning was supervised, the learning process should stop if it was found that the system started to decrease performance on the cross-validation set. Another option would be to develop a formula for cost-benefit analysis if it is worth continuing to learn. In (Tan 1993), the author identifies the weighing “accuracy versus efficiency” for a decision-tree based controller for a robot. Since the system operates in real-time, there is a trade-off on the size and computational cost in finding a solution. In (Aytug and Koehler 1996), the authors analyzed the
convergence behavior of a GA, although fixed size in their case, to “derive bounds on the number of iterations required” to gain an optimal solution. The gradient of convergence may also be used to discover when to stop, such as in (De Jong 1975). Finally, it might just be determined that a solution is “good enough” such as in (Larranaga et al. 1997) when tackling NP-Hard problems. The true optimal solution may never be found, but a close solution that satisfies a second set of criteria may be. Adding a stopping criterion could have reduced the computational time of FALCONET by eliminating extraneous generations/iterations from being evaluated. However for this dissertation, the generations were fixed to provide a better comparison between different approaches or different methods.

8.2 Architecture Changes

8.2.1 Using Contexts

It has been previously identified in (Gonzalez and Ahlers 1994), that CxBR can be used with tactical agents by dividing the problem space into contexts. Each context only contains the necessary information that is needed and related actions to perform. CxBR thus reduces the computational space by limiting what an agent needs to be concerned about or do at a given time. For example in the Crane simulation, a context could be developed for locating boxes, another for picking up boxes, and yet another for returning them to the destination. Within the “picking up box” context, the agent does not need to worry about the other
boxes or even moving in the X-Y plane. This reduction in both inputs and outputs reduces the computation and distractions that could hurt the learning process.

However, transition rules need to be developed in order to determine which context should be active. These rules could be preprogrammed for a given domain, but this would increase the load on programmer. In (Fernlund and Gonzalez 2004), the author used GP to learn the transitions and actions for a car driving agent. The dataset had to be manually partitioned to identify examples of when a particular context is active. More recently, an extension for Fernlund’s work has been created to eliminate the manual partitioning (Trinh 2009). This would be like having many sub-agents that are able to accomplish a specific sub-task. The learning procedure would not be affected by the amount of time spent in each sub-task and only optimize for the majority case. The sub-agents for “picking up box” would only be learning on observed data of when a box was being picked up. After the sub-agents were constructed, the transition rules are developed when each context should be active. A co-evolution procedure of combining together different sub-agents and sentinel rules is done to produce a single overall agent with several parts. This procedure could potentially have been directly used with FALCONET on the Observational stage. PIGEON could be used to calculate the actions and transition rules in order to produce the sub-agents.

Using contexts could be a potential solution to the crane problem. It had already been determined difficult to even write an agent by hand that did not contain at least states. The sub-agents developed in each context could have used the observed data to learn a sub-task. CxBR was not used in this dissertation for several reasons. CxBR is related
to the implementation of the agent architecture and does not affect FALCONET in any way. The purpose of the dissertation was to test the multi-stage learning process and the haptics-based Instructional approach. CxBR was not related to the variable being tested (Observational, Instructional, and Experiential learning) and could have complicated the experimental process. Additionally, the co-evolution phase and the need to evolve transition rules would have significantly increased computational time when other algorithms (NEAT, PSO, and PIGEON) were being identified. Increasing the number of experimental tests for identifying an appropriate algorithm not just for the action but for the transition rules as well. Finally, the overall computation time would have increased by having to create all the sub-agents and evaluate them together. Even with the computation time, CxBR is still an attractive addition to be used in FALCONET. With optimizations and increases in computer speed, it would be worthwhile investigation, but outside the scope of this research.

8.2.2 Modularity and Concurrency

Another implementation change is to use modularity and concurrency in the production of agents. In contrast to CxBR which only has a single context active at a given time and sub-agents that are related to each context, a modular agent with concurrency would have multiple sub-agents that each only control a subset of the actions. A module would be developed that might just control the X-Y movement of the crane while another module controls the Z-axis and gripper. The data could be partitioned to train each different aspect
of the agent separately. This can then be used with a state-machine and rules of when to turn on and off different parts of the agent. This process such as the one defined in (Cremer et al. 1995) uses a Hierarchical Concurrent State Machine (HCSM) to “model behaviors that involve attending to multiple concurrent concerns and arbitrating between conflicting demands for limited resources.” A co-evolution procedure would be required to combine together these multiple sub-agents which were developed separately.

There are attractive notions within the modularity and concurrency aspect. Making smaller networks for specific actions and controllable items could allow the system to optimize better for each individual output to the simulation rather than chose an output that is sub-optimal for each but relatively good overall. However, there could be potentially complex interactions between the modules since they are running at the same time. As for training, the modular design also allows for interesting changes to FALCONET when the human coach could focus on just one set of actions at a time when training in Instructional learning. For example, the coach may have identified that the X-Y module for the Crane simulation functions properly. He can then just provide rewards and punishment to the gripper sub-agent module without having to worry about improperly teaching the X-Y control. The individual focus would reduce the workload on the human coach and allow more direct application of reward. Algorithmically, PIGEON could be used to construct each sub-agent and rule system. However, the computational cost would be higher and a distraction from the main goal being tested. Modularity and concurrency are interesting future options, but outside the scope of this dissertation.
8.3 Improve Learning

8.3.1 Alternative Crane

Several different concepts for an alternative Crane approach were proposed over the course of the dissertation that may help the learning process. It was mentioned previously that Experiential learning can be improved by using sub-goals and partial credit in the fitness function in order to differentiate between agents in the early stages and hint the system in the proper direction. The reader can recall that the fitness in Crane is primarily based on the number of boxes returned to the destination area. This is a discrete value because there are only ten boxes and therefore only eleven scores from 0 to 60. An additional score was added to account for those boxes brought toward the destination, but not successfully brought all the way. This partial credit was intended to reward those agents that gained the concept of picking up a box but did not return it to the final destination. However after viewing the results of all learning phases, it was potentially wishful thinking to expect that the agents would successfully pick up a box to receive the partial credit.

Other forms of partial credit were created after the fact to discover if the learning systems could potentially succeed without architectural changes to the algorithms such as the introduction of contexts or modularity. Due to the conclusions of the dissertation, it was found that agents produced by the Observational then Instructional approach were the most cons-
sistent as the introduction phase to Experiential learning. For the purposes of comparison, it will be the approach used for experimentation.

The first type of extra credit introduced was the introduction of points for the amount of time in which the agent could hold a box, however, since the system had not learned to pick up a box this would likely not have helped. Another was a distance metric from the crane to the box to influence the agent to move toward the box. This is a desired motion that must happen before a box can be picked up. The results showed that the agent was still unable to pick up a box. The learned behavior of the agent would be to lower itself to the ground and ram into the closest box. This would make sense since the sub-goal, which is scored based on distance to a box, is being done to receive the points, but the overall goal is not being accomplished. The next stage would be to give points for being above the box, then possibly points for being above then lowering onto the box. At some point in this process, the fitness function is potentially just telling the exact motion the agent should do and the system might as well be preprogrammed. There exist a cutoff point in which the system is no longer discovering how to do the task.

Another concept was to modify the system to incorporate concepts from Section 8.2.1 involving contexts. It was found in Section 7.4.1 on Observational then Instructional training that the human coaches identified that the agents could return the boxes if brought right above or if they trainer picked up the box for the agent. As a concept similar to CxBR in Section 8.2.1, the agents were split apart in order to learn only an individual context, returning the box once picked up. The other context sub-agent was borrowed from Section
since the preprogrammed agents were able to successfully pick up boxes already. An experiment was run in which the preprogrammed agent would select the box and move to pick it up. The transition rule was if there was contact between the box and the crane. Therefore, the learning agent had to properly return the box to the destination and release the box in the right location. Once the box was released, the other context based on the preprogrammed agent would take over. Using this contextual separation, the system was able to appropriately pick up and drop off boxes.

As seen in Figure 8.1, the Context split agent that contained half preprogrammed and half learning system was able to improve to a high level of performance averaging almost 54 (9 of 10 boxes) in the eight Experiential runs. There was even a case in generation/iteration 934 in which the best agent returned all ten boxes for all eight of the experimental runs.
Observing the runs of the best agent, the preprogrammed part of the agent would bring the crane to the closest box and lower down the crane arm until it touched the box. The learning agent would then take over control which would grip the box, move toward the destination, raise up the crane, and, when in the destination, drop off the box.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Mean</th>
<th>StdDev</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crane</td>
<td>19.61</td>
<td>13.22</td>
<td>3.35</td>
<td>51.73</td>
</tr>
</tbody>
</table>

Table 8.1: Crane FALCONET Context Split

Table 8.1 lists that over the course of 100 trial runs of the best agent, the average fitness score of 19.61 (3.2 out of 10 boxes) was achieved. While this score is not better than either the humans or the hand-made agent in Table 7.13, it is significantly better than all other learning systems. The ability to pick up and return a box is a significant milestone. The system as currently programmed could not build a single agent that could solve the problem, but by splitting up the problem into contexts, the system could achieve some positive results.

The ability to control the “return and drop off context” shows that the agent did learn how to solve part of the problem and that splitting up the problem into contexts can help the learning process. Future work would be to learn the actions of all the different contexts and, even later, learn the transition rules between the contexts. Additional work needs to be done to investigate how to properly combine CxBR and FALCONET, but the initial results suggest that contexts would be beneficial in complex problems.
8.4 Extensions

8.4.1 Training Human Subjects

An extension of the agent learning process is to use the FALCONET process but interchange
the human and computer. To develop an approach that can be used to teach a task or skill to
an agent, which can subsequently and seamlessly teach other humans the same task or skill.

The concept is that a human expert’s time is finite and it can be expensive to relocate a single
individual to train others, but an agent based on the expert could be easily disseminated
around the country. The same procedure that FALCONET uses to teach a computer agent
to do a motor skill task by using haptic-based force-feedback and Joysticks can be used to
train a person. It has already been shown in (Gillespie et al. 1998) that showing a human
the optimal way to perform, or in this case a near optimal agent, can significantly reduce
learning time. Additionally, it has also been shown in (Bayart et al. 2005) that a haptic
force can increase the learning process. The roles are simply reversed where the agent still
provides the force-feedback, but now the human student is learning. The techniques used to
validate and grade the agent can be used on a human. This concept, while outside the scope
of this dissertation, would provide an interesting area of research.
APPENDIX A

LEARNED NETWORKS

Figure A.1: Best FALCONET Network for Chaser
Figure A.2: Best FALCONET Network for Sheep
Figure A.3: Best FALCONET Network for Car
APPENDIX B
QUESTIONNAIRES

On a Scale of 1 to 5
1=Strongly Disagree, 2=Disagree, 3=Neutral, 4=Agree, 5=Strongly Agree
V=Violet, O=Orange

<table>
<thead>
<tr>
<th>Question</th>
<th>V</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do you feel the observational agent’s actions were similar to your own...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...in the Chaser Domain?</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>...in the Sheep Domain?</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>...in the Crane Domain?</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>...in the Car Domain?</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Do you believe you could train the agent only haptically?</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Did the haptics give a greater understanding of the agent’s decisions?</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Do you feel the agent improved over the haptic training?</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Do you feel that the haptics helped in the training?</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Do you feel the observational agent performed in a human-like manner visually?</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Do you feel the observational agent performed in a human-like manner haptically?</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Do you feel the experiential agent performed in a human-like manner visually?</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Do you feel the experiential agent performed in a human-like manner haptically?</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Table B.1: Questionnaire Responses
LIST OF REFERENCES


286


